Facultad de Ciencias de la Administración
Escuela de Ingeniería de Sistemas

Tutorial de desarrollo y distribución de aplicaciones para dispositivos iOS utilizando el IDE XCode.

Tesis previa a la obtención del título de Ingeniero en Sistemas.

Alumno:
Fausto Xavier Salazar Jara

Director:
Ing. Pablo Fernando Pintado Zumba

Cuenca, Ecuador
Junio - 2013

Índice de contenidos

ÍNDICE DE ILUSTRACIONES ..... **iv**

RESUMEN ..... **xiii**

ABSTRACT ..... **xiv**

INTRODUCCIÓN: **1**

Capítulo 1: Herramientas a Utilizar **3**
1.1 IDE XCode **4**
1.2 Interface Builder **7**
1.3 iOS Simulator **9**
1.4 Servicio Web de Amazon: AWS Simple DB **11**
1.5 iBooks Author **15**

Conclusiones **17**

Capítulo 2: Tutorial de desarrollo y distribución de aplicaciones iOS utilizando el IDE XCode **18**
2.1 Cómo instalar el IDE XCode y sus componentes en Mac OS X Lion **19**
2.1.1 Suscripción como desarrollador de Apple 21
2.2 Cómo crear un nuevo proyecto **36**
2.3 Análisis de tipos de posibles ficheros de un proyecto **42**
2.3.1 Cómo agregar un fichero a un proyecto 57
2.3.2 Asignar un ícono para la aplicación 66
2.3.3 Agrupamiento de ficheros dentro de un proyecto 70
2.4 Cómo crear interfaces gráficas de usuario **72**
2.4.1 Análisis de componentes comunes 73
2.4.2 Construcción de una interface gráfica de usuario 91
2.4.3 Análisis de plantillas prediseñadas para un proyecto 107
2.4.4 Rotación de interfaces y sus componentes 128
2.5 Asignación de variables y eventos a componentes en *View Controllers* y *Table View Controllers* **146**
2.5.1 Asignación de variables a componentes de GUI 147
2.5.2 Asignación de métodos a componentes de GUI 154
2.5.3 Eventos comunes de Views 160
2.5.4 Eventos comunes de Table Views 162
2.6 Conexión de vistas en Storyboards **165**
2.6.1 Implementación de un *Navigation Controller* (Control de Navegación) 166
2.6.2 Implementación de un Tab Bar Controller (Control de Pestañas) 173
2.6.3 Análisis de tipos de transiciones 179
2.7 Cómo realizar persistencia de datos **184**
2.7.1 Estructura de Core Data 185
2.7.2 Creación de una aplicación que implementa *Core Data* 187
2.8 Cómo conectar una aplicación con redes sociales **204**
2.8.1 Cómo añadir frameworks 205
2.8.1.1 Análisis de frameworks comunes 208
2.8.2 Implementación del framework de Twitter 212
2.8.3 Implementación del framework de Facebook 216
2.9 Cómo probar y depurar una aplicación en dispositivos físicos **223**
2.9.1 Pasos para probar una aplicación en un dispositivo iOS físico 226
2.9.2 Compilación y depuración de aplicaciones 238
2.10 Uso de archivos multimedia **241**
2.10.1 Implementación de fotos de galería y cámara 242

2.10.1.1 Uso del componente UIImagePickerController....243
2.10.2 Acceso al micrófono del dispositivo....246
2.11 Acoplamiento de aplicaciones para que funcionen en iPads....250
2.12 Seguridad en aplicaciones iOS....256
2.13 Cómo subir una aplicación a la tienda AppStore....262
2.13.1 Consideraciones para evitar el rechazo de aplicaciones....264
2.13.2 Portal iTunes Connect....267
2.13.3 Acuerdo de contratos, impuestos y banca....269
2.13.4 Solicitud de certificado y perfil de distribución....273
2.13.5 Requerimientos para administrar aplicaciones en iTunes Connect....277
2.13.6 Administración de aplicaciones en iTunes Connect y distribución....280
2.14 Obtener calificaciones de usuarios para la aplicación....294
2.14.1 Motivar a usuarios a valorar positivamente la aplicación mediante AppiRater....295
2.14.2 Implementación de AppiRater....297
2.15 Identificación única de dispositivos iOS....302
2.16 Implementación del servicio Amazon Web Service: Simple DB....307
2.16.1 Conexión de una aplicación con el servicio AWS Simple DB....308
2.16.2 Seguridad en el servicio web Amazon SimpleDB....316
2.16.3 Implementación de un Anonymous Token Vending Machine....319
2.16.4 Desarrollo de una aplicación *iOS* comunicada con un Amazon SimpleDB mediante un Token Vending Machine....323
2.17 Cómo generar valor agregado a través de la aplicación....328
2.17.1 Implementación de un banner publicitario *iAd* en una aplicación....329
2.17.2 Mejores practicas al implementar publicidad en aplicaciones....335

Conclusiones....338

Capítulo 3: Desarrollo de una aplicación iOS de recordatorios por GPS mediante el tutorial....339
3.1 Análisis y diseño de la aplicación....340
3.1.1 Funcionalidad y ámbito de la aplicación....341
3.1.2 Levantamiento de requisitos....342
3.1.3 Diagramas de funcionalidad de la aplicación....347
3.1.4 Diseño de Base de Datos....351
3.1.5 Diseño de Interface....352
3.2 Desarrollo de la aplicación....355
3.3 Pruebas de la aplicación....356
3.4 Envío para revisión y aprobación para comercialización en AppStore....359
3.5 Aplicación de sugerencias del tutorial....360
3.5.1 Generación de una versión gratuita....361
3.5.2 Implementación de iAds....362
3.5.3 Motivar a usuarios a calificar la aplicación mediante AppiRater....363
3.6 Estadísticas de descargas y ventas de la aplicación....364

Conclusiones....366

Capítulo 4: Manual de usuario de la aplicación....367
4.1 Módulo de registro de lugares....368
4.2 Módulo de registro de tareas....371
4.3 Reportes....374
4.3.1 Reporte de lugares....374
4.3.2 Reporte de tareas asociadas a lugares....374
4.3.3 Reporte de tareas de acuerdo a proximidad de lugares....375
4.4 Configuraciones o preferencias del sistema....377

Conclusiones....380

CONCLUSIONES GENERALES **381**
RECOMENDACIONES **382**
GLOSARIO **383**
BIBLIOGRAFÍA **390**

ÍNDICE DE ILUSTRACIONES

Fig. 1.1: *IDE XCode* ____ *5*
Fig. 1.2: *Áreas de XCode* ____ *6*
Fig. 1.3: *Interface Builder* ____ *7*
Fig. 1.4: *Ejemplo de un Storyboard* ____ *8*
Fig. 1.5: *iOS Simulator (Orientación Portrait)* ____ *9*
Fig. 1.6: *iOS Simulator (Orientación Landscape)* ____ *10*
Fig. 1.7: *Base de datos de gestión de clientes* ____ *12*
Fig. 1.8: *Dominio resultante al añadir campos específicos para registros* ____ *13*
Fig. 1.9: *Herramienta de Apple iBooks Author* ____ *15*
Fig. 2.1: *Descarga e instalación de XCode* ____ *19*
Fig. 2.2: *Resultados de búsqueda "XCode" en AppStore* ____ *19*
Fig. 2.3: *Información sobre XCode en AppStore* ____ *19*
Fig. 2.4: *Instalación de XCode* ____ *19*
Fig. 2.5: *Acuerdo de licencia* ____ *20*
Fig. 2.6: *XCode en proceso de instalación* ____ *20*
Fig. 2.7: *Instalación finalizada exitosamente* ____ *20*
Fig. 2.8: *Carpeta "Developer"* ____ *20*
Fig. 2.9: *Programa de desarrollador de iOS* ____ *21*
Fig. 2.10: *Indicaciones de suscripción* ____ *22*
Fig. 2.11: *Pasos para la suscripción* ____ *23*
Fig. 2.12: *Registrar usando cuenta de Apple* ____ *23*
Fig. 2.13: *Selección de tipo de suscripción: individual / compañía* ____ *24*
Fig. 2.14: *Suscripción mediante la cuenta de Apple* ____ *25*
Fig. 2.15: *Llenar el formulario con la información personal correspondiente* ____ *26*
Fig. 2.16: *Marcar las categorías y tipos de aplicaciones que se planea desarrollar* ____ *27*
Fig. 2.17: *Acuerdo de registro como desarrollador de Apple* ____ *28*
Fig. 2.18: *Ingresar el código de verificación recibido* ____ *28*
Fig. 2.19: *Formulario de facturación* ____ *29*
Fig. 2.20: *Selección de programas en los que puede suscribirse el desarrollador* ____ *30*
Fig. 2.21: *Revisión de información ingresada* ____ *31*
Fig. 2.22: *Acuerdo de licencia de suscripción al programa de desarrollador iOS* ____ *32*
Fig. 2.23: *Tienda en línea Apple no disponible* ____ *33*
Fig. 2.24: *Formulario de datos para compra del programa de desarrollador iOS* ____ *34*
Fig. 2.25: *Email de confirmación de activación de cuenta* ____ *35*
Fig. 2.26: *Ventana inicial al abrir XCode* ____ *36*
Fig. 2.27: *Botón de nuevo proyecto* ____ *36*
Fig. 2.28: *Crear una aplicación en blanco o vacía* ____ *37*
Fig. 2.29: *Nombre y opciones para el proyecto* ____ *38*
Fig. 2.30: *Listado Device Family* ____ *40*
Fig. 2.31: *Ruta de alojamiento para el proyecto* ____ *41*
Fig. 2.32: *Ventana de un nuevo proyecto recién creado* ____ *41*
Fig. 2.33: *Ubicación del archivo Info.plist* ____ *44*
Fig. 2.34: *Interface de un Info.plist en XCode* ____ *45*
Fig. 2.35: *Contenidos de un archivo Info.plist* ____ *45*
Fig. 2.36: *Configuración de orientación Landscape en iPad y orientación Portrait en otros dispositivos iOS mediante el archivo Info.plist* ____ *46*
Fig. 2.37: *InfoPlist.strings dentro del directorio en.lproj* ____ *48*

Fig. 2.38: *InfoPlist.strings dentro del directorio French.lproj* 48
Fig. 2.39: *Contenido del archivo main.m* 49
Fig. 2.40: *Contenido del archivo Prefix.pch* 50
Fig. 2.41: *UIView* 54
Fig. 2.42: *Elementos de un Storyboard* 56
Fig. 2.43: *Forma de agregar un nuevo archivo* 57
Fig. 2.44: *Selección de archivo Storyboard* 58
Fig. 2.45: *Device Family en el que funcionará el Storyboard* 58
Fig. 2.46: *Nombrar el Storyboard* 59
Fig. 2.47: *Storyboard agregado con éxito* 59
Fig. 2.48: *Asignación de un Storyboard al proyecto* 60
Fig. 2.49: *Selección de un archivo Objective-C class* 61
Fig. 2.50: *Nombre de la clase y subclase a la que pertenece* 61
Fig. 2.51: *Selección de la ruta de almacenamiento del UIViewController* 62
Fig. 2.52: *UIViewController agregado con éxito* 63
Fig. 2.53: *Arrastrar y soltar una imagen sobre el Área de Navegación del proyecto* 64
Fig. 2.54: *Opciones de agregado de imágenes en un proyecto* 64
Fig. 2.55: *Archivo de imagen agregado con éxito* 65
Fig. 2.56: *Íconos de las aplicaciones de Apple Inc.* 66
Fig. 2.57: *Archivos de imagen para ícono agregados al proyecto* 67
Fig. 2.58: *Insertar una propiedad o fila en el archivo de propiedades del proyecto* 67
Fig. 2.59: *Agregar propiedad Icon files* 68
Fig. 2.60: *Agregado de iconos en el archivo de propiedades* 68
Fig. 2.61: *Ícono de la aplicación asignado con éxito* 69
Fig. 2.62: *Ícono de la aplicación asignado al proyecto exitosamente* 69
Fig. 2.63: *Archivos imagen agrupados dentro del proyecto* 70
Fig. 2.64: *Creación de un nuevo grupo* 70
Fig. 2.65: *Nuevo grupo a partir de selección de archivos* 71
Fig. 2.66: *Archivos agrupados* 71
Fig. 2.67: *Barra de selección de librerías* 73
Fig. 2.68: *Mostrar librería de objetos de GUI de una aplicación iOS* 73
Fig. 2.69: *Ícono View Controller* 74
Fig. 2.70: *View Controller vacío* 74
Fig. 2.71: *View Controller que presenta una GUI* 74
Fig. 2.72: *Ícono Table View Controller* 75
Fig. 2.73: *Table View Controller vacío* 75
Fig. 2.74: *Aplicación nativa de iOS "Contactos" usa Table View Controller* 75
Fig. 2.75: *Ícono Navigation Controller* 76
Fig. 2.76: *Navigation Controller vacío* 76
Fig. 2.77: *Aplicación nativa de dispositivos iOS "Ajustes" que incorpora un Navigation Controller* 77
Fig. 2.78: *Ícono Tab Bar Controller* 78
Fig. 2.79: *Tab Bar Controller de la aplicación nativa de Apple "Teléfono" (5 pestañas)* 78
Fig. 2.80: *Tab Bar Controller de 2 pestañas* 78
Fig. 2.81: *Tab Bar Controller de 4 pestañas* 78
Fig. 2.82: *Ícono Label* 79
Fig. 2.83: *View Controller con 3 Labels personalizados* 79
Fig. 2.84: *Ícono Round Rect Button* 80
Fig. 2.85: *View Controller con 5 botones* 80
Fig. 2.86: *Ícono Segmented Control* 81
Fig. 2.87: *View Controller con 3 Segmented Control* 81
Fig. 2.88: *Ícono Text Field* 82
Fig. 2.89: *View Controller con 2 Text Field* 82
Fig. 2.90: *Ícono Switch* 83
Fig. 2.91: *View Controller con 3 Switches* 83
Fig. 2.92: *Ícono Activity Indicator View* 84
Fig. 2.93: *View Controller con 2 Activity Indicator Views* 84
Fig. 2.94: *Ícono Progress View* 85
Fig. 2.95: *View Controller con Progress Views clasificados por estilo* 85
Fig. 2.96: *Ícono Image View* 86
Fig. 2.97: *View Controller con un Image View sin imagen asignada y otro con una imagen asignada* 86

Fig. 2.98: *Ícono Web View* 87
Fig. 2.99: *Web View dentro de un View Controller* 87
Fig. 2.100: *Web View cargando contenido web* 87
Fig. 2.101: *Ícono Map View* 88
Fig. 2.102: *Map View dentro de un View Controller* 88
Fig. 2.103: *Map View presentando información de mapa* 88
Fig. 2.104: *Ícono Picker View* 89
Fig. 2.105: *Picker View de un solo componente* 89
Fig. 2.106: *Picker View de 3 componentes: Date Picker* 89
Fig. 2.107: *Ícono Ad BannerView* 90
Fig. 2.108: *Ad BannerView dentro de un View Controller* 90
Fig. 2.109: *Ad BannerView presentado publicidad* 90
Fig. 2.110: *Interface gráfica de usuario para presentar los vehículos disponibles en la empresa* 92
Fig. 2.111: *Ficheros del proyecto EjemploGUI* 93
Fig. 2.112: *Arrastrar un componente View Controller sobre un Storyboard* 94
Fig. 2.113: *Barra de selección de inspector* 94
Fig. 2.114: *Asignación de clase controladora del View Controller* 95
Fig. 2.115: *Arrastrar un Image View sobre el View Controller* 95
Fig. 2.116: *Redimensionado y alineamiento de un componente* 96
Fig. 2.117: *Asignado de una imagen al componente Image View* 96
Fig. 2.118: *Arrastrar un Label sobre el View Controller* 97
Fig. 2.119: *Label "Vehículo:" en vista de EjemploGUI* 97
Fig. 2.120: *Información del vehículo en Labels* 97
Fig. 2.121: *Tipo de letra de un Label* 98
Fig. 2.122: *Color de texto de un Label* 98
Fig. 2.123: *Labels de titulo personalizados* 98
Fig. 2.124: *Propiedades de Labels de descripción del vehículo* 98
Fig. 2.125: *GUI con una imagen y Labels personalizados* 99
Fig. 2.126: *Arrastrar un Round Rect Button sobre el View Controller* 99
Fig. 2.127: *Round Rect Button personalizado en forma de un teléfono* 100
Fig. 2.128: *Botón teléfono y Label* 100
Fig. 2.129: *Agregar componentes Image View y botones* 101
Fig. 2.130: *Componentes con imágenes asignadas* 101
Fig. 2.131: *Arrastrar un Segmented Control sobre el View Controller* 101
Fig. 2.132: *Personalización de un Segmented Control* 102
Fig. 2.133: *Personalización del segundo segmento del Segmented Control* 102
Fig. 2.134: *Segmented Control y Labels* 103
Fig. 2.135: *Arrastrar un Text Field sobre el View Controller* 103
Fig. 2.136: *Ajustar propiedades del Round Rect Button* 104
Fig. 2.137: *Propiedad Placeholder de un Text Field* 104
Fig. 2.138: *Comentar una selección de texto (Método loadView de EjemploGUIViewController.m)* 105
Fig. 2.139: *Método "application: didFinishLaunchingWithOptions:"* 105
Fig. 2.140: *EjemploGUI se ejecutará en el simulador iOS* 106
Fig. 2.141: *iOS Simulator ejecutando la aplicación* 106
Fig. 2.142: *Plantillas prediseñadas para una aplicación iOS* 107
Fig. 2.143: *Archivos de una aplicación Master-Detail* 108
Fig. 2.144: *iPhone Storyboard de una aplicación Master-Detail* 109
Fig. 2.145: *Secuencia de una aplicación Master-Detail en iPhone* 109
Fig. 2.146: *iPad Storyboard de una aplicación Master-Detail* 110
Fig. 2.147: *Secuencia de una aplicación Master Detail en iPad* 111
Fig. 2.148: *Archivos de un juego OpenGL* 112
Fig. 2.149: *Juego basado en OpenGL en ejecución* 113
Fig. 2.150: *Archivos de una aplicación basada en páginas* 114
Fig. 2.151: *iPhone Storyboard de una aplicación basada en páginas* 115
Fig. 2.152: *iPad Storyboard de una aplicación basada en páginas* 115
Fig. 2.153: *Secuencia de una aplicación basada en páginas* 116
Fig. 2.154: *Archivos de una aplicación de una sola vista* 117
Fig. 2.155: *iPhone Storyboard de una aplicación de una sola ventana* 118
Fig. 2.156: *iPad Storyboard de una aplicación de una sola ventana* 118
Fig. 2.157: *Archivos de una aplicación de barra de pestañas* 119

Fig. 2.158: *iPhone Storyboard de una aplicación basada en pestañas* 120
Fig. 2.159: *iPad Storyboard de una aplicación basada en pestañas* 121
Fig. 2.160: *Secuencia de una aplicación basada en pestañas* 121
Fig. 2.161: *Archivos de una aplicación de utilidades* 122
Fig. 2.162: *iPhone Storyboard de una aplicación de utilidades* 123
Fig. 2.163: *Secuencia de una aplicación de utilidades en iPhone* 124
Fig. 2.164: *iPad Storyboard de una aplicación de utilidades* 124
Fig. 2.165: *Secuencia de una aplicación de utilidades en iPad* 125
Fig. 2.166: *Archivos de una aplicación vacía* 126
Fig. 2.167: *Aplicación vacía* 126
Fig. 2.168: *Proyecto RotationExample* 129
Fig. 2.169: *Orientaciones de un dispositivo iOS* 129
Fig. 2.170: *Seleccionar las orientaciones deseadas para la aplicación* 130
Fig. 2.171: *Interface gráfica de usuario de RotationExample* 130
Fig. 2.172: *Objetivo de la aplicación RotationExample* 131
Fig. 2.173: *Cambiar la orientación del iOS Simulator* 131
Fig. 2.174: *Componentes desaparecidos al girar el dispositivo iOS* 132
Fig. 2.175: *Métodos del fichero RotationExampleViewController.m* 133
Fig. 2.176: *Contenido del método shouldAutorortateToInterfaceOrientation:* 133
Fig. 2.177: *Propiedades de tamaño del objeto* 134
Fig. 2.178: *Propiedad de auto ajuste de tamaño de componentes iOS* 135
Fig. 2.179: *Tabla de ejemplos de la funcionalidad autosizing* 136
Fig. 2.180: *Tabla de patrones de autosizing* 139
Fig. 2.181: *Tabla de patrones de autosizing* 139
Fig. 2.182: *Componentes solapados al girar el dispositivo iOS* 139
Fig. 2.183: *Presentación de los botones en diferentes orientaciones del dispositivo* 140
Fig. 2.184: *Selección de un botón y visualización de sus atributos de tamaño* 140
Fig. 2.185: *Tabla de tamaño y posición de cada componente de la interface en orientación Portrait* 141
Fig. 2.186: *Ajustar la orientación de una vista en un Storyboard* 141
Fig. 2.187: *Selección de un botón y visualización de sus atributos de tamaño.* 142
Fig. 2.188: *Tabla de tamaño y posición de cada componente de la interface en orientación Landscape* 142
Fig. 2.189: *Variables que se deben asignar a cada botón* 143
Fig. 2.190: *Selección del archivo en donde se implementará el método* 143
Fig. 2.191: *Métodos que inician con la palabra "-will"* 144
Fig. 2.192: *Método seleccionado* 144
Fig. 2.193: *Método de reposicionamiento implementado* 145
Fig. 2.194: *Agregar un Label sobre el View Controller y ajustarlo* 147
Fig. 2.195: *Mostrar el Assistant editor* 148
Fig. 2.196: *Conexión de un objeto con su respectiva clase controladora* 149
Fig. 2.197: *Ventana de propiedades de la conexión (variable/outlet)* 149
Fig. 2.198: *Línea adicionada al archivo cabecera* 150
Fig. 2.199: *Línea adicionada al archivo de implementación* 151
Fig. 2.200: *Manejar las propiedades de una variable mediante código fuente* 152
Fig. 2.201: *Funcionalidad de AdvancedVariablesExample* 153
Fig. 2.202: *Agregar un Round Rect Button y ajustarlo como se indica.* 154
Fig. 2.203: *Conexión del Round Rect Button con su clase controladora.* 155
Fig. 2.204: *Ventana de propiedades de la conexión (método/action)* 156
Fig. 2.205: *Métodos del objeto Round Rect Button* 156
Fig. 2.206: *Línea adicionada al archivo cabecera* 157
Fig. 2.207: *Línea adicionada al archivo de implementación* 157
Fig. 2.208: *Código que se debe escribir para generar un color de fondo aleatorio* 158
Fig. 2.209: *Aplicación recién construida en ejecución* 159
Fig. 2.210: *Funcionalidad de AdvancedActionsExample* 159
Fig. 2.211: *Eventos para configurar un Table View* 164
Fig. 2.212: *Arrastrar un Navigation Controller sobre el Storyboard* 166
Fig. 2.213: *Navigation Controller sin Table View Controller (no necesario)* 167
Fig. 2.214: *Agregar un View Controller y un Table View Controller* 167
Fig. 2.215: *Conexión del Navigation Controller con el View Controller* 168
Fig. 2.216: *Tipos de conexión* 168

Fig. 2.217: *Navigation Controller conectado al View Controller* 169
Fig. 2.218: *Vista 1 personalizada y con Navigation Controller* 169
Fig. 2.219: *Conexión del botón con el Table View Controller* 170
Fig. 2.220: *Storyboard de NavigationControllerExample* 170
Fig. 2.221: *Flujo de una aplicación que implementa un Navigation Controller* 171
Fig. 2.222: *Instanciar una vista mediante código fuente.* 172
Fig. 2.223: *Asignación de un identificador para una vista.* 172
Fig. 2.224: *Arrastrar un Tab Bar Controller sobre el Storyboard* 173
Fig. 2.225: *Asociar mas View Controllers al Table View Controller* 174
Fig. 2.226: *Tipos de conexión con un Tab Bar Controller* 174
Fig. 2.227: *Tab Bar Controller asociado a cuatro View Controllers* 175
Fig. 2.228: *Personalización del ícono y título de pestaña de un View Controller* 176
Fig. 2.229: *Personalización del resto de pestañas de los View Controllers del Storyboard* 176
Fig. 2.230: *Tab Bar Controller con Tab Bar personalizado* 177
Fig. 2.231: *TabBarControllerExample en ejecución* 177
Fig. 2.232: *Flecha que representa una conexión entre dos vistas (Segue)* 179
Fig. 2.233: *Storyboard de la aplicación demostrativa TransitionsExample* 180
Fig. 2.234: *Propiedades de un Segue* 180
Fig. 2.235: *Tipo de transición Cover Vertical* 181
Fig. 2.236: *Tipo de transición Flip Horizontal* 182
Fig. 2.237: *Tipo de transición Cross Dissolve* 182
Fig. 2.238: *Tipo de transición Partial Curl* 183
Fig. 2.239: *Arquitectura de CoreData* 185
Fig. 2.240: *Diagrama ejemplificado de Core Data* 186
Fig. 2.241: *Marcar opción Use Core Data al crear el proyecto* 188
Fig. 2.242: *Interface gráfica de CoreDataExample* 188
Fig. 2.243: *Tabla de variables y acciones a asignar a componentes* 189
Fig. 2.244: *Fichero AppDelegate.h al implementar CoreData* 190
Fig. 2.245: *managedObjectContext en fichero AppDelegate.m* 190
Fig. 2.246: *managedObjectModel en fichero AppDelegate.m* 191
Fig. 2.247: *persistentStoreCoordinator en fichero AppDelegate.m* 191
Fig. 2.248: *applicationDocumentsDirectory en fichero AppDelegate.m* 191
Fig. 2.249: *saveContext en fichero AppDelegate.m* 192
Fig. 2.250: *Agregar una clase con sus atributos al modelo de datos* 193
Fig. 2.251: *Visualización del modelo de datos establecido.* 194
Fig. 2.252: *Agregar un fichero NSManagedObjectSubclass* 194
Fig. 2.253: *Archivos del proyecto CoreDataExample* 195
Fig. 2.254: *Importar los ficheros cabecera de las clases necesarias* 195
Fig. 2.255: *Método grabar marca llamado al presionar el botón "Grabar"* 196
Fig. 2.256: *Método borrar marca llamado al presionar el botón "Borrar"* 197
Fig. 2.257: *Método modificar marca llamado al presionar el botón "Modificar"* 197
Fig. 2.258: *Método mostrar marcas llamado al presionar el botón "Mostrar/Actualizar marcas ingresadas"* 198
Fig. 2.259: *Estructura de una consulta* 199
Fig. 2.260: *Funcionamiento de la aplicación CoreDateExample* 201
Fig. 2.261: *Visualización de la base de datos creada desde la iOS App en MesaSQLite* 202
Fig. 2.262: *Error en consola al ejecutar aplicación con modelo alterado* 202
Fig. 2.263: *Eliminar una aplicación* 203
Fig. 2.264: *Forma de agregar un nuevo framework al proyecto* 205
Fig. 2.265: *Listado de frameworks y librerías disponibles* 205
Fig. 2.266: *Framework CoreLocation agregado con éxito* 206
Fig. 2.267: *Organización de frameworks del proyecto* 206
Fig. 2.268: *Arrastrar al proyecto un framework descargado de internet* 207
Fig. 2.269: *Framework agregado exitosamente* 207
Fig. 2.270: *CoreLocation solicitud para usar la localización del usuario* 208
Fig. 2.271: *Framework CoreData* 210
Fig. 2.272: *MapKit con puntos de interés y localización del usuario* 210
Fig. 2.273: *Ficheros de la aplicación TwitterTest* 212
Fig. 2.274: *View Controller de TwitterTest* 213
Fig. 2.275: *Fichero cabecera del View Controller de TwitterTest* 213

Fig. 2.276: *Fichero de implementación del View Controller de TwitterTest* 214
Fig. 2.277: *Aplicación TwitterTest en ejecución* 215
Fig. 2.278: *Descargar e instalar el paquete Facebook iOS SDK* 216
Fig. 2.279: *Contenidos de la carpeta FacebookSDK* 217
Fig. 2.280: *Frameworks y archivos del proyecto FacebookTest* 218
Fig. 2.281: *Crear enlaces entre la aplicación y SQLite / Objective C* 218
Fig. 2.282: *Agregar atributo FacebookAppID al fichero Info.plist* 219
Fig. 2.283: *View Controller de FacebookTest* 220
Fig. 2.284: *Fichero cabecera del View Controller de FacebookTest* 220
Fig. 2.285: *Fichero de implementación del View Controller de FacebookTest* 221
Fig. 2.286: *Aplicación FacebookTest en ejecución* 222
Fig. 2.287: *iOS Provisioning Portal exclusivo para miembros desarrolladores de Apple* 223
Fig. 2.288: *Abrir el organizador de XCode (Organizer)* 226
Fig. 2.289: *Usar iPhone 5 para desarrollo* 226
Fig. 2.290: *Procesando archivos necesarios para iPhone5* 227
Fig. 2.291: *Iniciar sesión con los datos de desarrollador* 227
Fig. 2.292: *No solicitar un certificado automáticamente* 227
Fig. 2.293: *Dispositivo iOS conectado y configurado pero sin perfil de aprovisionamiento* 228
Fig. 2.294: *Ingreso al centro de miembros desarrolladores de Apple* 228
Fig. 2.295: *Acceso al iOS Provisioning Portal* 229
Fig. 2.296: *Dispositivo agregado exitosamente al iOS Provisioning Profile* 229
Fig. 2.297: *Solicitar un certificado de desarrollador* 230
Fig. 2.298: *Solicitar un certificado de una autoridad de certificación mediante la aplicación Acceso a Llaveros incluida por defecto en el Mac* 230
Fig. 2.299: *Datos del desarrollador para el certificado* 231
Fig. 2.300: *Directorio de grabado* 231
Fig. 2.301: *Archivo de solicitud de certificado* 231
Fig. 2.302: *Conclusión del procedimiento* 231
Fig. 2.303: *Enviar la solicitud de certificado de desarrollador* 232
Fig. 2.304: *iOS Provisioning Portal - Certificates* 232
Fig. 2.305: *Crear un nuevo App ID para la aplicación RotationExample* 233
Fig. 2.306: *Escribir los datos necesarios para crear el App ID* 233
Fig. 2.307: *Listado de App IDs con el estado de cada uno de sus atributos* 234
Fig. 2.308: *Crear un nuevo Development Provisioning Profile* 234
Fig. 2.309: *Datos de un nuevo Development Provisioning Profile* 235
Fig. 2.310: *Descargar el Development Provisioning Profile para RotationExample* 235
Fig. 2.311: *Descarga de certificados de desarrollador y WWDR* 236
Fig. 2.312: *Ejecución de los ficheros descargados* 236
Fig. 2.313: *Configurar el firmado de código de la aplicación* 237
Fig. 2.314: *Ejecución de RotationExample en un dispositivo físico* 237
Fig. 2.315: *Error ubicado en la compilación del proyecto AdvancedVariablesExample* 238
Fig. 2.316: *Insertar un breakpoint en proyecto AdvancedVariablesExample* 239
Fig. 2.317: *Depuración de una aplicación* 239
Fig. 2.318: *Barra de Depuración* 239
Fig. 2.319: *View Controller de PhotosApp* 243
Fig. 2.320: *Archivo cabecera del View Controller de PhotosApp* 244
Fig. 2.321: *Archivo de implementación del View Controller de PhotosApp* 245
Fig. 2.322: *Aplicación PhotosApp en ejecución en el simulador iOS* 245
Fig. 2.323: *View Controller de MicrophoneApp* 247
Fig. 2.324: *Archivo cabecera del View Controller de MicrophoneApp* 247
Fig. 2.325: *Archivo de implementación del View Controller de MicrophoneApp* 249
Fig. 2.326: *Aplicación SD para iPhone e iPod y HD para iPad* 250
Fig. 2.327: *Aplicación exclusiva para iPhone e iPod* 250
Fig. 2.328: *Aplicación Universal para cualquier dispositivo iOS* 251
Fig. 2.329: *AdvancedActionsExample iPad Storyboard* 252
Fig. 2.330: *Asignación de una variable existente a un componente en un Storyboard* 253
Fig. 2.331: *Asignación de métodos existentes a componentes en un Storyboard* 254
Fig. 2.332: *Configuración de los ajustes del proyecto de la aplicación universal* 254
Fig. 2.333: *Método aumentarNumero. Presenta una alerta si se ejecuta en un iPad* 255
Fig. 2.334: *Aplicación universal AdvanceActionsExample en ejecución en diferentes dispositivos iOS* 255

Fig. 2.335: *Implicaciones negativas en los desarrolladores* 256
Fig. 2.336: *Forma común de hacking de aplicaciones* 256
Fig. 2.337: *Diferentes tipos de hacks en aplicaciones móviles* 257
Fig. 2.338: *Solicitar un nuevo contrato de aplicaciones de pago de iOS* 269
Fig. 2.339: *Validación de información legal del desarrollador o empresa* 269
Fig. 2.340: *Editar información bancaria para que el contrato entre en vigencia* 270
Fig. 2.341: *Agregar una cuenta bancaria* 270
Fig. 2.342: *Alternativa para buscar el banco en el cual se dispone una cuenta* 270
Fig. 2.343: *Ingresar los datos de la cuenta bancaria del banco seleccionado en el paso anterior* 271
Fig. 2.344: *Seleccionar la cuenta bancaria creada que aparece ahora en el listado* 271
Fig. 2.345: *Creación de un App Id para la aplicación TwitterTest* 273
Fig. 2.346: *Solicitar un certificado de desarrollador* 274
Fig. 2.347: *Seleccionar el CSR y enviarlo para solicitar el certificado* 274
Fig. 2.348: *Crear un nuevo Distribution Provisioning Profile* 275
Fig. 2.349: *Datos del Distribution Provisioning Profile* 275
Fig. 2.350: *Descargar y ejecutar el Distribution Provisioning Profile y el Certificado* 276
Fig. 2.351: *Cambiar el tipo de dispositivo del iOS Simulator* 278
Fig. 2.352: *Escalar para reducir el tamaño del dispositivo iOS simulado* 278
Fig. 2.353: *Copiar pantalla de la aplicación en un dispositivo concreto.* 279
Fig. 2.354: *Iniciar sesión en el portal iTunes Connect* 280
Fig. 2.355: *Ingresar a la sección Manage Your Apps* 281
Fig. 2.356: *Presionar para agregar una nueva aplicación* 281
Fig. 2.357: *Ingresar la información de la aplicación y su Bundle ID* 282
Fig. 2.358: *Ingresar datos financieros y fecha de disponibilidad de la aplicación* 282
Fig. 2.359: *Ingreso de versión, copyright, categoría y rating de la aplicación* 284
Fig. 2.360: *Completar información de revisión de la aplicación* 285
Fig. 2.361: *Ingresar el ícono y las capturas de pantalla de la aplicación* 286
Fig. 2.362: *Aplicación en estado Prepare for Upload* 287
Fig. 2.363: *Verificar datos de la aplicación y continuar* 287
Fig. 2.364: *Uso de criptografía en la aplicación* 287
Fig. 2.365: *Indicar que se está listo para enviar la aplicación para revisión* 288
Fig. 2.366: *Aplicación en estado Waiting For Upload* 288
Fig. 2.367: *Seleccionar el certificado de distribuidor para el firmado de la aplicación* 289
Fig. 2.368: *Edición del esquema del proyecto para el archivado de la aplicación* 289
Fig. 2.369: *Configuración del esquema de archivado de la aplicación* 290
Fig. 2.370: *Forma de archivar la aplicación* 290
Fig. 2.371: *Autorizar el firmado de la aplicación* 290
Fig. 2.372: *Aplicación archivada y presentada en el "Organizador" de XCode* 291
Fig. 2.373: *Procedimiento para enviar la aplicación para revisión desde el Organizador de XCode* 292
Fig. 2.374: *Aplicación en estado "Waiting For Review"* 292
Fig. 2.375: *Notificaciones vía correo electrónico del estado de la aplicación* 293
Fig. 2.376: *Aplicación aprobada para su comercialización* 293
Fig. 2.377: *AppiRater implementado en la aplicación Prayer Book* 295
Fig. 2.378: *Interface Gráfica de AppiRaterTest y ficheros necesarios* 297
Fig. 2.379: *Fichero AppDelegate del proyecto AppiRaterTest* 298
Fig. 2.380: *Definición de constantes de la clase AppiRater* 299
Fig. 2.381: *Establecer el significant event dentro del método de presión del botón* 300
Fig. 2.382: *Especificar que la clase no utiliza Automatic Reference Counting* 300
Fig. 2.383: *Aplicación AppiRaterTest en funcionamiento* 301
Fig. 2.384: *Identificador UDID de un dispositivo iOS* 303
Fig. 2.385: *Información que se podría relacionar con un identificador UDID* 304
Fig. 2.386: *Proyecto UniqueIdentifierTest* 305
Fig. 2.387: *SDKs de AWS disponibles para las distintas plataformas* 308
Fig. 2.388: *Credenciales de Seguridad del usuario registrado* 309
Fig. 2.389: *Scratchpad que permite explorar las API de Amazon SimpleDB* 310
Fig. 2.390: *Interface gráfica y ficheros del proyecto AWSSimpleDBTest* 310
Fig. 2.391: *Archivo cabecera del View Controller de AWSSimpleDBTest* 311
Fig. 2.392: *Constantes y viewDidLoad del archivo de implementación de AWSSimpleDBTest* 312
Fig. 2.393: *Método que permite crear un dominio en la SimpleDB* 312
Fig. 2.394: *Método que permite almacenar registros en un dominio de la SimpleDB* 313

Fig. 2.395: *Método que permite leer registros almacenados en un dominio de la SimpleDB* 313
Fig. 2.396: *Sentencias que permiten la depuración de la conexión con la SimpleDB (AppDelegate)* 314
Fig. 2.397: *AWSSimpleDBTest en ejecución* 314
Fig. 2.398: *Visualización de la información en el SimpleDB mediante el Scratchpad* 315
Fig. 2.399: *Flujo de credenciales temporales desde AWS Security Token Service hasta iOS App* 317
Fig. 2.400: *Interacción entre un TVM anónimo y un dispositivo iOS* 318
Fig. 2.401: *Completar los detalles de la aplicación TVM* 319
Fig. 2.402: *Completar los detalles del entorno de la aplicación TVM* 320
Fig. 2.403: *Completar detalles de configuración de la aplicación TVM* 321
Fig. 2.404: *Editar la configuración del TVM creado* 322
Fig. 2.405: *Ingresar las credenciales de seguridad para el TVM* 322
Fig. 2.406: *Interface gráfica y frameworks de TVMTest* 323
Fig. 2.407: *Archivo cabecera del View Controller de TVMTest* 324
Fig. 2.408: *Clases necesarias para comunicación con el TVM, URL del TVM* 325
Fig. 2.409: *Indicar las clases que no soportan ARC* 325
Fig. 2.410: *Método que permite grabar un registro en la SimpleDB utilizando el TVM* 326
Fig. 2.411: *Método que permite leer un registro de la SimpleDB utilizando el TVM* 326
Fig. 2.412: *Aplicación TVMTest en ejecución* 327
Fig. 2.413: *Interface gráfica y archivos del proyecto iAdTest* 329
Fig. 2.414: *Fichero cabecera del View Controller de iAdTest* 330
Fig. 2.415: *Fichero de implementación del View Controller de iAdTest* 331
Fig. 2.416: *Implementar eventos de delegado del AdBannerView* 332
Fig. 2.417: *Implementar métodos para rotar interfaces de usuario en iOS 5 e iOS 6* 332
Fig. 2.418: *Instrucción para evitar las advertencias de uso de métodos deprecados* 333
Fig. 2.419: *Aplicación iAdTest en ejecución* 333
Fig. 2.420: *Métodos para manejar eventos de presentación y ocultamiento de vista publicitaria* 334
Fig. 2.421: *Contrato de red iAd acordado junto a los otros contratos vigentes* 335
Fig. 2.422: *Configurar red iAd en una aplicación en iTunes Connect* 336
Fig. 3.1: *Diagrama de casos de uso de la aplicación iOS de recordatorios por GPS* 342
Fig. 3.2: *Diagrama de Secuencias - Mantenimiento y reporte de lugares* 347
Fig. 3.3: *Diagrama de Secuencias - Mantenimiento y reporte de tareas* 347
Fig. 3.4: *Diagrama de Secuencias - Realizar una tarea* 348
Fig. 3.5: *Diagrama de Secuencias - Contactar al desarrollador* 348
Fig. 3.6: *Diagrama de Secuencias – Configurar la aplicación iOS* 349
Fig. 3.7: *Diagrama de Secuencias – Compartir en Redes Sociales* 349
Fig. 3.8: *Diagrama de actividades de la aplicación de recordatorios por GPS GPSReminds* 350
Fig. 3.9: *Diagrama de base de datos de GPSReminds* 351
Fig. 3.10: *Tab Bar Controller de la aplicación GPSReminds* 352
Fig. 3.11: *Interface gráfica de usuario para los lugares* 353
Fig. 3.12: *Interface gráfica de usuario para los recordatorios* 353
Fig. 3.13: *Interface gráfica de usuario para el reporte de tareas de acuerdo a la ubicación actual del usuario* 354
Fig. 3.14: *Interface gráfica de usuario para los ajustes de la aplicación* 354
Fig. 3.15: *Caso de prueba: Validación de entradas de la vista Agregar Lugar* 356
Fig. 3.16: *Caso de prueba: Validación de entradas de la vista Agregar Tarea* 357
Fig. 3.17: *Caso de prueba: Validación de entradas de la vista Distancia* 357
Fig. 3.18: *Aplicación iOS GPSReminds disponible en la tienda AppStore* 359
Fig. 3.19: *Aplicación iOS GPSReminds Free disponible en la tienda AppStore* 362
Fig. 3.20: *GPSReminds con un banner publicitario* 362
Fig. 3.21: *Resumen de ventas de la aplicación* 364
Fig. 3.22: *Resumen de ganancias perteneciente al mes de marzo del 2012* 364
Fig. 3.23: *Ganancias después de los impuestos* 365
Fig. 4.1: *Vista de Lugares* 368
Fig. 4.2: *Tipos de vistas de mapa* 368
Fig. 4.3: *Advertencia de modificación tarea asignada a un lugar* 369
Fig. 4.4: *Listado de lugares* 369
Fig. 4.5: *Vista para modificar lugares* 369
Fig. 4.6: *Eliminar un lugar* 370
Fig. 4.7: *Eliminar un lugar (Swipe To Delete)* 370
Fig. 4.8: *Ventana informativa* 371

Fig. 4.9: *Picker View de lugares* _____ *371*
Fig. 4.10: *Vista de tareas* _____ *371*
Fig. 4.11: *Vista para agregar tareas* _____ *371*
Fig. 4.12: *Tocar una tarea para marcarla como realizada* _____ *372*
Fig. 4.13: *Listado de tareas* _____ *372*
Fig. 4.14: *Eliminar una tarea* _____ *373*
Fig. 4.15: *Eliminar una tarea (Swipe To Delete)* _____ *373*
Fig. 4.16: *Reporte de tareas pendientes en un lugar cercano* _____ *375*
Fig. 4.17: *Reporte de tareas en lugares cercanos (pestaña "Aquí")* _____ *376*
Fig. 4.18: *Se ha encontrado más de un lugar cercano en el reporte* _____ *376*
Fig. 4.19: *Configuraciones de la aplicación GPSReminds* _____ *377*
Fig. 4.20: *Ventana "Acerca De" la aplicación* _____ *377*
Fig. 4.21: *Selección del idioma de la aplicación* _____ *378*
Fig. 4.22: *Configuración del parámtro distancia de la aplicación* _____ *378*
Fig. 4.23: *Publicar en Facebook indicando que se está utilizando la aplicación* _____ *379*
Fig. 4.24: *Enviar un Tweet indicando que se esta utilizando la aplicación* _____ *379*

RESUMEN

En el presente documento, se pretende explicar la manera de desarrollar aplicaciones móviles para dispositivos *iOS: iPhone, iPod* e *iPad;* abarcando temas como: instalación de los componentes necesarios, creación de proyectos, desarrollo de interfaces gráficas de usuario, conexión de componentes de interface gráfica con su respectivo código fuente, ejecución de aplicaciones en dispositivos físicos, entre otros. De esta forma, los usuarios con conocimientos básicos de programación, podrán utilizar este documento para desarrollar aplicaciones *iOS* y distribuirlas en la tienda de aplicaciones de *Apple, AppStore*. Para desarrollar un documento muy completo, se demostrará que se puede crear una aplicación *iOS* mediante éste; que adicionalmente contendrá sugerencias para desarrolladores e información sobre las herramientas con las que se desarrollarán las aplicaciones.

ABSTRACT

The purpose of the present document is to explain how to develop mobile application for *iOS* devices: *iPhone, iPod,* and *iPad.* It deals with topics regarding installation of the components, development of projects, development of graph based user interface, connection to the components of the graph based user interface with its corresponding source of codes, and implementation of the applications in actual devices among others. Consequently, the users who have a basic knowledge in programming will be able to use this document to develop *iOS* applications and distribute them in Apple Stores. To complete this document we will prove that with the aid of this study it is possible to create an *iOS* application. In addition, the document will contain suggestions for developers and information regarding the tools needed to create the application.

UNIVERSIDAD DEL
AZUAY
DPTO. IDIOMAS

Translated by,
Diana Lee Rodas

INTRODUCCIÓN:

En la actualidad existen algunos ejemplos en línea de desarrollo de aplicaciones para dispositivos *iOS*, la mayoría de los mismos, son para usuarios con experiencia con el *IDE XCode*, están en inglés y son casi obsoletos, debido a que fueron desarrollados para versiones anteriores del mismo. Este *IDE*, ha evolucionado con el tiempo, integrando todos sus componentes en una sola ventana en su versión más actual[1]. Se desconoce la existencia de un tutorial para el desarrollo de una aplicación *iOS* que abarque desde la instalación del *IDE*, hasta el proceso de aprobación y distribución de una aplicación en la tienda *AppStore*, es por esto que se lo desarrollará.

Este documento se diseñará con el propósito de ser una referencia útil para los lectores interesados, pudiendo ser estos: desarrolladores de software, personal docente, estudiantes, entre otros interesados en incursionar en el mundo del desarrollo de aplicaciones para dispositivos *iOS.* Contendrá los pasos para el desarrollo de aplicaciones *iOS*, desde la instalación del *IDE*, hasta la publicación de una aplicación en la tienda de *Apple: AppStore*.

Adicionalmente, se presentarán ejemplos que permitan al lector seguir paso a paso la creación de una aplicación determinada, se crearán varias aplicaciones sencillas, que integren los conocimientos adquiridos en los apartados del tutorial.

Se presentará también un conjunto de buenas prácticas como la manera de generar valor agregado adicional mediante el uso de *iAds*, la forma de popularizar la aplicación en la

[1] Apple. (2012). *About XCode*. Recuperado el 28 de Julio de 2012, de About XCode: https://developer.apple.com/library/mac/#documentation/ToolsLanguages/Conceptual/Xcode_User_Guide/000-About_Xcode/about.html

tienda *AppStore* mediante calificaciones positivas de los usuarios, y, sugerencias para el desarrollo de versiones gratuitas que permitan popularizar la aplicación.

Se demostrará que se puede crear una aplicación utilizando el tutorial, desarrollando una aplicación para *iOS* que será publicada para su comercialización en la tienda de *Apple*, *AppStore*.

Capítulo 1: Herramientas a Utilizar

El desarrollo y distribución o comercialización de aplicaciones para dispositivos móviles que ejecutan *iOS* (sistema operativo de dispositivos móviles de la empresa *Apple Inc.*) es muy atractivo y entretenido cuando se dispone del conocimiento y las herramientas necesarias para este propósito.

En este capítulo se presentarán las principales herramientas necesarias para diseñar, codificar, depurar, probar y distribuir aplicaciones para *iPhone*, *iPod* e *iPad*, que de ahora en adelante en el tutorial serán llamados "dispositivos *iOS*".

Para el desarrollo de este capítulo se referenciarán los libros *"XCode 4: Developer Reference"*, *"Beginning iOS 5 Development"* y *"Mastering XCode 4: Develop & Design",* así como los portales web de *Amazon* y *Apple* detallados en el anexo bibliográfico.

Cabe recalcar que se necesita esencialmente de un computador *Mac*, en el cual se instalará la mayoría de herramientas que se presentarán en este capítulo. No es necesaria una suscripción como desarrollador, pero presenta beneficios que se conocerán en posteriores apartados de este tutorial. Cada herramienta se compone de un conjunto de características y funcionalidades únicas, se detallarán las mismas a manera de resumen a continuación.

1.1 IDE XCode

XCode es el poderoso *IDE* o entorno de desarrollo integrado de la empresa *Apple Inc.* para el desarrollo de aplicaciones tanto para computadores *Mac* como para dispositivos *iOS*. *XCode* incluye la herramienta de análisis de aplicaciones *Instruments*, el simulador *iOS* y los *SDKs* tanto de *iOS* como de *Mac OS X*. Se puede ejecutar *XCode* en los sistemas operativos *OS X Mountain Lion* y *OS X Lion*.

La interface de *XCode* integra: edición de código, pruebas, depuraciones y la siguiente herramienta que se presentará para diseño de interfaces: *Interface Builder,* en una sola ventana. El compilador *LLVM* (*Low Level Virtual Machine*) que integra, subraya los errores de código, tanto de sintaxis como de lógica, a medida que se escribe; e incluso, es capaz de arreglar algunos problemas automáticamente[2]. Este compilador soporta *C, Objective-C, C++*, entre otros lenguajes. Es una gran ventaja debido a que integra características de coloreado de sintaxis y auto completado de código.

XCode soporta edición de versiones, implementa una línea de tiempo para comparar entre dos versiones de un código fuente, se puede "viajar en el tiempo" a través del proyecto, comparando cualquier dos versiones.

Otra de las ventajas que presenta *XCode* es el *Assistant* (o asistente) que permite visualizar documentos relacionados en una sola ventana. Por lo tanto: Se puede visualizar la cabecera correspondiente al archivo de implementación que se encuentre editando; así como se puede visualizar la clase controladora de la interface de usuario que se encuentre diseñando, facilitando el proceso de construcción de interfaces[3].

[2] Apple. (2012). *XCode*. Recuperado el 25 de Julio de 2012, de XCode: https://developer.apple.com/xcode/index.php

[3] Wenkt, R. *XCode 4: Developer Reference.* Indianapolis, Indiana: Wiley Publishing Inc. p24.

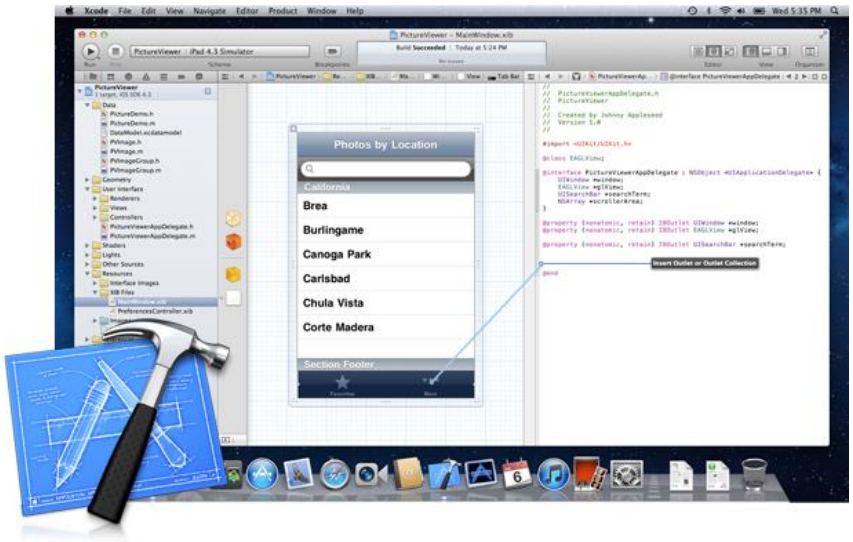


Fig. 1.1: *IDE XCode*

XCode soporta el desarrollo de varias personas simultáneamente usando sistemas *SCM (Source Control Management)* de manera de poder visualizar cambios realizados en un proyecto por cada uno de los desarrolladores involucrados en el mismo.

Presenta una simple ventana, llamada ventana de área de trabajo, comúnmente conocida como *Workspace Window*, en donde se presenta la mayoría de datos e información necesaria para la construcción de una aplicación.

En resumen, el *IDE XCode* presenta muchas características que facilitan el trabajo del desarrollador. Entre ellas se destacan las siguientes:

Interface de una sola ventana: Permite realizar la mayoría de actividades del desarrollo de aplicaciones para dispositivos *iOS* en una sola ventana.

Diseño de interfaces gráficas de usuario: Mediante *Interface Builder*, en donde se puede especificar la mayoría de los detalles de la interface de usuario de la aplicación *iOS*, como la distribución de objetos en la *GUI* y su conexión con la lógica y datos que maneja la aplicación. *Interface Builder* trabaja de cerca con otros editores como el de código fuente, para que la comunicación entre diseño e implementación se realice tan pronto como sea posible.

Edición Asistida: Cuando se necesite trabajar sobre diferentes aspectos de un mismo componente, como por ejemplo la interface de usuario y la implementación de la funcionalidad de la misma, se pueden utilizar múltiples editores que presentan lo que se necesita justo cuando se lo necesita. Por ejemplo, cuando se trabaje en un archivo de implementación ("*archivo.m*") en el editor principal, *XCode* puede abrir el archivo cabecera correspondiente ("*archivo.h*") en un panel de edición secundario.

Identificación y corrección automática de errores: *XCode* verifica el código fuente a medida que el desarrollador lo escribe. Cuando encuentra un error, lo resalta, presenta detalles sobre el mismo y de ser posible lo corrige automáticamente.

Control de Fuente (Source Control): Permite almacenar de manera segura todos los archivos de un proyecto en *Git* (software de control de versiones) y repositorios de subversiones de código fuente.

Distribución de aplicaciones: Permite publicar las aplicaciones desarrolladas en la tienda virtual de la empresa *Apple Inc.* para su futura comercialización[4].

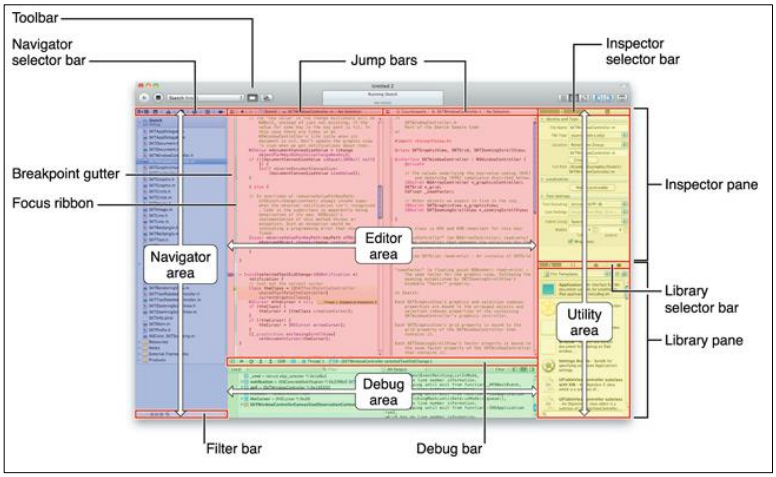


Fig. 1.2: Áreas de *XCode*[5]

[4] Apple. (2012). *About XCode*. Recuperado el 28 de Julio de 2012, de About XCode: https://developer.apple.com/library/mac/#documentation/ToolsLanguages/Conceptual/Xcode_User_Guide/000-About_Xcode/about.html

[5] Apple. (2012). *About XCode*. Recuperado el 28 de Julio de 2012, de About XCode.

1.2 Interface Builder

Es un editor de *XCode* que provee una interface gráfica para la creación de archivos de interfaces de usuario. Se encuentra completamente integrado en el *IDE*, de forma que se puede escribir y editar código fuente y unirlo directamente a la interface de usuario sin cambiar de ventanas, como se solía hacer en versiones previas de *XCode*.

Consta de un área de utilidades en la parte derecha, en la cual se pueden encontrar los objetos de interface, así como la librería de controles y los inspectores de interface. Presenta la característica "*Drag&Drop"*, por lo que se pueden arrastrar los controles de la librería al lienzo y de esta forma diseñar la interface de usuario de la aplicación para *iOS* o *Mac*.

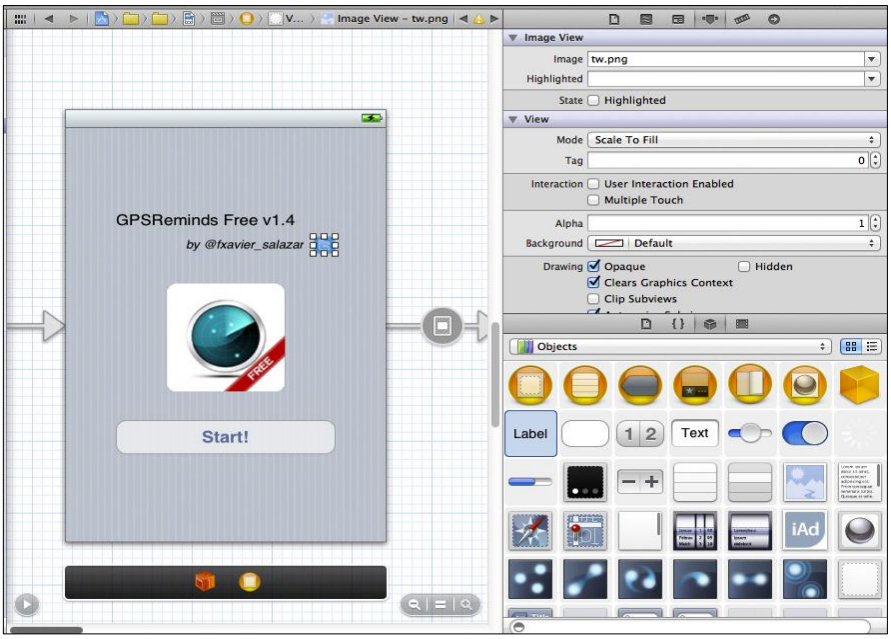


Fig. 1.3: *Interface Builder*

La característica mas atractiva de *Interface Builder* es que permite arrastrar conexiones directamente desde el diseño de interfaces de usuario hacia la ventana de edición de código fuente. Esto facilita el enlace tanto de acciones (eventos) y de variables (*outlets*) con simplemente arrastrar una conexión desde un objeto de interface de usuario hacia código fuente existente.

Lo mejor, es que si aún no se dispone de código fuente para las acciones (eventos) o variables de interface de usuario, se generará automáticamente al arrastrar una conexión desde el objeto deseado hasta un espacio en blanco en el editor de código fuente.

A partir de la versión 4.2 de *XCode* se incorporaron los *Storyboards*, (Fig. 1.4) que permiten usar *Interface Builder* para diseñar todas las pantallas de una aplicación, junto a las transiciones entre las mismas y los controles utilizados para activar estas transiciones. De esta manera se pueden trazar gráficamente todos los caminos posibles de la aplicación, reduciendo la cantidad de código que se necesitaba anteriormente para una aplicación compuesta de varias ventanas.

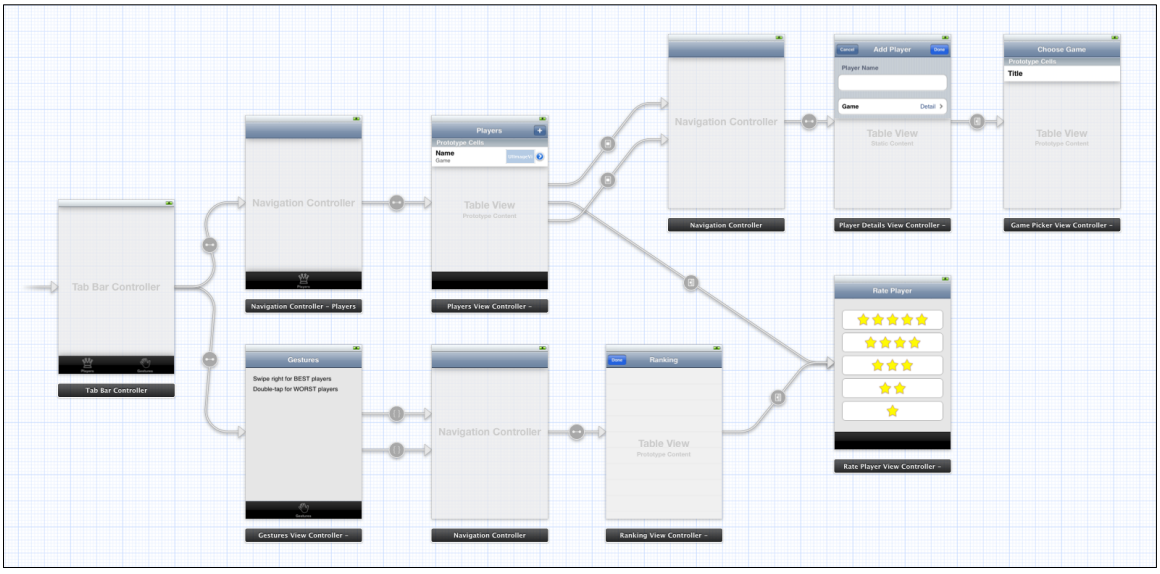


Fig. 1.4: Ejemplo de un *Storyboard*

1.3 iOS Simulator

El *iOS Simulator* incluido en el *IDE XCode* permite construir, instalar, ejecutar y depurar las aplicaciones desarrolladas para dispositivos *iOS* en un computador *Mac*. Permite simular tres dispositivos *iOS*: (*iPhone (iPod), iPhone* con pantalla de retina e *iPad*) incluyendo muchas versiones de *firmware iOS*.

Fig. 1.5: *iOS Simulator* (Orientación *Portrait*)

Su principal característica es la capacidad de simular la mayoría de acciones que un usuario puede realizar en un dispositivo *iOS* físico. Entre éstas se encuentran: rotar a la izquierda, rotar a la derecha, realizar el gesto de sacudir, presionar el botón home y el botón de bloqueo. Se pueden simular advertencias de memoria e incluso simular el uso de un teclado *hardware* externo para el caso de *iPad*.

En cuanto a gestos con los dedos sobre la pantalla, se pueden simular los siguientes: presionar, pulsar y mantener pulsado, doble toque, deslizar, arrastrar, entre muchos otros[6].

[6] Joshua, N. (2011). *Mastering XCode 4: Develop & Design.* Berkeley, California, Estados Unidos: Peachpit Press. p263-p264

Al simular un dispositivo *iOS* físico, permite instalar y desinstalar aplicaciones en el mismo. De igual manera permite visualizar en el IDE, la consola y los registros de errores inesperados producidos mientras se ejecutan las aplicaciones en el simulador.

Lamentablemente, en la actualidad el simulador no permite emular la cámara de un dispositivo *iOS* físico, ni su acelerómetro, el cual es el encargado de detectar el movimiento y el giro en un dispositivo iOS.

Fig. 1.6: *iOS Simulator* (Orientación *Landscape*)

1.4 Servicio Web de Amazon: AWS Simple DB

Es un almacén de datos no relacionales flexible y de alta disponibilidad que facilita el trabajo de administración de bases de datos. Los desarrolladores se encargan de almacenar elementos de datos y consultarlos mediante servicios web, *Amazon SimpleDB* realiza el resto de tareas.

Se encuentra optimizado para ofrecer alta disponibilidad y flexibilidad con poca o nula carga administrativa. Su labor, es crear y gestionar réplicas de datos y distribuirlas geográficamente para permitir alta disponibilidad y capacidad de duración. El servicio cobra únicamente por recursos realmente consumidos en almacenamiento de datos y distribución de solicitudes.

Se puede cambiar el modelo de datos en cualquier momento, y el sistema se encargará de indexar los datos automáticamente por el contratante.

Funcionalidad: *Amazon SimpleDB* ofrece una interface de servicios web para crear y almacenar varios conjuntos de datos, consultarlos y obtener resultados de dichas consultas. La información es encontrada enseguida gracias al indexado de datos que ofrece.

Fijación de Precios: Se paga únicamente por los recursos consumidos, sin una cuota mínima. Se puede optar por un plan gratuito en el cual los clientes reciben gratis 25 horas de máquina para *SimpleDB* y 1GB de almacenamiento mensualmente. Muchas aplicaciones podrían funcionar permanentemente dentro de los límites de este plan gratuito[7].

[7] Amazon. (2012). *Amazon SimpleDB.* Recuperado el 27 de Julio de 2012, de Amazon SimpleDB: http://aws.amazon.com/es/simpledb/

Estructuración de datos: El modelo de datos que utiliza *Amazon SimpleDB* facilita el almacenamiento, gestión y consulta de datos estructurados. Los conjuntos de datos se organizan en dominios en donde se pueden realizar consultas de todos los datos almacenados en un dominio concreto. Los dominios son colecciones de artículos descritos mediante pares atributo-valor. Para ejemplificar los dominios y los pares atributo-valor, se presenta el siguiente ejemplo[8]:

Se tomará los detalles de una base de datos de una gestión de clientes detallada en la Fig. 1.7 y se analizará como se representarían en *Amazon SimpleDB*

ID_Cliente	Nombre	Apellido	Dirección postal	Ciudad	Estado	CP	Teléfono
123	Bob	Smith	123 Main St	Springfield	MO	65801	222-333-4444
456	James	Johnson	456 Front St	Seattle	WA	98104	333-444-5555

Fig. 1.7: Base de datos de gestión de clientes[9]

La tabla completa se constituirá en un dominio llamado "clientes". Cada cliente será una fila de la tabla o un elemento de dominio. La información de contacto se describe en los encabezados de columna (atributos). Cada valor ocupa su propia celda.

Para añadir los registros se utiliza la instrucción *PUT* (colocar) las ID_Cliente en el dominio, junto con los pares atributo-valor correspondientes al cliente, el resultado sería similar al siguiente:

PUT (elemento, 123), (Nombre, Bob), (Apellido, Smith), (Dirección postal, 123 Main St), (Ciudad, Springfield), (Estado, MO), (CP, 65801), (Teléfono, 222-333-4444)

Amazon SimpleDB se diferencia de bases de datos tradicionales en el aspecto en que es flexible y representa esta flexibilidad de manera en que se pueden añadir atributos

[8] Amazon. (2012). *Amazon SimpleDB.* Recuperado el 27 de Julio de 2012, de Amazon SimpleDB: http://aws.amazon.com/es/simpledb/#details

[9] Amazon. (2012). *Amazon SimpleDB.* Recuperado el 27 de Julio de 2012, de Amazon SimpleDB.

específicos que apliquen solo a determinados registros. Por ejemplo, si se desea añadir el campo "Correo Electrónico" a la tabla anterior, tradicionalmente se debería volver a construir la tabla "clientes", volver a escribir las consultas, volver a construir índices, etc.; con *Amazon SimpleDB* basta con añadir al dominio llamado "clientes" los registros nuevos junto a los atributos adicionales (pares atributo-valor)[10]. El dominio resultante sería similar al de la Fig. 1.8.

ID_Cliente	Nombre	Apellido	Dirección postal	Ciudad	Estado	CP	Teléfono	Correo electrónico
123	Bob	Smith	123 Main St	Springfield	MO	65801	222-333-4444	
456	James	Johnson	456 Front St	Seattle	WA	98104	333-444-5555	
789	Deborah	Thomas	789 Garfield	New York	NY	10001	444-555-6666	dthomas@xyz.com

Fig. 1.8: Dominio resultante al añadir campos específicos para registros[11]

***APIs*:** Las *API* que incluye *Amazon SimpleDB* implementan la escritura, indexación y consulta de datos. La interface se centra en las funciones principales, proporcionando una *API* básica para que a los contratantes les resulte sencillo utilizar el servicio.

CreateDomain: Crea un dominio que contendrá conjuntos de datos**.**

DeleteDomain: Elimina un dominio

ListDomains: Genera un listado de los dominios creados.

DomainMetadata: Recupera información de: hora de creación del dominio, de almacenamiento (nombres y atributos de elementos junto al tamaño total en *bytes*).

PutAttributes: Agrega o actualiza un elemento y sus atributos, o añade pares atributo-valor a elementos existentes.

[10] Amazon. (2012). *Amazon SimpleDB*. Recuperado el 27 de Julio de 2012, de Amazon SimpleDB: http://aws.amazon.com/es/simpledb/#details
[11] Amazon. (2012). *Amazon SimpleDB*. Recuperado el 27 de Julio de 2012, de Amazon SimpleDB.

BatchPutAttributes: Realiza hasta 25 operaciones *PUT* en una sola instrucción.

DeleteAttributes: Elimina un elemento, atributo o valor de atributo.

GetAttributes: Recupera un elemento y un subconjunto de sus atributos o valores

Select: Consulta un conjunto de datos con la sintaxis conocida *"select objective from* domain_name *where* expression_query*"*. Admite los valores: *=, !=, <, >, >=, <=, like, not like, between, is null, is not null, y every()*. Permite ordenar resultados mediante la instrucción *SORT* y contar los elementos que cumplen ciertas condiciones mediante *Count*.[12]

[12] Amazon. (2012). *Amazon SimpleDB*. Recuperado el 27 de Julio de 2012, de Amazon SimpleDB.

1.5 iBooks Author

Herramienta de *Apple* que se encuentra disponible de forma gratuita y sirve para crear libros *Multi-Touch* para dispositivos *iPad.* El objetivo de esta herramienta es el de permitir crear libros que integren contenidos que son imposibles para los libros de papel, como: galerías, videos, objetos 3D, entre otros[13].

Se utilizará esta herramienta para desarrollar un *iBook* cuyos contenidos serán los mismos de éste tutorial, de manera de que los interesados puedan visualizar el tutorial directamente en sus dispositivos *iPad*.

El *iBook* a desarrollar permitirá a los lectores interactuar con las figuras del mismo, así como abrir directamente los enlaces web. Además presentará un índice de contenidos interactivo que permitirá al lector dirigirse directamente al sub apartado de su interés.

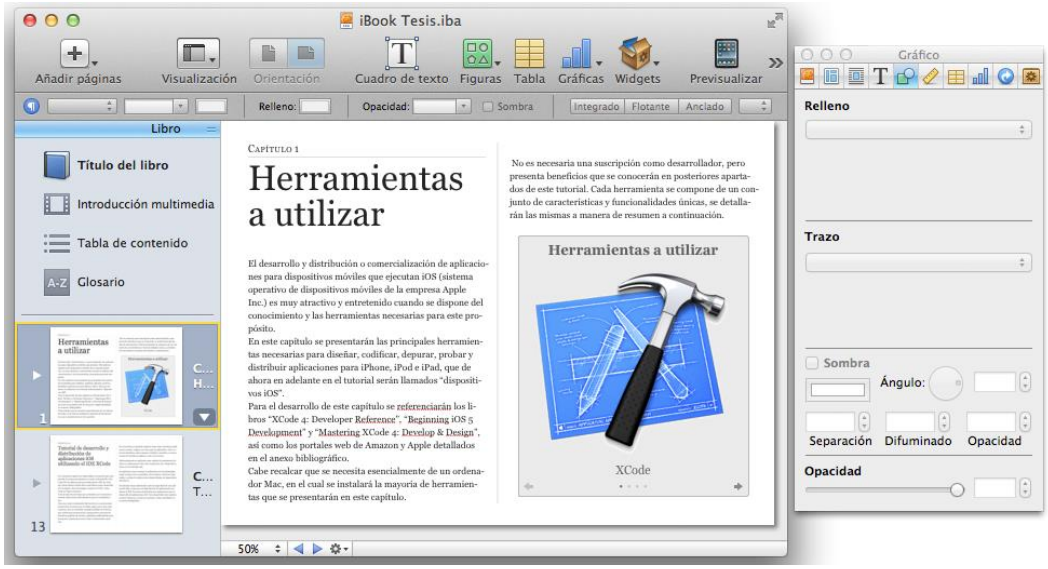


Fig. 1.9: Herramienta de Apple iBooks Author

[13] Apple. (2011). *iBooks Author*. Recuperado el 9 de Junio de 2013, de iBooks Author: http://www.apple.com/es/ibooks-author/

La creación de éste *iBook* se realizará de la siguiente manera:

- Se utilizará ésta herramienta: *iBooks Author.*
- Se decidirá la plantilla a partir de la cuál se desarrollará el *iBook:* Entre las plantillas disponibles en la herramienta, se seleccionará la denominada "Básica" debido a que se la considera formal y adecuada para éste tutorial a desarrollar.
- Al finalizar la redacción de un apartado o sub apartado de éste tutorial: Se dedicará un tiempo para redactarlo dentro del *iBook* con los mismos contenidos y con figuras interactivas para los usuarios. Para la introducción a un nuevo capítulo se utilizará el componente "Capítulo", para cada apartado y subapartado se utilizará componentes "Sección" y para los contenidos de los apartados y sub apartados se utilizará componentes "Página". Para las figuras interactivas se agregará un pie de foto para cada imagen que se agregue, lo que permite que el usuario al presionar una imagen la pueda visualizar en pantalla completa. Para esto se utiliza el inspector de atributos de objeto que se puede observar en la parte derecha de la Fig. 1.9
- Se generará un índice de temas o secciones interactivo: Éste índice es automáticamente generado al agregar componentes "Capítulo", "Sección" y "Páginas" al *iBook*. La herramienta utiliza el título de cada componente para generar un índice interactivo, además crea un enlace hacia la página deseada al presionar el título de cualquiera de estos componentes en el índice
- Se probará el *iBook* de manera continua en un dispositivo *iPad* de manera de comprobar su correcto funcionamiento, sus contenidos y la interacción con las imágenes del mismo.
- Se presentará el *iBook* como anexo de éste trabajo de graduación.

Conclusiones

Al concluir este capítulo, se han presentado las principales herramientas necesarias para el desarrollo de aplicaciones para dispositivos *iOS*, que son principalmente: el *IDE XCode*, *Interface Builder*, *iOS Simulator* y el Servicio Web de *Amazon SimpleDB*.

Se han presentado las principales funciones de cada herramienta, y de esta forma se ha permitido al lector conocer que actividades realizará con cada una de las herramientas necesarias para el desarrollo de aplicaciones para dispositivos *iOS*.

Adicionalmente, se ha detallado la forma en la que se utilizará la herramienta *iBooks Author* para crear el *iBook* que se presentará como anexo a este trabajo de graduación.

Es muy importante que el lector haya conocido y se haya familiarizado con las características y ventajas que presenta el *IDE XCode*, debido a que con ésta herramienta podrá desarrollar e incursionar en la comercialización de aplicaciones para dispositivos *iOS*.

Capítulo 2: Tutorial de desarrollo y distribución de aplicaciones iOS utilizando el IDE XCode

En el presente capítulo se desarrollará un tutorial que comprenda los temas principales en cuanto al desarrollo y distribución de aplicaciones para dispositivos *iOS.* Se tratarán temas básicos desde cómo suscribirse como desarrollador de *Apple,* cómo descargar e instalar el *IDE* y cómo crear un nuevo proyecto.

Para la mayoría de temas que se detallen en el tutorial se crearán aplicaciones demostrativas que los complementen.

Para una mejor compresión del tutorial, no se presentará únicamente los pasos que se deben seguir para crear aplicaciones, sino se realizarán también análisis de: ficheros que conforman los proyectos, componentes comunes de interfaces gráficas de usuario, plantillas prediseñadas para proyectos, transiciones entre vistas y *frameworks* comunes.

En el tutorial se pretende explicar cómo crear interfaces gráficas de usuario, asignar un ícono para la aplicación, cómo rotar las interfaces, cómo asignar variables y métodos a componentes de interfaces gráficas, entre otros temas.

Adicionalmente se explicará como realizar la persistencia de datos en aplicaciones *iOS,* tanto localmente en el dispositivo, como en un servidor web.

Se explicará como conectar la aplicación con las principales redes sociales de la actualidad, cómo utilizar archivos multimedia y probar las aplicaciones desarrolladas en dispositivos *iOS* físicos.

Se tratarán temas adicionales como la seguridad de una aplicación *iOS* y el proceso de distribución de aplicaciones mediante el *IDE*. Se presentará finalmente sugerencias para el desarrollo de aplicaciones *iOS*. Para desarrollar este capítulo se hará referencia a todos los portales y libros detallados en el anexo bibliográfico.

2.1 Cómo instalar el IDE XCode y sus componentes en Mac OS X Lion

Fig. 2.1: Descarga e instalación de *XCode*[14]

Para empezar se debe descargar el *IDE XCode* mediante de la aplicación *AppStore* incluida en los computadores *Mac* con sistema operativo *OS X Lion* y posterior. Para esto, se debe abrir la aplicación *AppStore* y escribir en el buscador "*XCode*", los resultados presentados serán similares los siguientes:

Fig. 2.2: Resultados de búsqueda *"XCode"* en *AppStore*

La versión actual de *XCode* 4.4 utiliza 3.16GB una vez instalado en el disco duro. Se debe proceder a descargarlo.

Fig. 2.3: Información sobre *XCode* en *AppStore*

Fig. 2.4: Instalación de *XCode*

[14] Apple. (2012). *Start Developing iOS Apps Today*. Recuperado el 28 de Julio de 2012, de Start Developing iOS Apps Today: http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/GetToolsandInstall.html

Una vez finalizada la descarga, el proceso de instalación comenzará automáticamente y se presentará una ventana similar a la Fig. 2.4, en la cual se debe presionar el botón "*Install*" para comenzar la instalación. Cuando se presione este botón, se presentará el acuerdo de licencia (Fig. 2.5), el cual debe ser aprobado presionando el botón "*Agree*".

Fig. 2.5: Acuerdo de licencia

Fig. 2.6: *XCode* en proceso de instalación

Una vez aceptado el acuerdo de licencia, *XCode* y sus componentes se instalarán en el computador *Mac*, tardará unos cuantos minutos debido a la capacidad del mismo (Fig. 2.6). Cuando finalice la instalación, se notificará que el proceso ha finalizado exitosamente (Fig. 2.7). A partir de este momento se puede trabajar con *XCode* y cada uno de sus componentes, que, como se puede observar en los programas instalados (Fig. 2.8), se han ubicado en una carpeta denominada: "*Developer*".

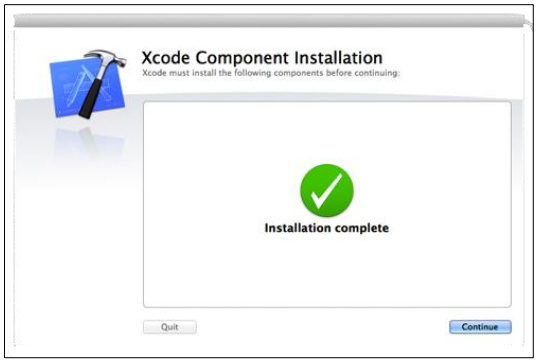


Fig. 2.7: Instalación finalizada exitosamente

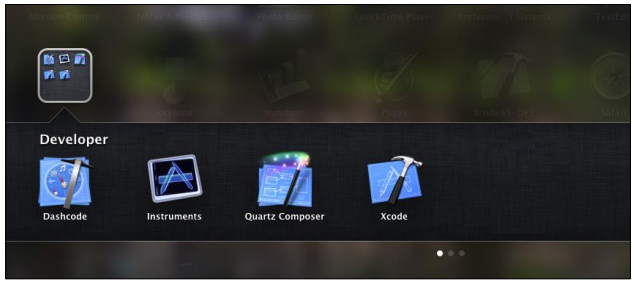


Fig. 2.8: Carpeta *"Developer"*

2.1.1 Suscripción como desarrollador de Apple

Si no se desea una suscripción como desarrollador de *Apple*, se puede omitir este paso. Al suscribirse en este programa, el desarrollador podrá probar sus aplicaciones en dispositivos físicos y distribuirlas o comercializarlas en la tienda *App Store*, caso contrario solo podrá probar las aplicaciones en el simulador *iOS* y no las podrá distribuir. La suscripción en el programa tiene un costo de $99 por año.

Para suscribirse, se debe dirigir al enlace:

https://developer.apple.com/programs/ios/ y presionar el botón *"Enroll Now"*.

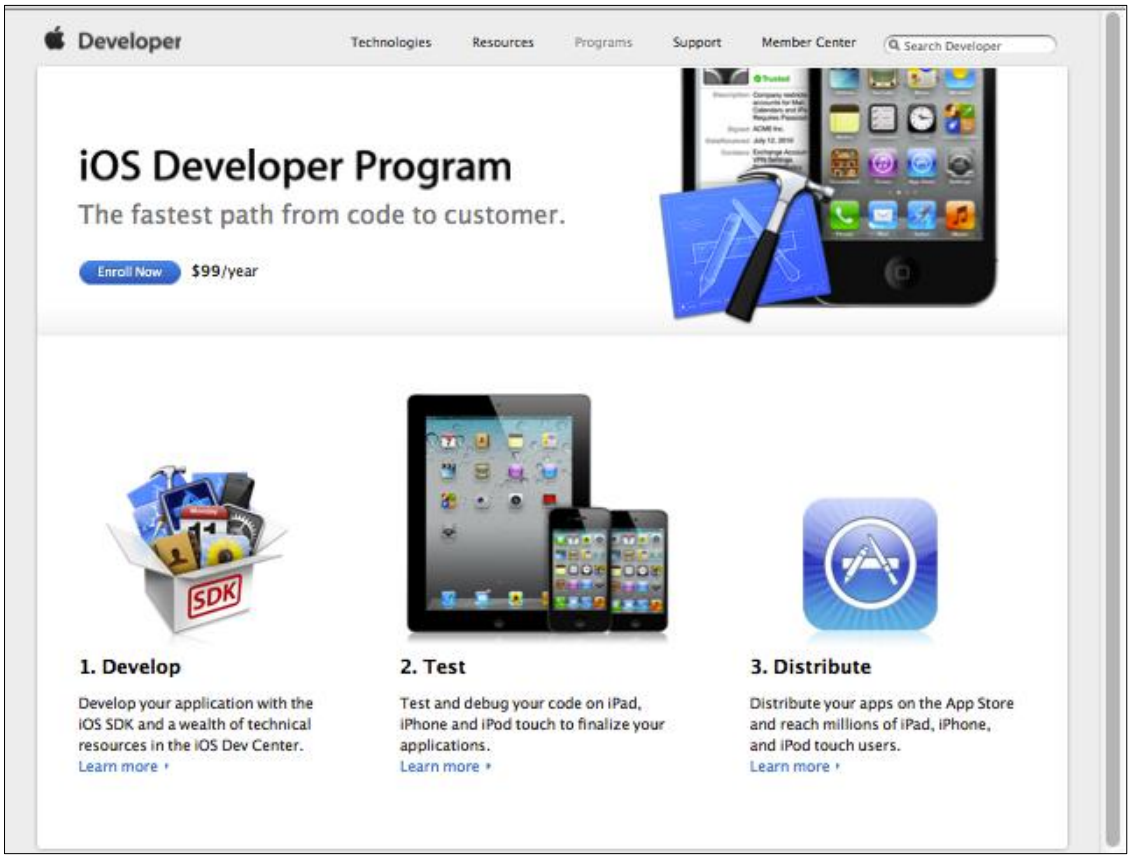


Fig. 2.9: Programa de desarrollador de iOS[15]

[15] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/

A continuación, se presentarán las indicaciones de suscripción (Fig. 2.10), que son las siguientes:

- **Seleccionar un tipo de suscripción:** Se puede suscribir como una persona individual o como una compañía u organización
- **Ingresar información:** Se debe proveer información personal básica, incluyendo nombre legal y dirección.
- **Comprar y activar el programa:** Después de que la información proveída sea verificada, se puede comprar el programa en la tienda en línea de *Apple*, al hacerlo, se recibirá un correo en 24 horas con indicaciones sobre como activar la membresía.

Proseguir presionando el botón "Continue".

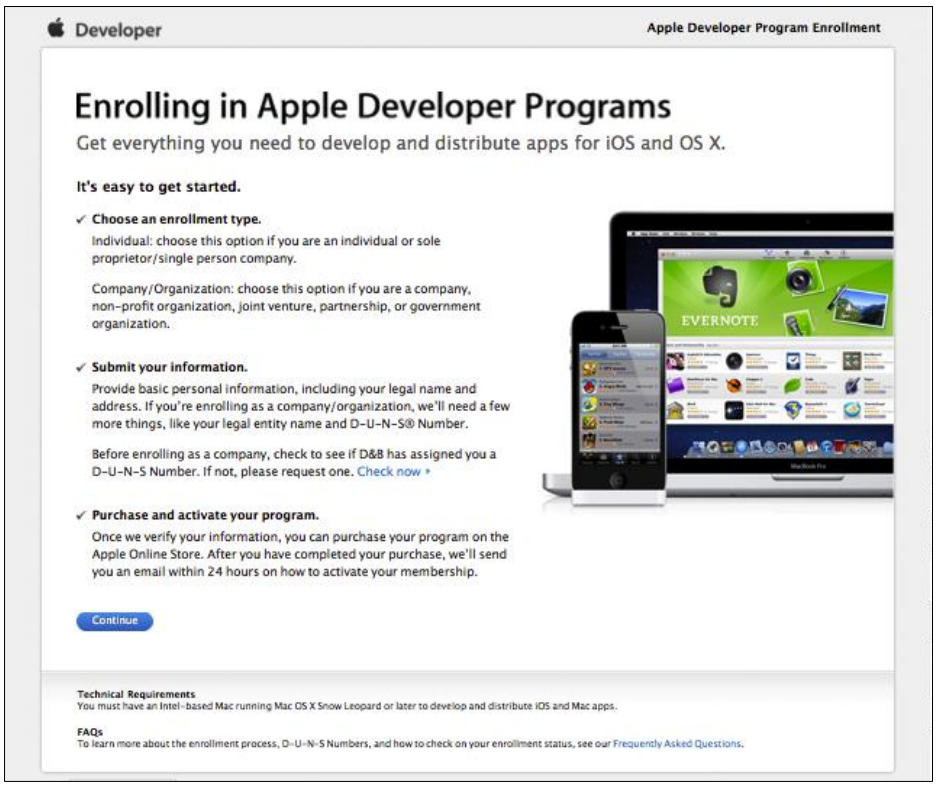


Fig. 2.10: Indicaciones de suscripción[16]

[16] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/start/standard/

A continuación se presentan los pasos de suscripción que son los siguientes:

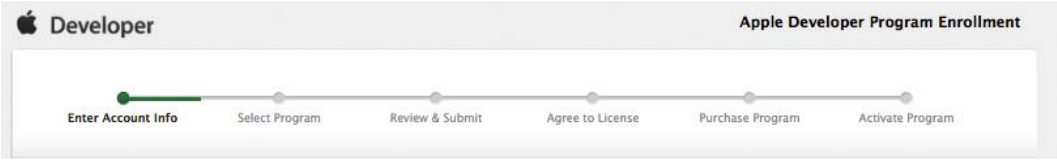


Fig. 2.11: Pasos para la suscripción[17]

Paso 1. Enter Account Info (Ingresar información de cuenta): En este paso se solicitará el ingreso de la cuenta de *Apple* del contratante, se puede usar la misma que se usa para comprar aplicaciones desde dispositivos *Apple* en *App Store* y sirve para utilizar servicios como *iTunes* o *iCloud.* En caso de no disponer de una, en este paso, se presenta la opción de crear una nueva cuenta.

Marcar "Registrar usando su ID de Apple" ("*Sign in with your Apple ID*") y presionar el botón *"Continue"*:

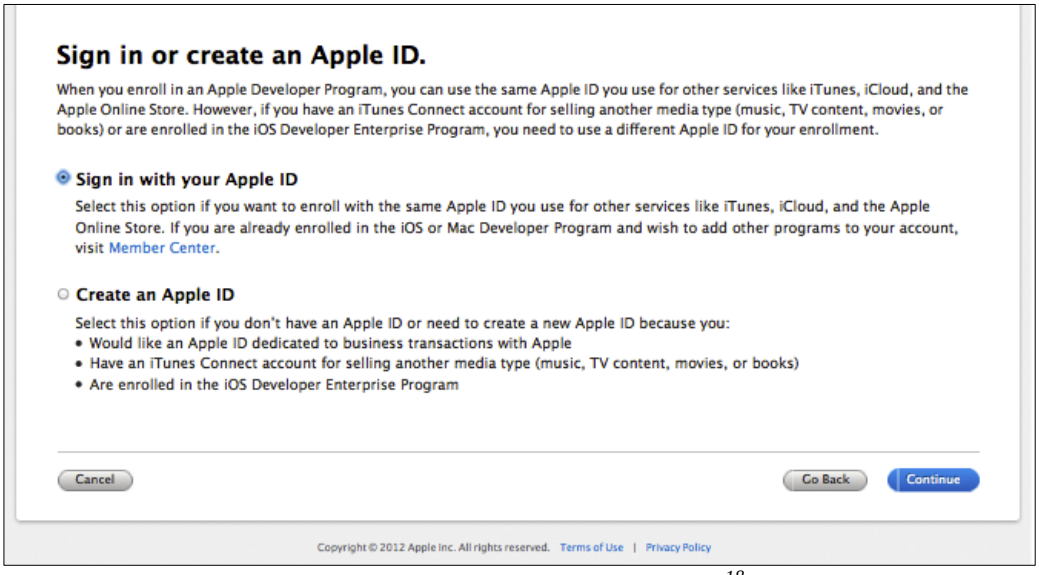


Fig. 2.12: Registrar usando cuenta de *Apple*[18]

[17] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/enroll/selectEnrollmentType.php?t=cm
[18] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/start/standard/create.php

Se debe elegir ahora, si se desea suscribir como una persona individual o como una compañía. En caso de suscripción como una persona individual, el nombre del suscriptor aparecerá como vendedor de la aplicación en la tienda *App Store*, caso contrario aparecerá el nombre de la compañía suscrita.

A continuación se explicará una suscripción como una persona individual. Para esto, se debe continuar presionando el botón *"Individual"*.

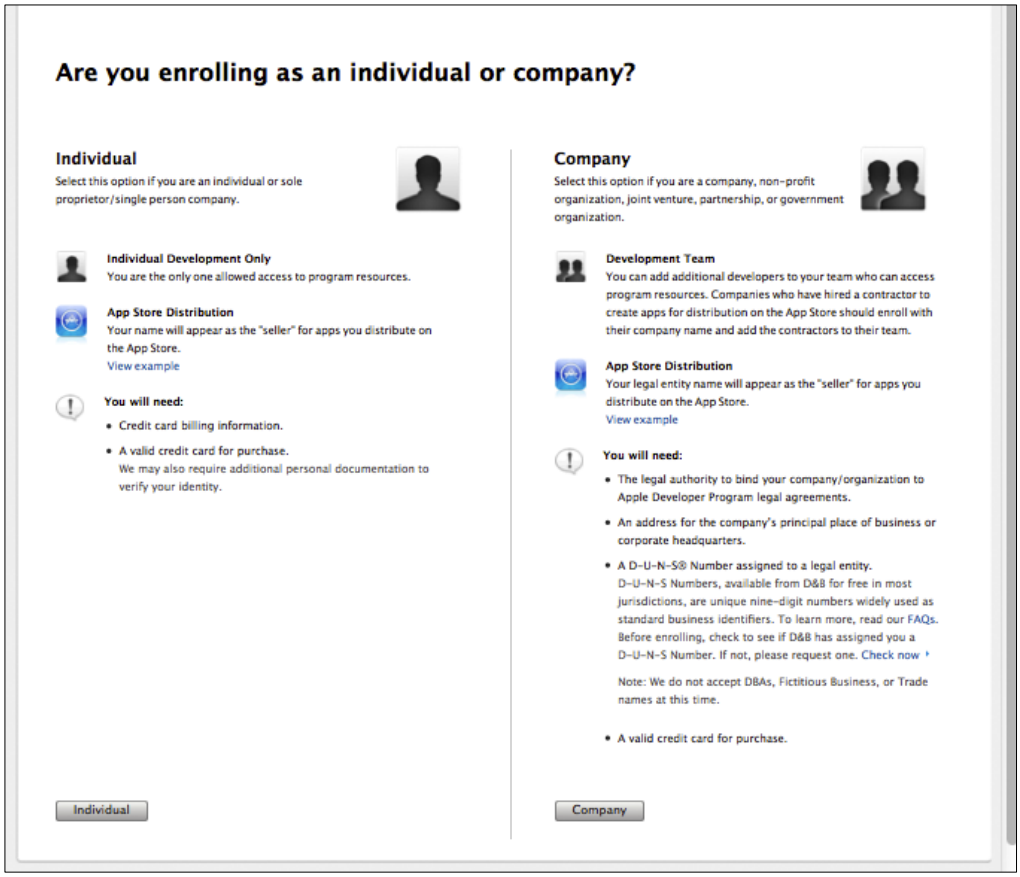


Fig. 2.13: Selección de tipo de suscripción: individual / compañía[19]

[19] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/enroll/selectEnrollmentType.php?t=cm

Al hacerlo, se solicitará el *Apple ID* y la contraseña del desarrollador, se los debe ingresar y continuar presionando el botón *"Sign In"*.

Fig. 2.14: Suscripción mediante la cuenta de *Apple*[20]

Acto seguido, el desarrollador debe ingresar su información personal: nombre, apellido, correo electrónico, compañía, país, dirección, ciudad, estado, código postal y teléfono. Como se puede observar en la ventana de la Fig. 2.15, *Apple* maneja esta información de manera confidencial[21]. Todos los campos son requeridos, por lo que si se deja alguno en blanco no se puede proseguir.

Llenar los datos y continuar presionando el botón *"Continue"*.

[20] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/
[21] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/ios/enroll/reviewEnrollment.action

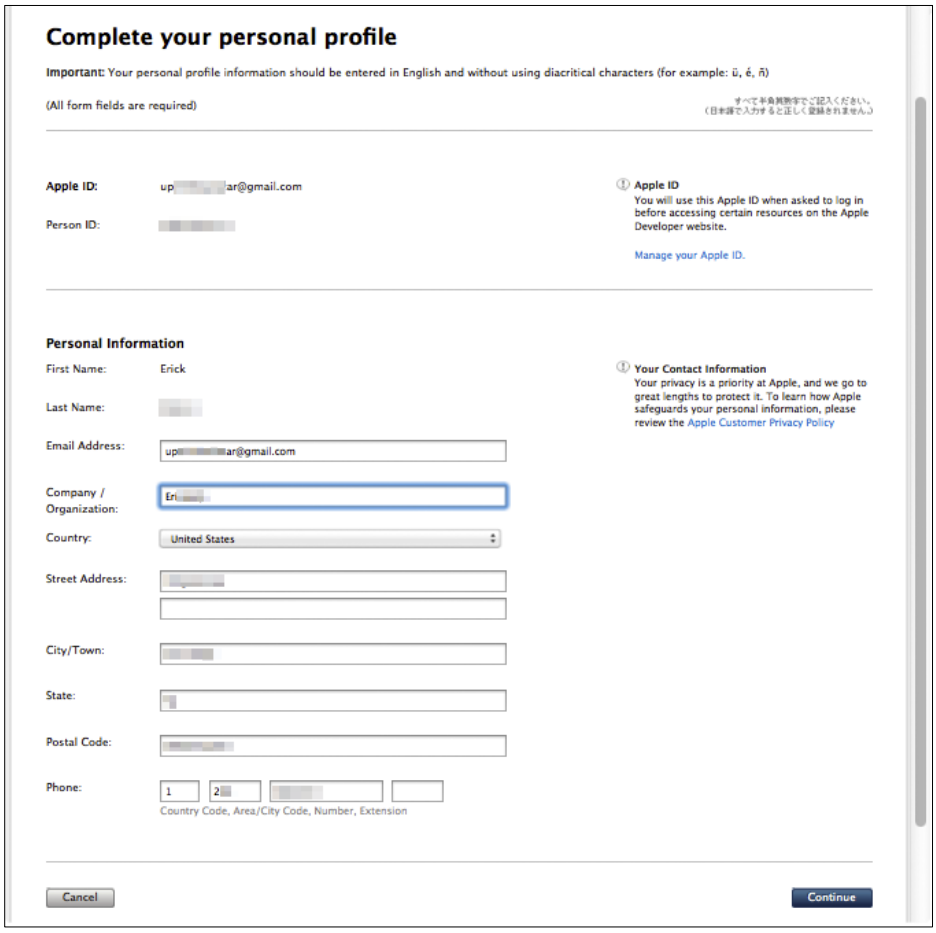


Fig. 2.15: Llenar el formulario con la información personal correspondiente[22]

En la siguiente ventana se deben contestar varias preguntas, marcando distintas casillas, entre estas preguntas se encuentran:

- ¿Para que plataformas de *Apple* desarrolla?
- ¿Cual es su principal mercado?
- ¿Que tipo de aplicaciones planea desarrollar?
- ¿Cual es la principal categoría de sus aplicaciones?
- ¿Cuántos años desarrolla para las plataformas *Apple*?
- ¿Desarrolla para otras plataformas?

[22] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/ios/enroll/reviewEnrollment.action

Se deben seleccionar todas las casillas que correspondan a las respuestas de cada pregunta como se indica en la Fig. 2.16 y proseguir presionando el botón *"Continue"*.

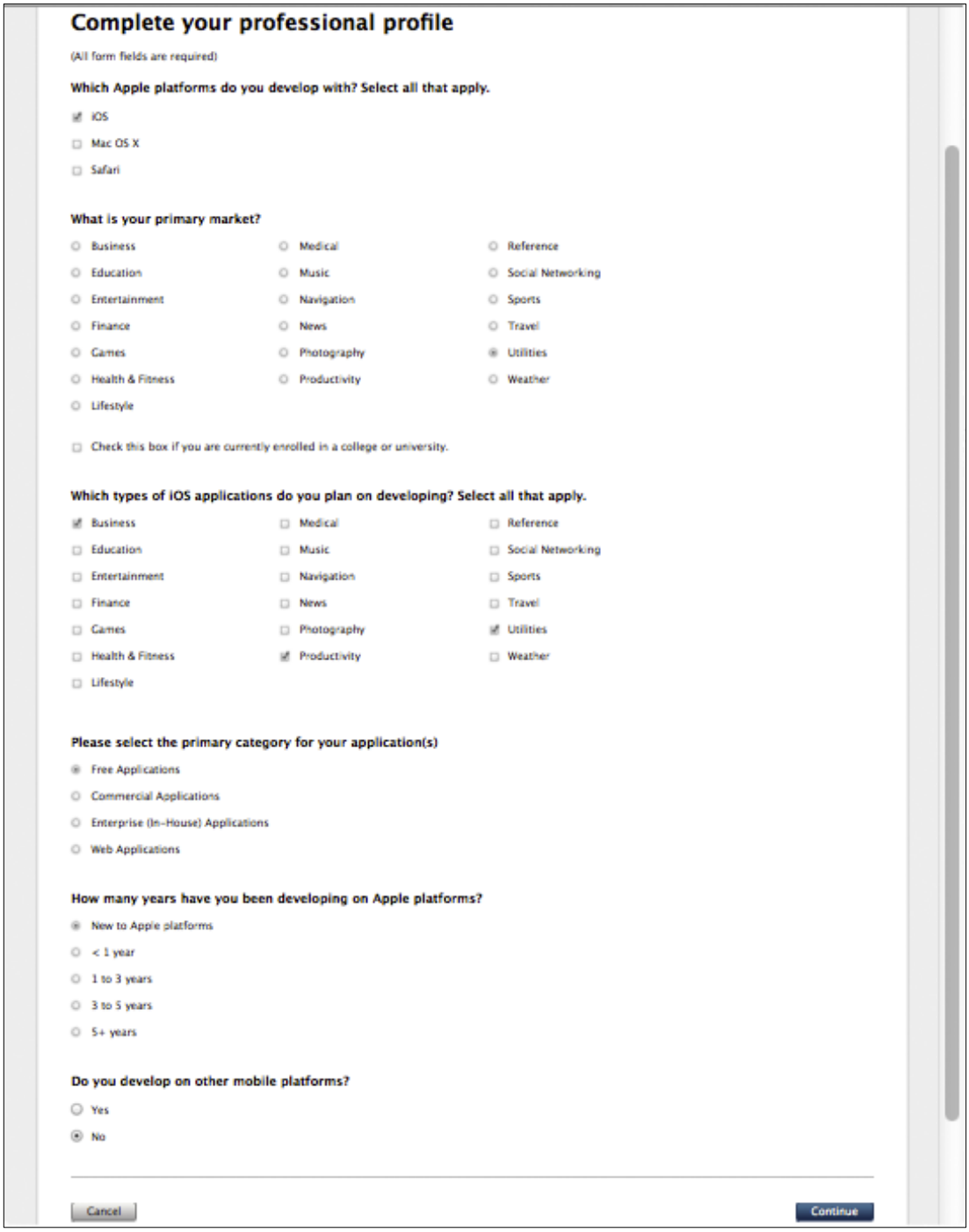


Fig. 2.16: Marcar las categorías y tipos de aplicaciones que se planea desarrollar[23]

[23] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/

El siguiente paso consiste en aprobar el acuerdo de registro como desarrollador, el cual se debe leer para comprender las políticas de *Apple*, marcar la casilla para indicar el acuerdo con el mismo, y presionar el botón *"I Agree"*.

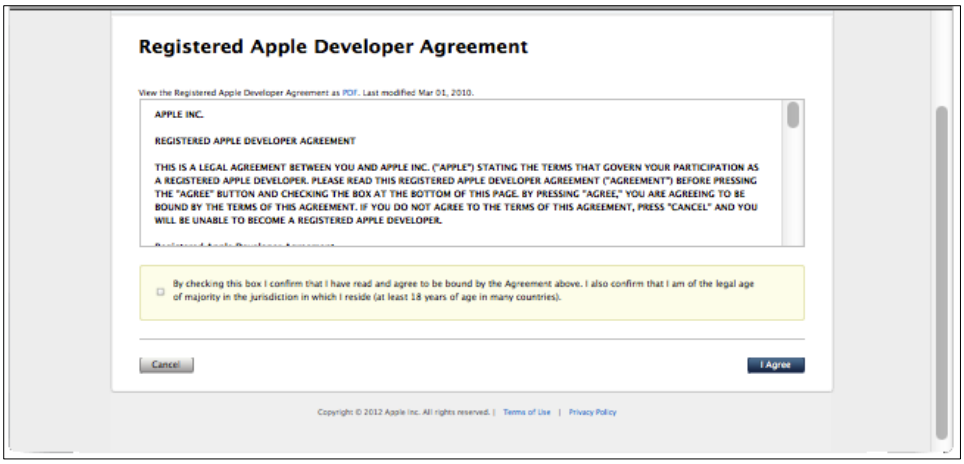


Fig. 2.17: Acuerdo de registro como desarrollador de *Apple*[24]

Enseguida se recibe un correo de confirmación de *Apple* para verificar la dirección de correo electrónico proveído, el cual contiene un número de confirmación que se debe ingresar en la siguiente ventana y presionar el botón *"Continue"*.

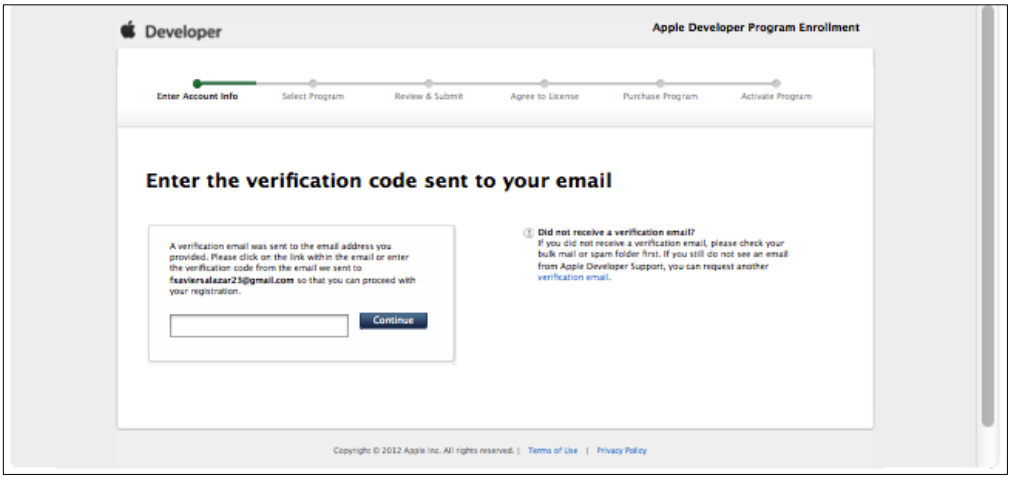


Fig. 2.18: Ingresar el código de verificación recibido[25]

[24] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/
[25] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program.

Se deben ingresar a continuación los datos de facturación. El nombre y apellido deben ser ingresados tal como se muestran en la tarjeta de crédito del contratante, mientras que la dirección, debe ser ingresada como se muestra en el último estado de cuenta de la tarjeta. Proseguir presionando el botón *"Continue"*.

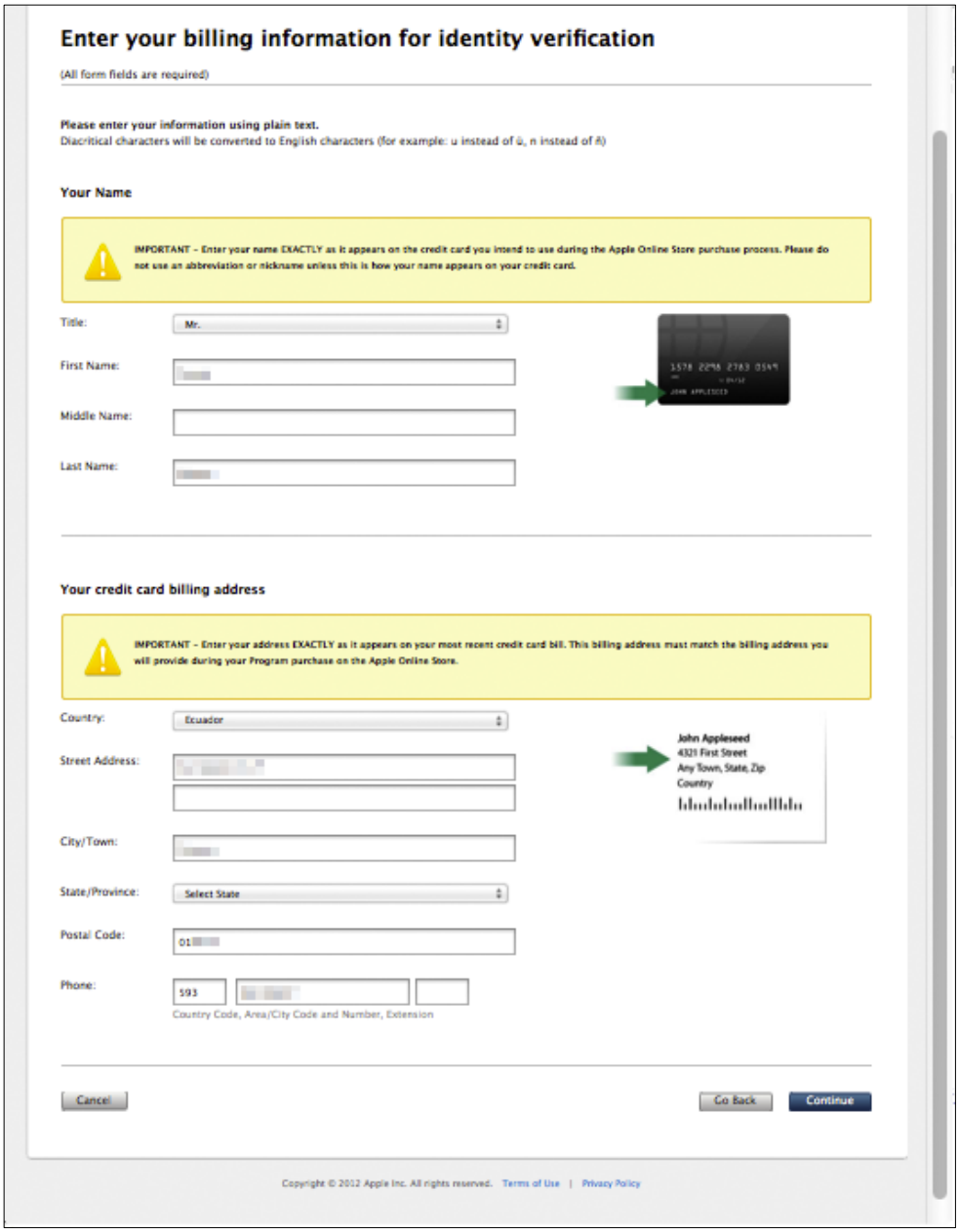


Fig. 2.19: Formulario de facturación[26]

[26] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/

Paso 2. *Select Program (Selección de programa)*: En este paso se debe seleccionar el tipo de programa en el cual se desea la suscripción, se puede seleccionar entre los programas de suscripción como desarrollador de *iOS*, *Mac* y *Safari*. Seleccionar *iOS Developer Program* debido a que se pretenden desarrollar aplicaciones para dispositivos *iOS* y presionar el botón *"Continue"*.

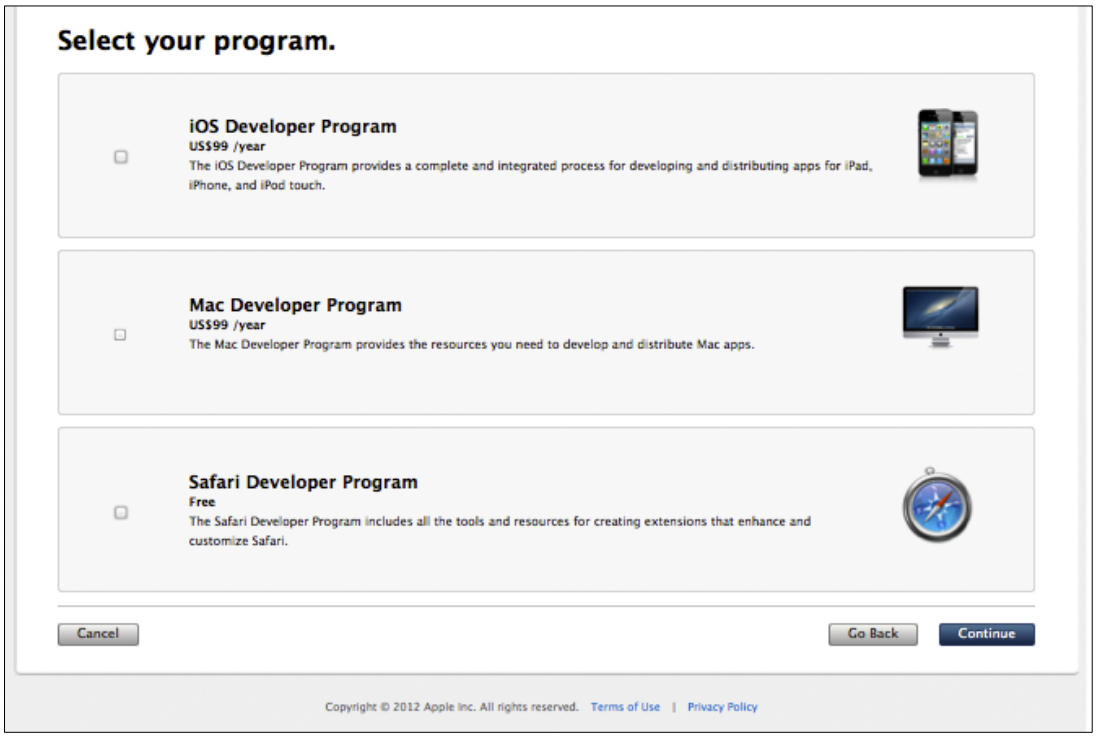


Fig. 2.20: Selección de programas en los que puede suscribirse el desarrollador[27]

[27] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/ios/enroll/selectProgram.action

Paso 3. *Review & Submit (Revisión)*: En este tercer paso se presentan los datos ingresados, se debe revisar que se los haya ingresado correctamente, debido a que el nombre ingresado aparecerá como vendedor de la aplicación en la tienda *App Store* y la facturación de las ventas llegarán digitalmente al correo electrónico y físicamente a la dirección proveída.

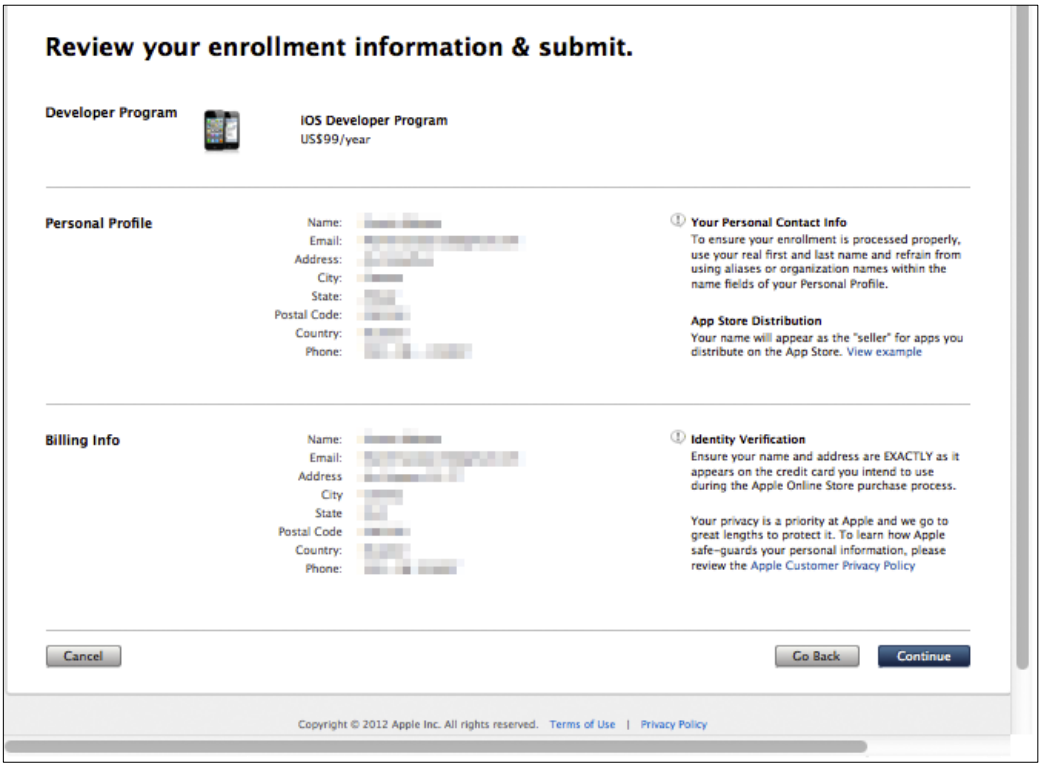


Fig. 2.21: Revisión de información ingresada[28]

[28] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/ios/enroll/reviewEnrollment.action

Paso 4. ***Agree to Licence (Aprobar el acuerdo de licencia)*****:** El cuarto paso consiste en leer y aceptar el acuerdo de licencia del programa de desarrollador de *iOS*. Para mayor comprensión, puede ser visualizado en distintos idiomas y descargado en formato *PDF*. Si se encuentra de acuerdo con las cláusulas, se debe marcar esta opción y continuar presionando el botón *"I Agree"*.

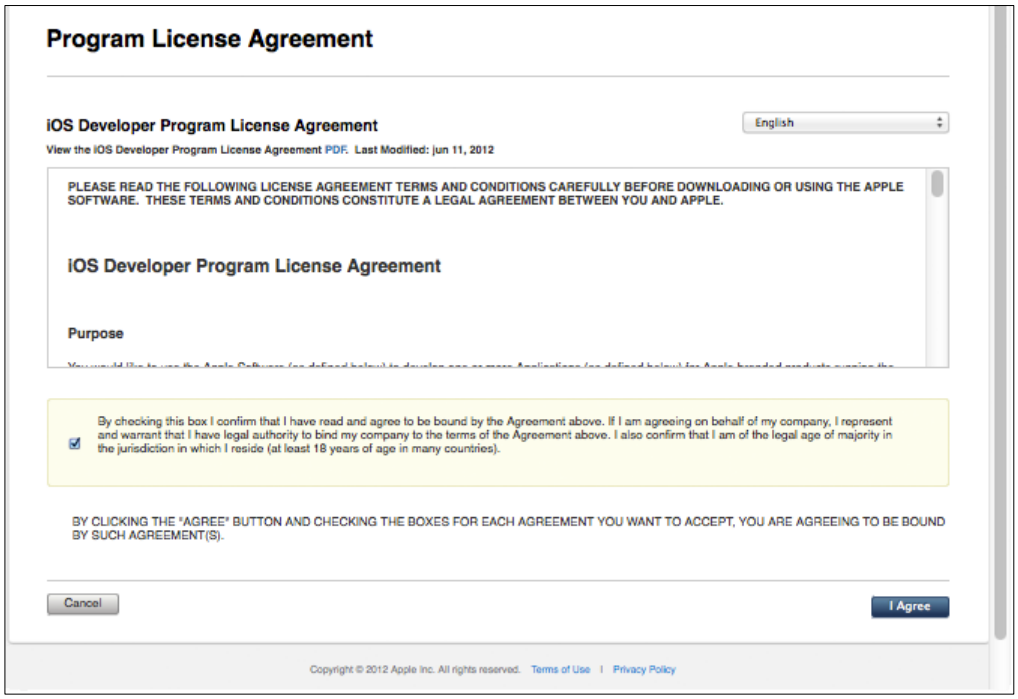


Fig. 2.22: Acuerdo de licencia de suscripción al programa de desarrollador *iOS*[29]

[29] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/

Paso 5. *Purchase Program* (Compra del programa): En el penúltimo paso se procede a comprar el programa, lamentablemente en el país en donde fue desarrollado este tutorial (Ecuador), no se dispone de una tienda en línea de *Apple*. Por lo tanto se presenta la siguiente pantalla:

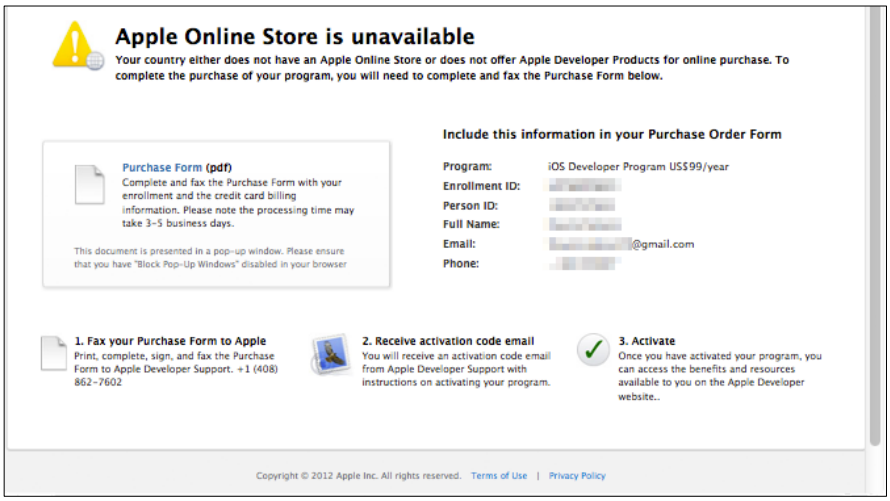


Fig. 2.23: Tienda en línea *Apple* no disponible[30]

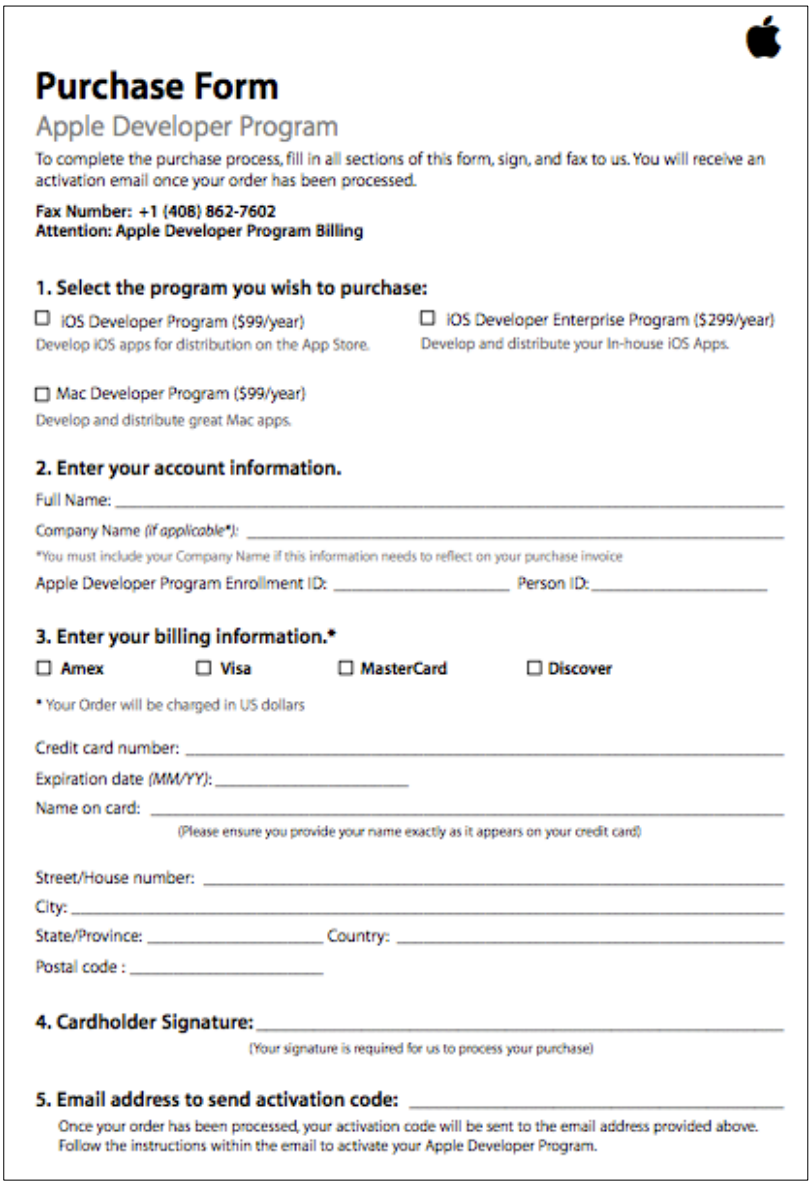
Lo que se debe hacer a continuación es lo siguiente:

- Descargar el formulario en formato *PDF*, completarlo manualmente y enviarlo por fax a las oficinas de soporte para desarrolladores de *Apple*: +1 (408) 862 – 7602
- Se recibirá un correo electrónico con el código de activación e instrucciones para activar el programa.
- Activar el programa (Paso 6) para poder acceder a todos los beneficios y recursos disponibles para los desarrolladores de *Apple*.

Nota: Este proceso puede tardar entre 3 y 5 días laborables.

[30] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/

Paso 6. *Activate Program* (Activación del programa): El último paso consiste en activar el programa comprado, lo que se debe hacer es descargar el formulario en formato *PDF* del paso anterior, el mismo que se presenta a continuación:



Purchase Form

Apple Developer Program

To complete the purchase process, fill in all sections of this form, sign, and fax to us. You will receive an activation email once your order has been processed.

Fax Number: +1 (408) 862-7602
Attention: Apple Developer Program Billing

1. Select the program you wish to purchase:

☐ iOS Developer Program ($99/year)
Develop iOS apps for distribution on the App Store.

☐ iOS Developer Enterprise Program ($299/year)
Develop and distribute your in-house iOS Apps.

☐ Mac Developer Program ($99/year)
Develop and distribute great Mac apps.

2. Enter your account information.

Full Name: _____

Company Name *(if applicable*)*: _____

*You must include your Company Name if this information needs to reflect on your purchase invoice

Apple Developer Program Enrollment ID: _____ Person ID: _____

3. Enter your billing information.*

☐ **Amex** ☐ **Visa** ☐ **MasterCard** ☐ **Discover**

* Your Order will be charged in US dollars

Credit card number: _____

Expiration date *(MM/YY)*: _____

Name on card: _____

(Please ensure you provide your name exactly as it appears on your credit card)

Street/House number: _____

City: _____

State/Province: _____ Country: _____

Postal code : _____

4. Cardholder Signature: _____

(Your signature is required for us to process your purchase)

5. Email address to send activation code: _____

Once your order has been processed, your activation code will be sent to the email address provided above. Follow the instructions within the email to activate your Apple Developer Program.

Fig. 2.24: Formulario de datos para compra del programa de desarrollador iOS[31]

Se debe imprimir, llenar y enviar por fax al número antes mencionado. Si se tiene problemas para enviar el fax, el servicio de *Apple* es tan eficiente que un encargado se contactará con el suscriptor vía correo o por teléfono, el objetivo de este contacto es

[31] Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/

para que se autorice la compra del programa, el cual será pagado con la tarjeta de crédito asociada a la cuenta de *Apple* del suscriptor, o con la indicada en el formulario en caso de que lo hayan recibido.

En el transcurso de las próximas 24 horas, se recibirá un correo de confirmación de activación de la cuenta como desarrollador de *iOS,* en el cual, *Apple* agradece por formar parte del programa de desarrolladores de la misma.

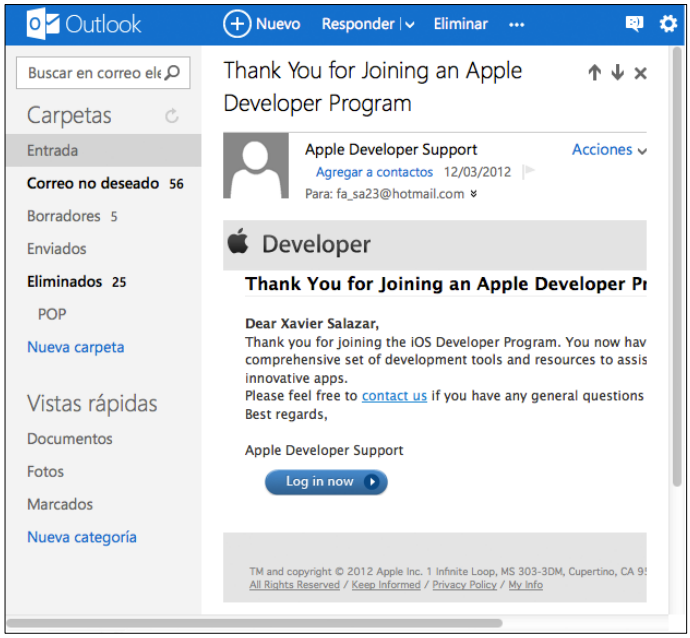


Fig. 2.25: Email de confirmación de activación de cuenta[32]

Nota: Si en cualquier momento se tiene problemas para continuar con el proceso de suscripción como desarrollador de *Apple*, se puede contactar con un especialista escribiendo un correo electrónico a la dirección devenroll@apple.com, los especialistas suelen responder a más tardar en 24 horas y son encargados de brindar ayuda con cualquier tipo de inconveniente (se asegura por experiencia personal del autor del tutorial).

[32] Outlook. (2012). Correo personal del desarrollador del tutorial. Recuperado el 6 de Agosto de 2012, de Outlook.

2.2 Cómo crear un nuevo proyecto

Una vez instalado el *IDE XCode*, se pueden desarrollar proyectos de aplicaciones tanto para dispositivos *iOS* como para computadores *Mac*. Se puede encontrar el ícono de la aplicación *XCode* en el *Dock* del *Mac*, ya que al instalarlo, crea por defecto un acceso directo en el mismo. En caso de no encontrar el acceso directo, se lo puede encontrar en el *Finder* en la carpeta de aplicaciones. Al abrirlo se presenta una ventana como la siguiente:

Fig. 2.26: Ventana inicial al abrir *XCode*

En esta ventana se encuentran los proyectos recientes en los que se ha trabajado, si se desea, se puede abrir otro proyecto presionando el botón *"Open Other..."*.

Se puede además visualizar la documentación o visitar el portal de desarrolladores de *Apple*. El objetivo del presente apartado es crear un nuevo proyecto por lo tanto, se debe presionar el botón situado en la parte superior:

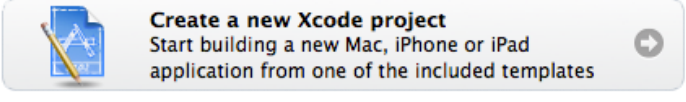


Fig. 2.27: Botón de nuevo proyecto

Las aplicaciones *iOS* son relativamente similares, esto es gracias a que *XCode*, incluye plantillas de diseños preestablecidos, de forma de que si se desea desarrollar una aplicación con estilo "Basado-en-páginas" o "Basado-en-Tabs", se puede seleccionar cualquiera de estas plantillas y desarrollar la aplicación a partir de la misma. Para conocer todas las plantillas prediseñadas que ofrece el *IDE XCode*, se puede referir al apartado "*2.4.3 Análisis de plantillas prediseñadas para un proyecto*".

Se explicará a continuación como crear un nuevo proyecto en blanco o vacío. En el modulo de la parte izquierda se debe indicar que se desea crear una aplicación *iOS* y acto seguido seleccionar *"Empty Application"*, como se indica en la Fig. 2.28.

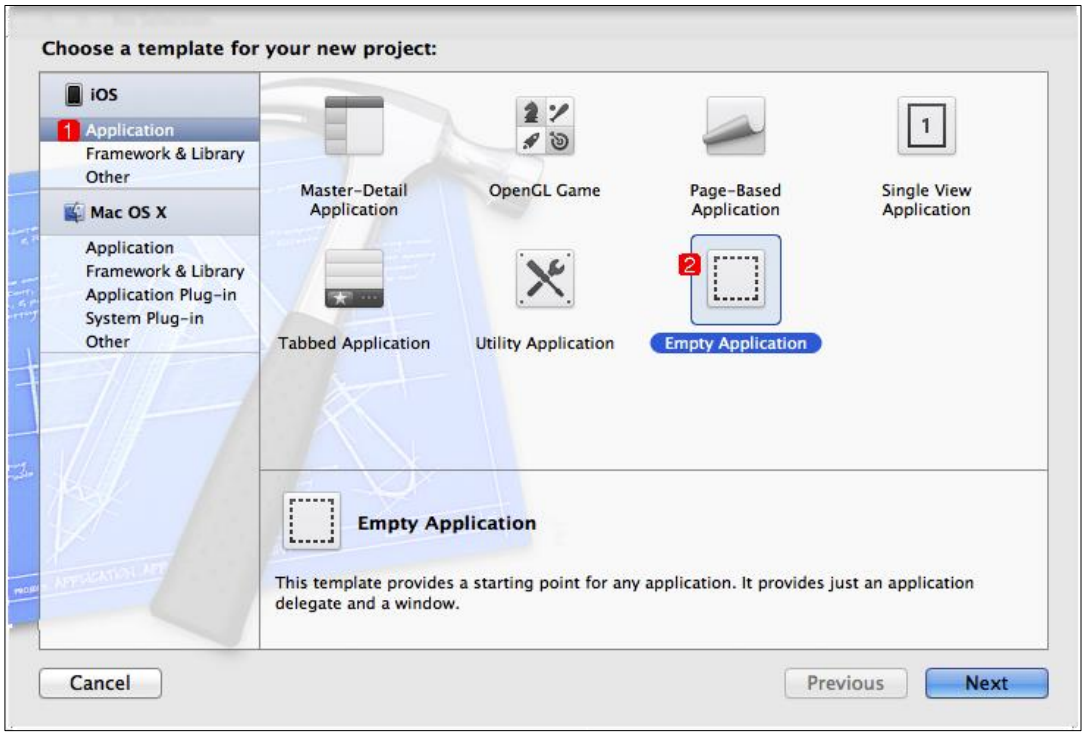


Fig. 2.28: Crear una aplicación en blanco o vacía

Empty Application: Esta "plantilla" es un punto inicial para cualquier aplicación; consta de un *AppDelegate*, los *frameworks* básicos y archivos de soporte. (Para conocer las características de un *AppDelegate* y estos archivos de soporte, se debe referir al apartado "*2.3 Análisis de tipos de posibles ficheros de un proyecto"*). Continuar presionando el botón *"Next"*.

A continuación se presenta la siguiente ventana:

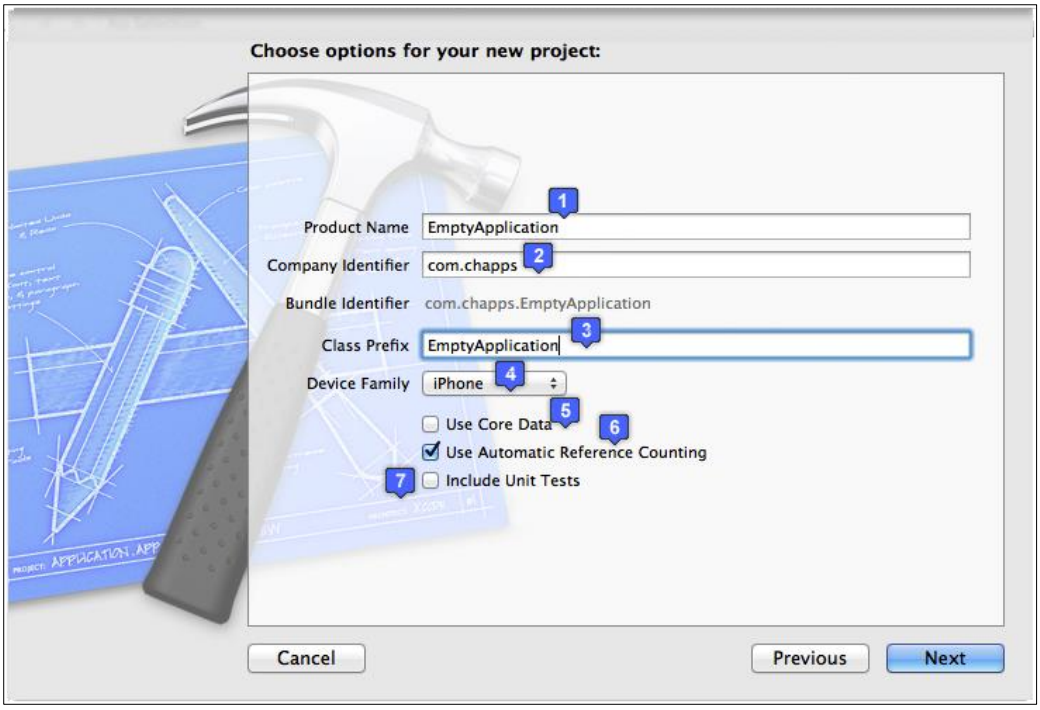


Fig. 2.29: Nombre y opciones para el proyecto

Se debe completar esta ventana con el nombre del proyecto y las opciones para el mismo. A continuación se detalla la información para cada campo:

1. *Product Name:* Se debe escribir aquí el nombre de la aplicación. Para ejemplificar, se la ha llamado *EmptyApplication* (o aplicación vacía). Este será el nombre para: el proyecto, su carpeta contenedora y la aplicación desarrollada.[33]

[33] Joshua, N. (2011). *Mastering XCode 4: Develop & Design.* Berkeley, California, Estados Unidos: Peachpit Press. p12, p13.

2. *Company Identifier:* Es tan importante como el nombre del producto, es usado para crear el identificador de paquete (o identificador de aplicación) conocido como "*Bundle Identifier*", presentado debajo del campo "*Company Identifier*"

Este identificador será una cadena utilizada para identificar al desarrollador de la aplicación, *Apple* exige a sus desarrolladores que utilicen como identificador el dominio de su empresa escrito a la inversa. Por ejemplo si la empresa se llama "*Chapps*" y dispone del dominio *chapps.com*, el identificador sería "*com.chapps*" seguido de un carácter punto (".") y el nombre de la aplicación. El *Bundle Identifier* resultante será "*com.chapps.EmptyApplication*", que será una cadena comprensible para el humano y única, para identificar la aplicación.

Si no se dispone de una compañía, se debe escribir el nombre del desarrollador, el identificador no necesariamente debe corresponder a un dominio existente. En la documentación de *Apple*, se indica que si no se dispone de un identificador de compañía, se puede usar la cadena "*edu.self"*. Si no se desea distribuir la aplicación desarrollada en la tienda *AppStore*, se puede dejar vacío este campo.

3. *Class Prefix:* *XCode* utiliza este prefijo de clase para nombrar a las clases que genera automáticamente; por ejemplo, la clase *AppDelegate*. Al escribir en este campo el prefijo *"EmptyApplication"*, la clase *AppDelegate* será nombrada automáticamente *EmptyApplicationAppDelegate*.[34]

4. *Device Family:* Se puede desarrollar una aplicación que funcione tanto en dispositivos *iPhone* como en dispositivos *iPad* sin tener que desarrollar versiones por separado. Estas aplicaciones son denominadas en *XCode* como aplicación de tipo "*Universal*". Es por esto que en este listado se debe escoger que tipo de aplicación se

34 Joshua, N. (2011). *Mastering XCode 4: Develop & Design.* Berkeley, California, Estados Unidos: Peachpit Press. p12, p13.

desarrollará, ya sea una aplicación solo para *iPhone (o iPod touch)*, solo para *iPad* o una aplicación *Universal*

Fig. 2.30: Listado *Device Family*

5. *Use Core Data*: Se debe marcar esta opción si se desea utilizar "*Core Data*", el *framework* de administración y persistencia de datos, que será analizado en el apartado *"2.7 Cómo realizar persistencia de datos"*. Se creará una aplicación vacía, por lo que no es necesario marcar esta opción.

6. *Use Automatic Reference Counting (ARC):* Se debe marcar esta opción si se desea utilizar *"ARC"*, que es una característica a nivel de compilador para simplificar el proceso de administración de memoria en las aplicaciones. Si no se tiene experiencia con *XCode*, es muy recomendado que se marque esta opción; caso contrario, si se desea asignar memoria para las variables manualmente no se debería marcar la misma. En la mayoría (o todos) los casos de este tutorial se utilizará esta característica ofrecida a partir de la versión 4.2 de *XCode*, por lo tanto se debe marcar.

7. *Include Unit Tests:* Se debe marcar esta opción si se desea incluir pruebas por unidad, que son una ventaja que incluye *XCode* y consiste en realizar pruebas a nivel de código fuente, es decir, permite por ejemplo, probar métodos con el objetivo de garantizar su correcto funcionamiento, incluso cuando se realicen cambios por corrección o mejora de rendimiento del método. Esto permite al desarrollador escribir un código fuente robusto y seguro, reduciendo la presencia de *bugs* en el mismo. Para crear un proyecto vacío no es necesario marcar esta opción[35].

Continuar presionando el botón: *"Next"*.

[35] Joshua, N. (2011). *Mastering XCode 4: Develop & Design.* Berkeley, California, Estados Unidos: Peachpit Press. p12, p13.

Se debe ahora seleccionar la ruta destino en donde se alojará el proyecto. Se presenta una ventana como la de la Fig. 2.31. Para el alojamiento de las aplicaciones de este tutorial se utilizará la carpeta denominada: "Aplicaciones". Si se marca la opción *create local git repository for this project*, (indicado con número 1 en la Fig. 2.31), se colocará el proyecto bajo control de versiones, pero no es necesario debido a que se está creando una aplicación vacía. Finalizar presionando el botón: *"Create"*.

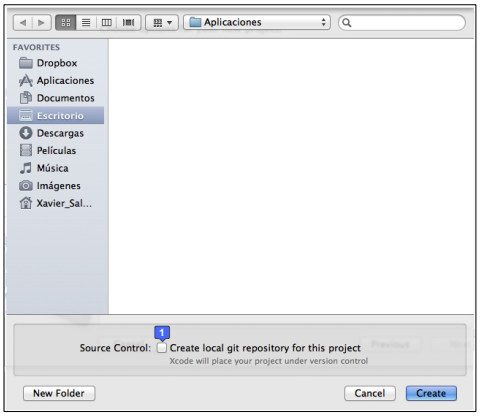


Fig. 2.31: Ruta de alojamiento para el proyecto

Finalmente se presentará la ventana del nuevo proyecto como se puede observar en la Fig. 2.32, en la cual, se han marcado los elementos que lo componen. En el siguiente apartado, se presenta un análisis de cada uno de estos elementos con el objetivo de conocer la importancia de los mismos.

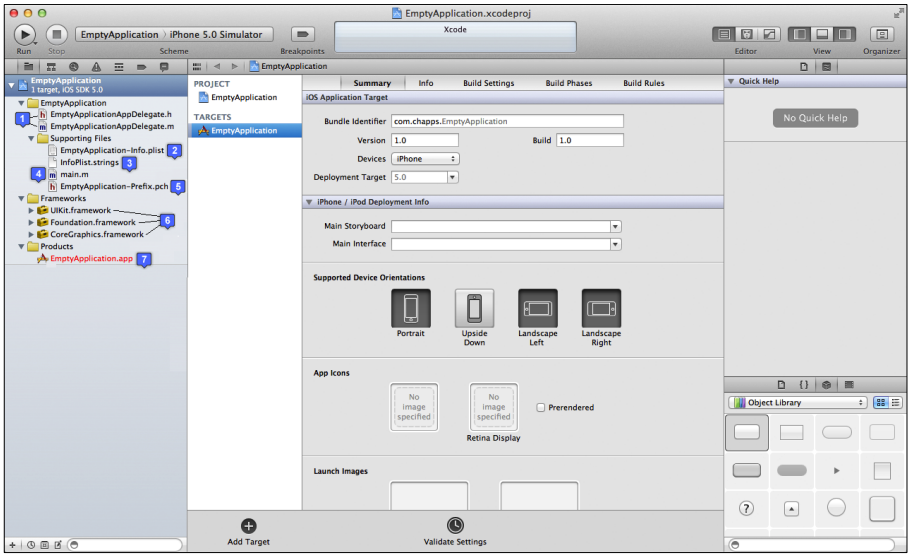


Fig. 2.32: Ventana de un nuevo proyecto recién creado

2.3 Análisis de tipos de posibles ficheros de un proyecto

Ahora que se ha creado un proyecto vacío, corresponde analizar cada uno de los ficheros y componentes que lo conforman. Para todos los proyectos, estos componentes se ubicarán en el área de navegación resaltado de color azul en la Fig. 1.2. En el proyecto vacío recién creado se encuentran los ficheros de la Fig. 2.32.

a) Ficheros autogenerados en el apartado anterior:

1. ***AppDelegate:*** *EmptyApplicationAppDelegate.h* / *EmptyApplicationAppDelegate.m:*

   El *AppDelegate* es como su nombre lo indica, el delegado de una aplicación. Al ser un objeto, es creado por una clase, es por esto que se compone de un fichero cabecera *EmptyApplicationAppDelegate.h* y uno de implementación *EmptyApplicationAppDelegate.m.*

   El *AppDelegate* es un objeto creado cuando la aplicación es iniciada en un dispositivo *iOS.* El principal objetivo de este objeto es el de manejar las transiciones de estado dentro de la aplicación. Por ejemplo, este objeto es responsable de la inicialización y manejo de transiciones desde y hacia el estado *background.* [36]

   Al analizar los archivos cabecera y de implementación, se observa que la clase *AppDelegate*, implementa los siguientes métodos principales.

   applicationDidFinishLaunchingWithOptions: Llamado cuando se ha lanzado la aplicación. Se puede implementar aquí, cualquier parámetro de inicialización adicional.

[36] Apple. (2012). *Core App Objects*. Recuperado el 22 de Agosto de 2012, de Core App Objects: http://developer.apple.com/library/ios/#DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/AppArchitecture/AppArchitecture.html

applicationWillResignActive: Llamado cuando la aplicación pasará de estado activo a inactivo. Esto puede ocurrir para ciertos tipos de interrupciones temporales como: una llamada entrante, un SMS, o cuando el usuario deja la aplicación y ésta pasa al estado *background.* Este método pudiera ser utilizado para pausar las tareas actuales de la aplicación, por ejemplo, en un juego, este método se debería utilizar para pausarlo.

applicationDidEnterBackground: Llamado cuando la aplicación entra en modo *background.* Se podría utilizar este método para grabar datos de usuario, invalidar *timers* en caso de existir, y almacenar la suficiente información sobre el estado de la aplicación en caso de que sea terminada después. Si la aplicación soporta ejecución en estado *background*, se llama a este método en lugar de *applicationWillTerminate:* cuando el usuario la finalice.

applicationWillEnterForeground: Llamado cuando la aplicación está a punto de entrar a estado *foreground* o activo; aquí se pueden deshacer muchos de los cambios realizados cuando la aplicación entró en estado *background.*

applicationDidBecomeActive: Es llamado después de que se activa la aplicación. Aquí se puede reiniciar cualquiera de las tareas que fueron pausadas mientras la aplicación estuvo inactiva. Si la aplicación estuvo previamente en estado *background,* se puede opcionalmente refrescar la interface de usuario.

applicationWillTerminate: Es llamado cuando la aplicación está a punto de terminar. Se deben grabar los datos de ser apropiado y se debe referir también al método *applicationDidEnterBackground:*.

2. ***Archivo de información y lista de propiedades (Info.plist):*** *EmptyApplication-Info.plist:*

Este archivo configura la aplicación. Todas las aplicaciones utilizan el mismo para almacenar información de configuración en un lugar en donde el sistema pueda accederlo fácilmente[37]. *iOS* utiliza este archivo para determinar el ícono, los tipos de documentos soportados, y muchos otros comportamientos que tienen impacto fuera de la aplicación en sí[38].

Un archivo de información y lista de propiedades (*Info.plist*) es un archivo estructurado de texto que contiene información esencial de configuración para una aplicación. El sistema utiliza llaves y valores contenidas en este archivo para obtener información sobre la aplicación y sus configuraciones.

Por convención, el nombre de un archivo de información y lista de propiedades es *"Info.plist"*, que tiene la letra inicial capital "I". Este archivo se encuentra en el nivel superior del directorio de la aplicación. (Fig. 2.33)

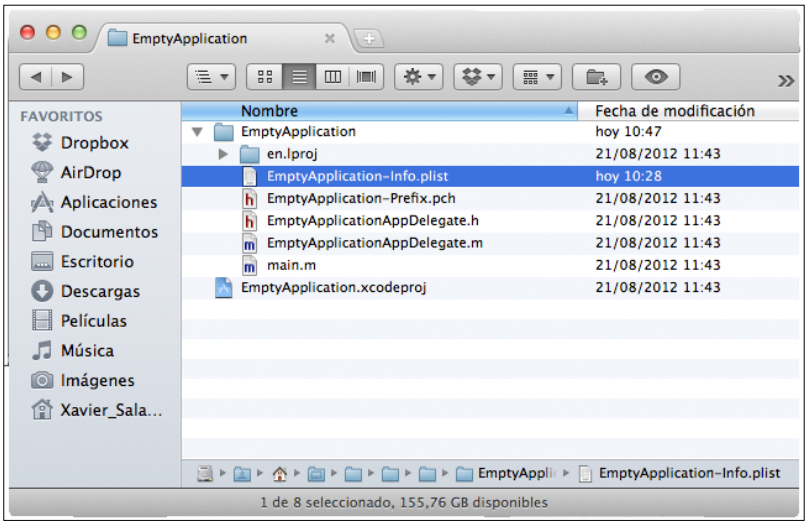


Fig. 2.33: Ubicación del archivo Info.plist

[37] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p28

[38] Apple. (2012). *About Info.plist keys*. Recuperado el 27 de Agosto de 2012, de About Info.plist keys: https://developer.apple.com/library/mac/#documentation/General/Reference/InfoPlistKeyReference/Introduction/Introduction.html

Se crea por defecto al crear cualquier nuevo proyecto y presenta una interface comprensible para el usuario (Fig. 2.34), es codificado en *Unicode UTF-8* y sus contenidos se estructuran usando *XML* (Fig. 2.35).

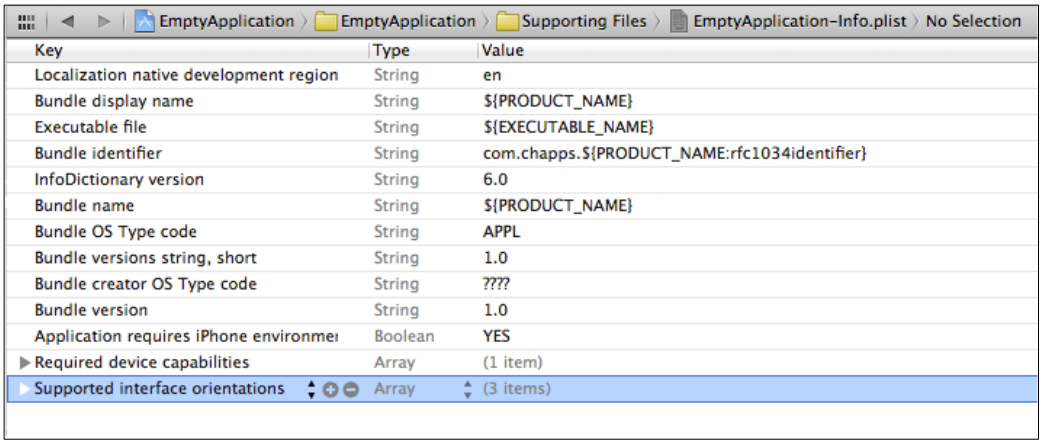


Fig. 2.34: Interface de un *Info.plist* en *XCode*

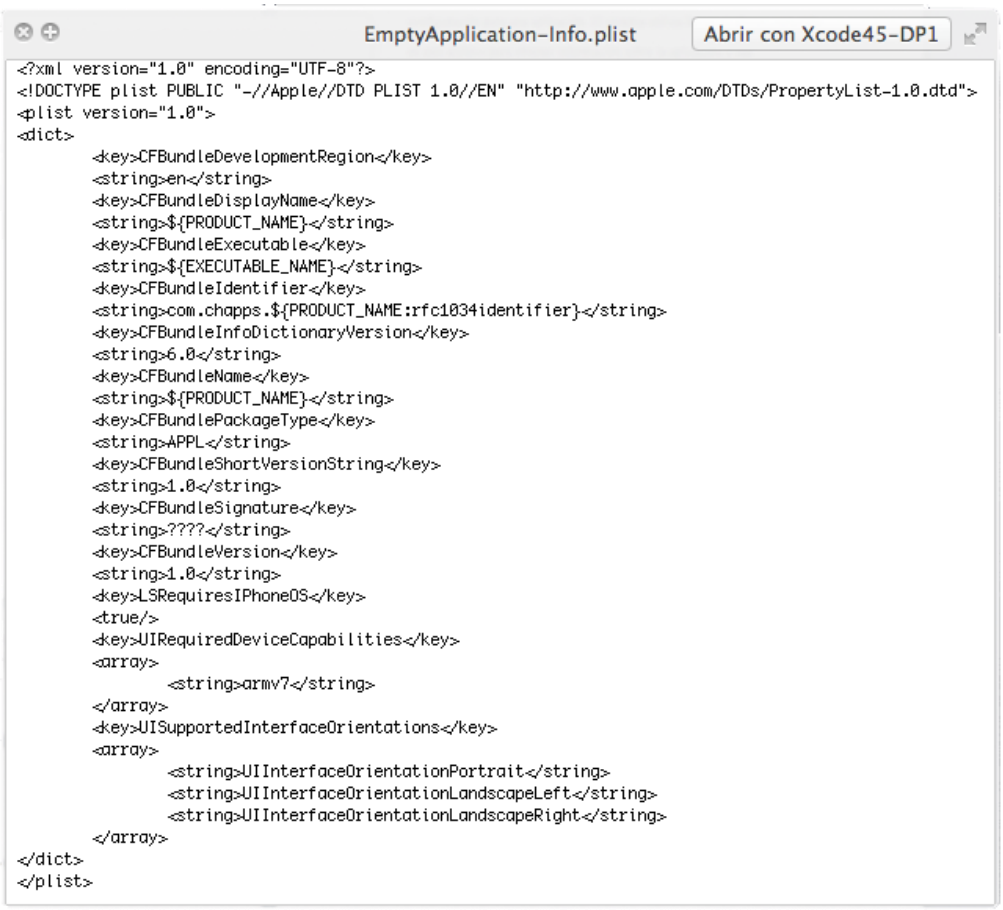


Fig. 2.35: Contenidos de un archivo *Info.plist*

Se puede editar cualquier valor por defecto creado por *XCode* en el archivo *Info.plist*. Para editar el valor de una llave (*Key*) específica, se debe presionar dos veces (doble click) sobre el valor en el editor del archivo de información y lista de propiedades de *XCode* (Fig. 2.34) y luego escribir el nuevo valor.

Se pueden agregar nuevos campos llave (*Key*) dentro del archivo, utilizando el editor del archivo de información y lista de propiedades de *XCode* (Fig. 2.34), para esto, se debe dar un *click* derecho sobre algún valor y elegir la opción *Add Row*, y, a partir de una lista, se puede seleccionar la llave que se desea adicionar junto con el valor a asignar. Por ejemplo se puede agregar la llave *"Icon File"*, y como valor escribir el nombre del ícono que se desea que la aplicación tenga. Se conocerá como agregar el archivo ícono en el apartado *"2.3.2 Asignar un ícono para la aplicación"*. De igual manera se pueden remover las filas agregadas.

Se pueden agregar llaves que afecten a dispositivos específicos, por ejemplo, se puede definir que la orientación de la aplicación varíe cuando se ejecute en un dispositivo *iPad*, definiendo la llave correspondiente de la siguiente forma, ya sea en un editor de *XML* o en el editor de *XCode*:

```
<key>UIInterfaceOrientation</key>
<string>UIInterfaceOrientationPortrait</string>
<key>UIInterfaceOrientation~ipad</key>
<string>UIInterfaceOrientationLandscapeRight</string>
```

Fig. 2.36: Configuración de orientación Landscape en iPad y orientación Portrait en otros dispositivos iOS mediante el archivo Info.plist[39]

Se recomienda no eliminar las llaves generadas automáticamente por *XCode* al crear un proyecto para que funcione correctamente.

[39] Apple. (2012). *About information Property List Files*. Recuperado el 27 de Agosto de 2012, de About information Property List Files: https://developer.apple.com/library/mac/#documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html#//apple_ref/doc/uid/TP40009254-SW1

3. *InfoPlist.strings*

Este archivo de cadenas (*strings*) sirve para almacenar *"valores localizados"*. Los valores localizados son valores que cambian en la aplicación de acuerdo a la *"localización"* del dispositivo. Estos valores son descritos de esta manera, pero en realidad no depende de la localización, sino del idioma que se utiliza en el dispositivo. [40]

El idioma se ajusta siguiendo la ruta Ajustes → General → Internacional → Idioma, en el dispositivo *iOS*. Algunas aplicaciones varían incluso su nombre al cambiar el idioma del dispositivo; esto se consigue utilizando este archivo de cadenas. Para esto se debe crear un directorio dentro de la aplicación, de acuerdo al idioma. Por defecto se ha creado la carpeta *"en.lproj"* en donde se almacenará el archivo de cadenas *"InfoPlist.strings"* para cuando la aplicación se ejecute en un dispositivo *iOS* que este configurado para funcionar en idioma inglés. Si se desea que la aplicación tenga una traducción al español, se deberá crear la carpeta *"es.lproj"* y dentro de la misma colocar el archivo de cadenas y todos los archivos multimedia en español que se necesiten visualizar en la aplicación. Se debe repetir el proceso para cualquier otro idioma que se quisiera adicionar.

Los contenidos del archivo *InfoPlist.strings* son las llaves individuales que se deseen localizar y su valor apropiado traducido. Las rutinas que buscan valores de llaves en el archivo *InfoPlist.strings* toman el lenguaje desde las preferencias del dispositivo y devuelven el valor de la versión localizada de la llave en caso de existir. Si no existiese este archivo, las rutinas devuelven el valor alojado en el archivo *Info.plist.*

[40] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p29

Se presenta un ejemplo en la documentación de *Apple*, para la aplicación de *Mac*: *TextEdit*. Los archivos *InfoPlist.strings* varían de contenido de acuerdo a su localización:

```
<key>CFBundleDisplayName</key>
<string>TextEdit</string>
<key>NSHumanReadableCopyright</key>
<string>Copyright ¬© 1995-2009, Apple Inc.,All Rights Reserved.
</string>
```

Fig. 2.37: *InfoPlist.strings* dentro del directorio *en.lproj*[41]

```
CFBundleDisplayName = "TextEdit";
NSHumanReadableCopyright = "Copyright © 1995-2009 Apple
Inc.\nTous droits réservés.";
```

Fig. 2.38: InfoPlist.strings dentro del directorio French.lproj[42]

Como se puede observar, se necesita de un archivo *InfoPlist.strings* para cada traducción que se desee realizar a la aplicación.

4. ***Main.m***

Este archivo de implementación es generado automáticamente por *XCode* al crear un proyecto; contiene el método *main()*. En este método se crea un objeto de aplicación, el cual, entre otras acciones, establece un ciclo de ejecución (un ciclo de ejecución registra fuentes de entrada y habilita la entrega de eventos entrantes a la aplicación). La mayoría de este trabajo lo realiza la función *UIApplicationMain*, incluida en el *framework UIKit* y llamada automáticamente en este archivo.

La sentencia *@autoreleasepool* soporta el sistema *Automatic Reference Counting ARC*, que provee una administración de tiempo de vida automático

[41] Apple. (2012). *About information Property List Files*. Recuperado el 27 de Agosto de 2012, de About information Property List Files: https://developer.apple.com/library/mac/#documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html#//apple_ref/doc/uid/TP40009254-SW1

[42] Apple. (2012). *About information Property List Files*. Recuperado el 27 de Agosto de 2012, de About information Property List Files.

para los objetos de la aplicación, asegurando que los mismos existan únicamente mientras son necesarios[43].

A continuación se presenta el contenido del archivo *main.m*, se lo puede seleccionar en la ventana de navegación para visualizarlo. Ver Fig. 2.39.

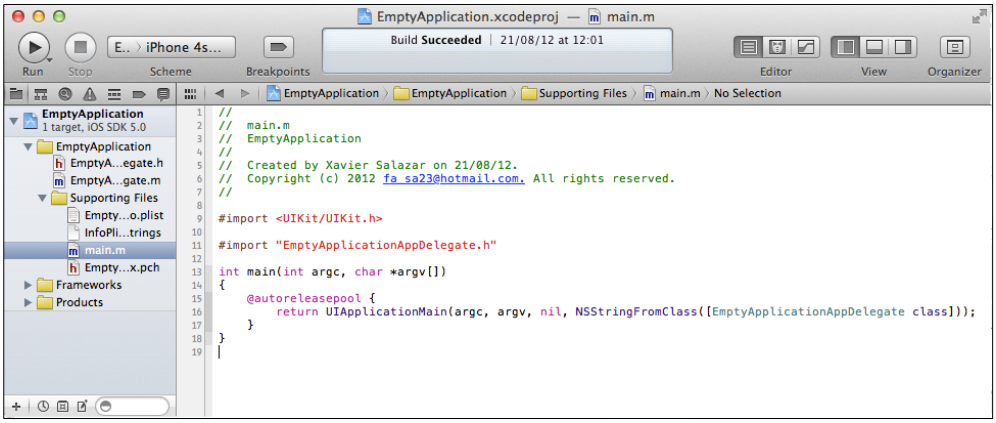


Fig. 2.39: Contenido del archivo main.m

La llamada a *UIApplicationMain*, revisa el archivo *Info.plist* de la aplicación, crea una instancia de la clase *UIApplication* y una instancia del *AppDelegate*, cuya principal tarea es la de proveer el objeto ventana en la cual se dibujará el contenido de la aplicación.

5. *EmptyApplication-Prefix.pch*

Este archivo es una lista de ficheros cabecera (o *header*) de *frameworks* externos que son usados por el proyecto. La extensión **.pch* es por *precompiled header*. Las cabeceras referenciadas en este archivo, no forman parte del proyecto y no suelen cambiar. *XCode* pre compila estas cabeceras y luego continua utilizando esas versiones pre compiladas para construcciones (*builds*) futuras, lo que reduce significativamente el tiempo de compilación del proyecto. [44]

[43] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p29
[44] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p29

Generalmente este archivo no requiere mayor atención, debido a que la mayoría de ficheros cabecera empleados, suelen encontrarse ya incluidos.

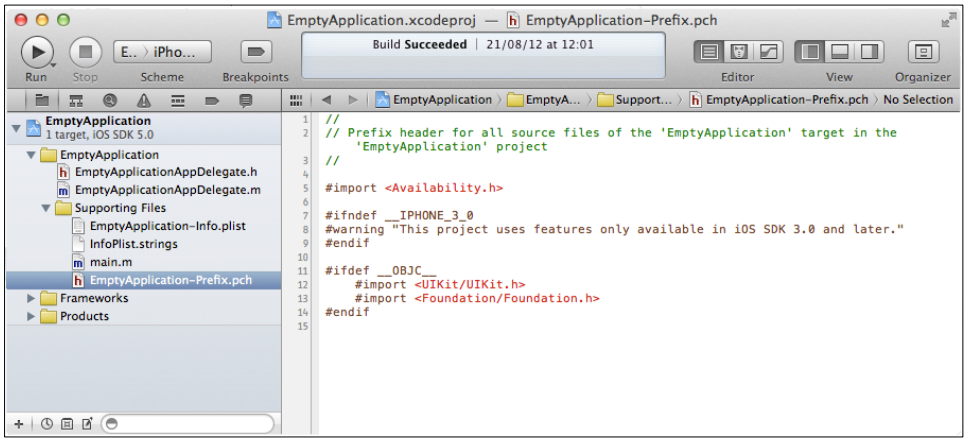


Fig. 2.40: Contenido del archivo Prefix.pch

6. ***Frameworks:***

Los frameworks de *Apple* son librerías pre escritas que pueden ser importadas en un proyecto para agregar funcionalidades adicionales, como soporte de video, sonido o persistencia de datos[45].

Múltiples aplicaciones pueden usar todos estos recursos simultáneamente. El sistema los carga en memoria a medida que los necesita y comparte la copia del recurso entre todas las aplicaciones cuando es posible.

Los *frameworks* proveen una librería de rutinas que pueden ser llamadas por una aplicación para realizar una tarea específica. Por ejemplo, los frameworks *Foundation* y *Application Kit* proveen las interfaces programáticas para las clases y métodos. Los frameworks ofrecen estas ventajas:

- Agrupan recursos separados, pero relacionados, en conjunto. Este agrupamiento facilita la instalación, desinstalación y localización de estos recursos.

[45] Wenkt, R. *XCode 4: Developer Reference.* Indianapolis, Indiana: Wiley Publishing Inc. p78

- Incluyen una amplia variedad de recursos, archivos cabecera (*headers*) y documentación.
- Solo una copia de recursos solo-lectura de un *framework* residen físicamente en memoria en cualquier momento dado, sin importar cuantos procesos estén usando esos recursos. Al compartir recursos, se reduce el consumo de memoria del sistema y ayuda a mejorar su rendimiento.

Además de usar los *frameworks* del sistema, se pueden crear *frameworks* personalizados y usarlos de manera privada para las aplicaciones que se desarrollen; o publicarlos para otros desarrolladores que pudieran reutilizarlos[46].

A continuación se detallan los *frameworks* que *XCode* ha incluido automáticamente al crear un proyecto vacío.

UIKit.framework: Provee las clases necesarias para construir y manejar interfaces de usuario de aplicaciones *iOS.* Incluye administración de eventos, vistas y otros controles diseñados específicamente para una interface de usuario táctil (*touch screen*)[47]. Muchas de las clases incluidas en este *framework* se utilizarán en el presente tutorial.

Foundation.framework: Define una capa base de las clases de *Objective-C*. Adicionalmente provee un conjunto de clases de objetos primitivos. Este *framework* fue diseñado con los siguientes objetivos:

- Proveer un set pequeño de clases de utilidades básicas.
- Soportar cadenas *Unicode*, persistencia y distribución de objetos.

---

[46] Apple. (2012). *What are Frameworks?* Recuperado el 28 de Agosto de 2012, de What are Frameworks?: https://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/BPFrameworks/Concepts/WhatAreFrameworks.html#//apple_ref/doc/uid/20002303-BBCEIJFI

[47] Apple. (2012). *UIKit Framework Reference*. Recuperado el 28 de Agosto de 2012, de UIKit Framework Reference: http://developer.apple.com/library/ios/#documentation/uikit/reference/UIKit_Framework/Introduction/Introduction.html#//apple_ref/doc/uid/TP40006955-CH1-SW2

- Proveer un nivel de independencia de sistema operativo, para fortalecer la portabilidad.

El *framework* incluye: clases que representan los tipos de dato básicos como cadenas (*strings*) y arreglos de *bytes* (*byte arrays*), colecciones de clases para almacenamiento de otros objetos, clases que representan información del sistema como fechas, y clases que representan puertos de comunicación[48].

Muchas de las clases incluidas en este *framework* se utilizarán en el presente tutorial.

CoreGraphics.framework: Es una *API* que sirve para administrar el trazado basado en caminos, rutas o pixeles; transformaciones, manejo de colores, patrones, gradientes y sombreados, administración de datos de imágenes, creación de imágenes, enmascarado, creación de documentos *PDF*, presentación, y análisis.[49]

7. ***Productos:*** *EmptyApplication.app*

Esta carpeta contiene la aplicación construida por el proyecto. *EmplyApplication.app* es la única aplicación que creará el proyecto vacío realizado en el apartado anterior "*2.2 Cómo crear un nuevo proyecto*". Si el nombre de la aplicación se encuentra en color rojo, significa que *XCode* está

[48] Apple. (2012). *Foundation Framework Reference*. Recuperado el 28 de Agosto de 2012, de Foundation Framework Reference: https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/ObjC_classic/Intro/IntroFoundation.html#//apple_ref/doc/uid/20000687-BAJDAJAG

[49] Apple. (2012). *Core Graphics Framework Reference*. Recuperado el 28 de Agosto de 2012, de Core Graphics Framework Reference: http://developer.apple.com/library/ios/#DOCUMENTATION/CoreGraphics/Reference/CoreGraphics_Framework/_index.html

indicando que alguno de los archivos referencian a algo que no se encuentra.[50] Esto es generalmente debido a que aún no se ha compilado o construido la aplicación.

[50] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p29

b) Ficheros adicionales que se pueden agregar a un proyecto:

Existen otros tipos de archivos que se pueden agregar a un proyecto, entre los cuales, se especificarán los mas relevantes y los que se pretende utilizar en el tutorial.

1. ***UIViewController subclass:***

Se encuentra en la categoría de plantillas para *iOS*, en la subcategoría *Cocoa Touch* (*framework* para desarrollo de aplicaciones para *iOS*). Es una subclase de *UIViewController*, cuyo fichero cabecera incluye la cabecera de la clase *UIKit*. Al agregarla a un proyecto, opcionalmente se puede seleccionar si se desea un archivo de interface de usuario *XIB*.

Una subclase *UIViewController* es una clase, que como su nombre lo indica, controla una vista en una aplicación. Se necesita una clase *UIViewController* para cada vista que se implemente en el proyecto, debido a que en esta clase se almacenarán las variables y se gestionarán los eventos que pudieran suscitarse dentro de la vista.

Una vista se encuentra contenida dentro de una ventana que es transparente para el usuario. La vista es lo que el usuario puede ver en cualquier momento en una aplicación, por ejemplo la Fig. 2.41.

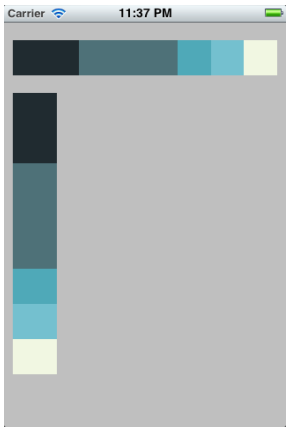


Fig. 2.41: *UIView*

Se explicará detalladamente esta clase en los apartados *"2.3.1 Como agregar un fichero a un proyecto"* y *"2.5 Asignación de variables y eventos a componentes en View Controllers y Table View Controllers"*.

2. ***Objective-C class***

   Se encuentra en la categoría de plantillas para *iOS*, en la subcategoría *Cocoa Touch*. Es una clase que se desarrolla en lenguaje *Objective-C* e incluye un archivo de implementación y un archivo cabecera.

   Si se desea trabajar con un archivo de este tipo, se puede indicar si la clase, es una subclase de otra, antes de añadirla al proyecto.

   Cuando se diseña una base de datos para administrar información dentro de una aplicación, el *IDE XCode* presenta la capacidad de autogeneramiento de clases especificadas dentro del diagrama de base de datos.

   Se explicará mas sobre este archivo en el apartado: *"2.7 Cómo realizar persistencia de datos"* y sus sub-apartados.

3. ***Storyboard***

   Se encuentra en la categoría de plantillas para *iOS*, dentro de la subcategoría *User Interface*. Es una representación visual de la interface de usuario de una aplicación *iOS*, que presenta ventanas de contenido y las conexiones entre las mismas. Un *Storyboard* se compone de una secuencia de escenas, cada una de las cuales representa un *View Controller* y sus *Views*. Las escenas son conectadas por objetos *Segue* que representan una transición.

   El *IDE XCode* provee un editor visual para *Storyboards*, en donde se puede diseñar la interface de usuario de la aplicación agregando elementos como

botones, tablas y cajas de texto a las escenas. Adicionalmente, el *Storyboard* permite conectar una vista a su objeto controlador y administrar la transferencia de información entre *View Controllers.* Se recomienda el uso de *Storyboards* para diseñar interfaces de usuario de aplicaciones *iOS* debido a que permiten visualizar la apariencia y flujo que tendrá la interface de usuario diseñada sobre un *canvas* o lienzo[51].

Se explicará como agregar un *Storyboard* al proyecto en el apartado *"2.3.1 Cómo agregar un fichero a un proyecto"*, en el sub apartado *"a) Agregar un Storyboard"*.

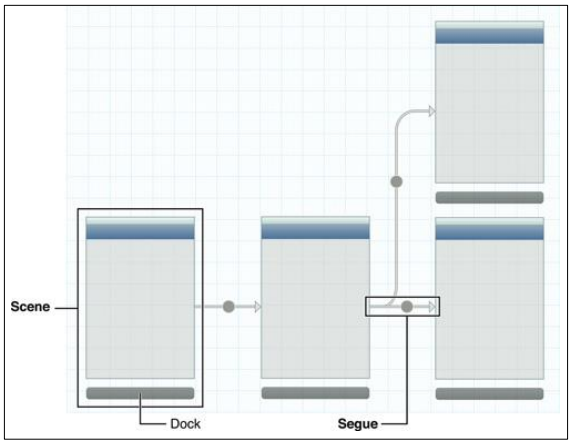


Fig. 2.42: Elementos de un Storyboard

4. ***Data Model***

Se encuentra en la categoría de plantillas para *iOS*, dentro de la subcategoría *Core Data*. Es un archivo de modelado de datos que permite utilizar el componente de diseño de *XCode.* Se explicará como agregar este archivo en un proyecto y como trabajar con el mismo en el apartado: *"2.7 Cómo realizar persistencia de datos"* y sus sub-apartados.

[51] Apple. (2012). *Storyboard*. Recuperado el 12 de Septiembre de 2012, de Storyboard: http://developer.apple.com/library/mac/#documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

2.3.1 Cómo agregar un fichero a un proyecto

Ahora que se conocen los principales ficheros que pueden formar parte de un proyecto, corresponde explicar como agregarlos. Para esto se debe crear un nuevo proyecto vacío y nombrarlo de cualquier forma. En el tutorial será nombrado "Ejemplo1". Si se requiere de ayuda para hacerlo, se puede referir al apartado "*2.2 Cómo crear un nuevo proyecto*".

a) Agregar un *Storyboard*

Para agregar un *Storyboard*, se debe abrir el proyecto "Ejemplo1" y dirigirse a File → New → File como se muestra en la Fig. 2.43.

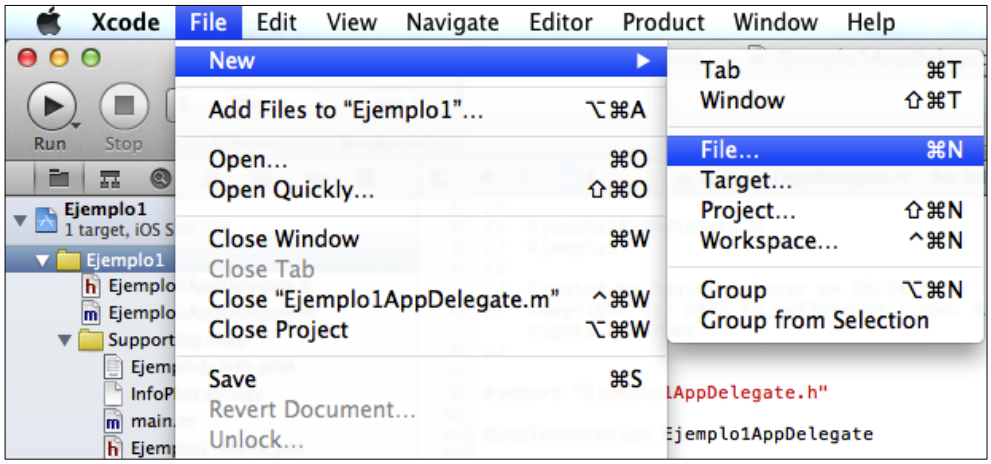


Fig. 2.43: Forma de agregar un nuevo archivo

En la ventana que se presenta a continuación, se debe indicar que se desea agregar un archivo *Storyboard* así: Se debe seleccionar la categoría "*User Interface"*, después *Storyboard* y finalizar presionando el botón "*Next"*.

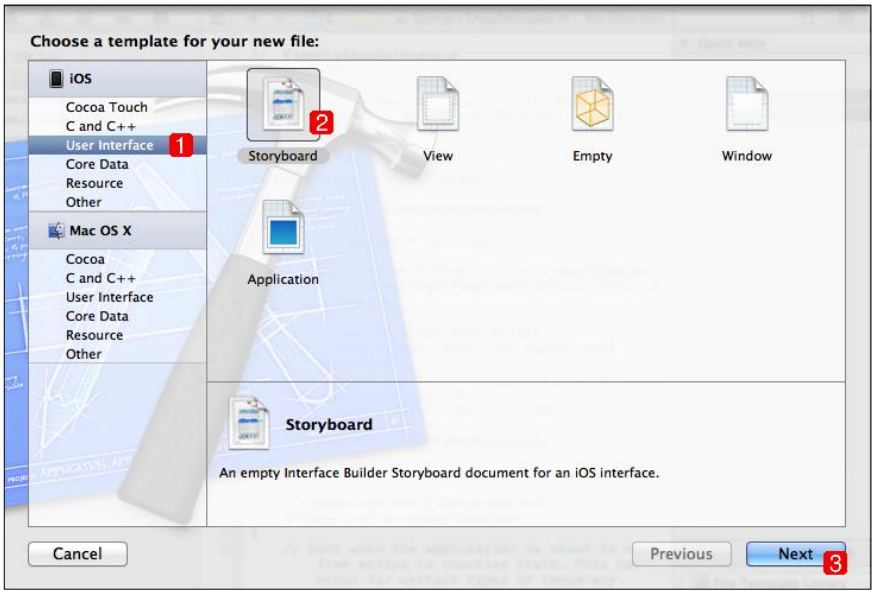


Fig. 2.44: Selección de archivo Storyboard

A continuación se pregunta si se desea que el *Storyboard* sea para *iPhone* o *iPad*. Si se desea desarrollar una aplicación universal, se deberían agregar dos *Storyboards*, uno para cuando la aplicación se ejecute en dispositivos *iPhone* y uno para cuando la aplicación se ejecute en dispositivos *iPad*.

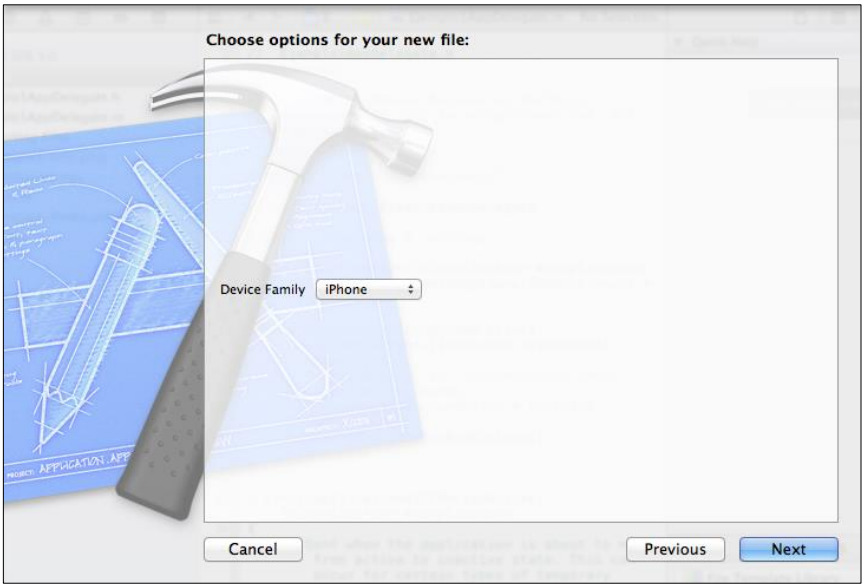


Fig. 2.45: *Device Family* en el que funcionará el *Storyboard*

Seleccionar *iPhone* en "*Device Family*" y presionar el botón *"Next"*

A continuación se debe nombrar el *Storyboard*; se lo hará como *Ejemplo1iPhoneStoryboard.storyboard*. Finalizar presionando el botón "Create".

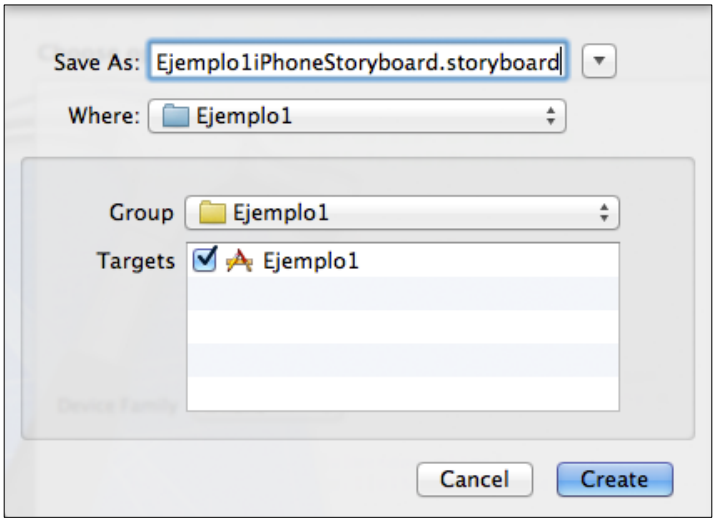


Fig. 2.46: Nombrar el *Storyboard*

Para verificar que se ha agregado exitosamente el *Storyboard*, se debe cerciorar de que éste se encuentre en el Área de Navegación como se muestra en la Fig. 2.47.

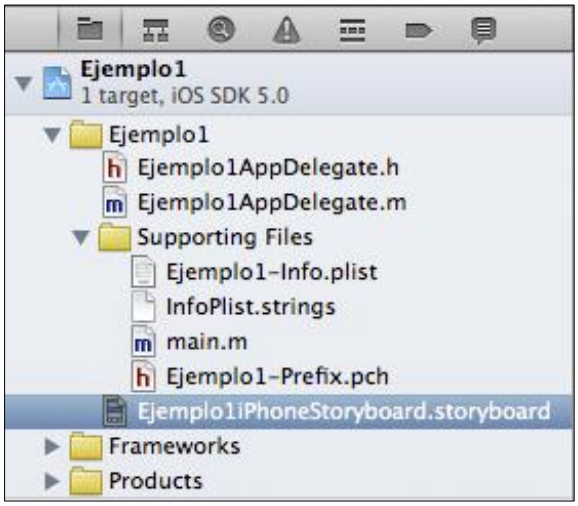


Fig. 2.47: *Storyboard* agregado con éxito

El último paso consiste en asignar el *Storyboard* creado a la aplicación; para esto: Se debe seleccionar el proyecto *Ejemplo1* en el Área de Navegación, acto seguido, seleccionar *Summary* y finalmente en el apartado *Main Storyboard*, se debe seleccionar el *Storyboard* recién agregado, tal como se muestra en la Fig. 2.48.

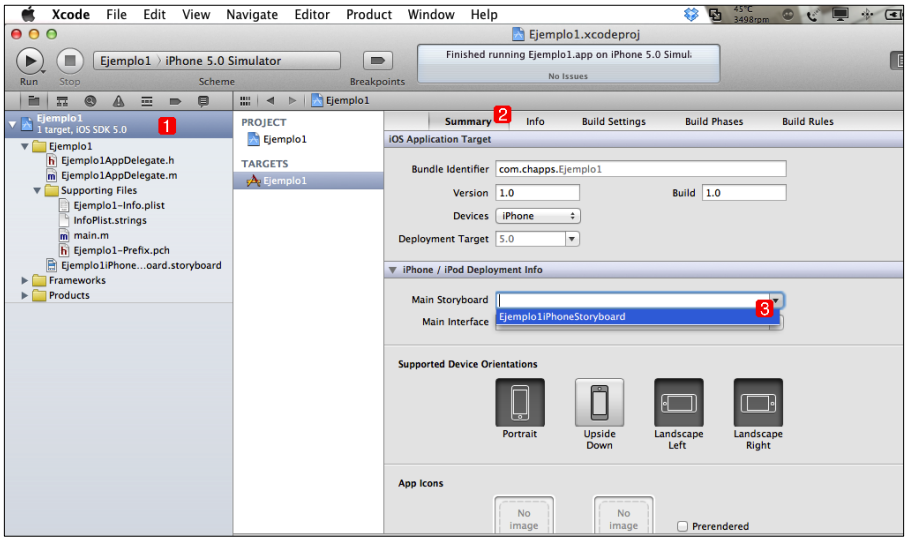


Fig. 2.48: Asignación de un *Storyboard* al proyecto

En este *Storyboard* se puede diseñar la interface gráfica de usuario de la aplicación, compuesta por un conjunto de vistas o ventanas enlazadas entre sí. Cada vista necesita una clase controladora, por lo tanto el siguiente archivo que se agregará al proyecto, será la clase controladora de una vista: *UIViewController*.

b) Agregar una clase *UIViewController*

Para agregar una clase *UIViewController*, se debe dirigir a File → New → File, como se indica en la Fig. 2.43. En la ventana que se presenta a continuación se debe indicar que se desea agregar una clase *UIViewController* así: Se debe

seleccionar la categoría "*Cocoa Touch*", seleccionar después *Objective-C class* y finalizar presionando el botón "Next".

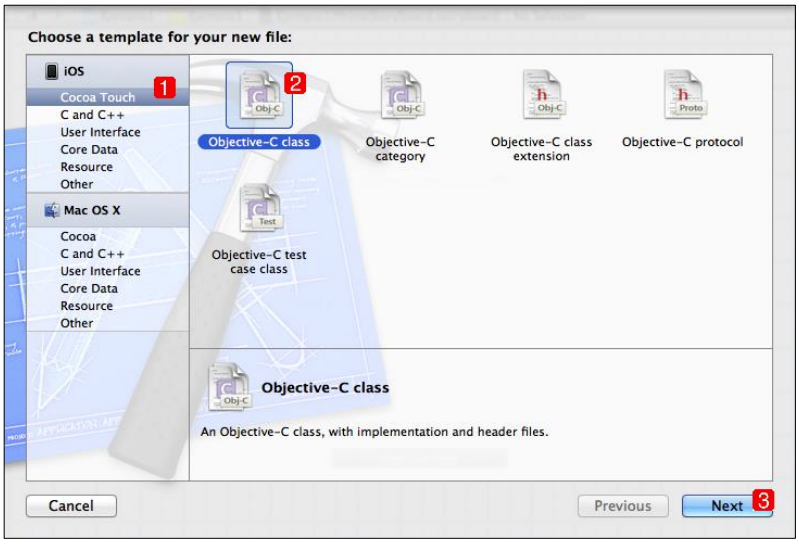


Fig. 2.49: Selección de un archivo Objective-C class

En la siguiente ventana, se debe escribir un nombre para identificar a la clase *UIViewController* que se agregará. Una aplicación generalmente está formada por varias vistas y, por lo tanto, varias de estas clases controladoras, así que debe ser nombrada de manera de que pueda ser fácilmente identificada; por ejemplo *VistaDeConfiguracionesViewController*. Para ejemplificar, se la nombrará *Ejemplo1ViewController*.

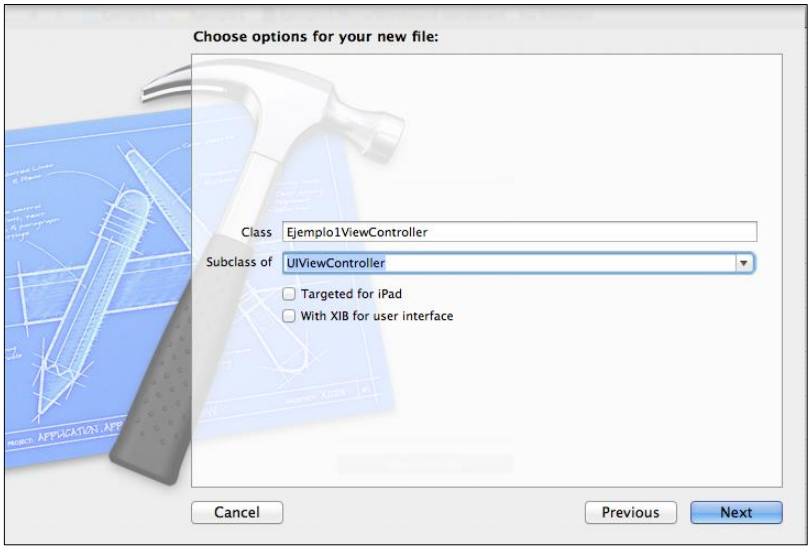


Fig. 2.50: Nombre de la clase y subclase a la que pertenece

A continuación se debe seleccionar de la lista "*Subclass of*" la clase *UIViewController*. Si se desea crear un controlador de una vista específico para dispositivos *iPad*, se debe marcar la opción "*Targeted for iPad"* antes de continuar. De igual manera si se desea crear el archivo *XIB* para la interface de usuario, se debe marcar la opción "*With XIB for user interface"*. Por ahora no es necesario marcar ninguna de estas opciones. Continuar presionando el botón *"Next"*.

Finalmente se presentará la ventana de la Fig. 2.51 en la cual se puede cambiar la ruta en donde se alojará el *UIViewController*. Se recomienda que no se cambie esta ruta debido a que de esta forma el archivo *UIViewController* creado, se almacenará dentro de la misma carpeta del proyecto sobre el cual se trabaja; en este caso "Ejemplo1". Finalizar presionando el botón *"Create"*.

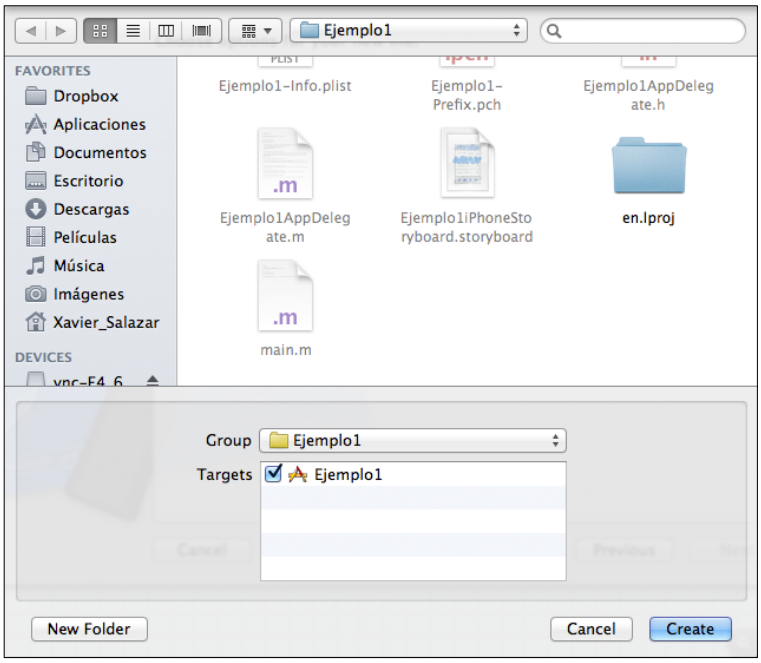


Fig. 2.51: Selección de la ruta de almacenamiento del *UIViewController*

Para verificar la correcta creación de la clase controladora *UIViewController*, se debe revisar que en el Área de Navegación del proyecto ahora se pueda encontrar tanto el archivo cabecera como el archivo de implementación de la misma, como se indica en la Fig. 2.52.

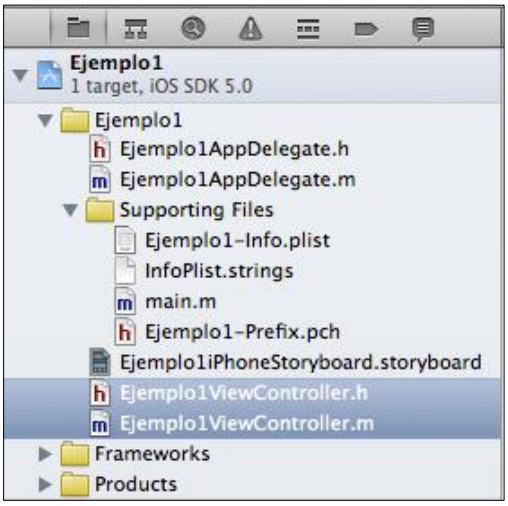


Fig. 2.52: *UIViewController* agregado con éxito

Finalmente se debe asignar el *UIViewController* a una ventana o vista de la interface gráfica de usuario de la aplicación para poder controlarla. Para esto, se debe referir al apartado *"2.4.2 Construcción de una interface gráfica de usuario"*. Específicamente al "Paso 6" del mismo.

c) Agregar un archivo de imagen

Una aplicación se caracteriza tanto por su facilidad de uso, como por la calidad de su contenido visual, es por esto que muchas de las aplicaciones presentan imágenes llamativas para atraer la atención del usuario. A continuación se explicará como agregar un archivo de imagen, que posteriormente podrá ser agregado a una vista o utilizado como ícono de la aplicación.

Para agregar un archivo de imagen, se debe arrastrar el mismo y soltarlo sobre el Área de Navegación del proyecto.

Fig. 2.53: Arrastrar y soltar una imagen sobre el Área de Navegación del proyecto

Al hacerlo, se presenta la ventana de la Fig. 2.54, en la cual, se puede configurar las opciones de insertado de la imagen en el proyecto.

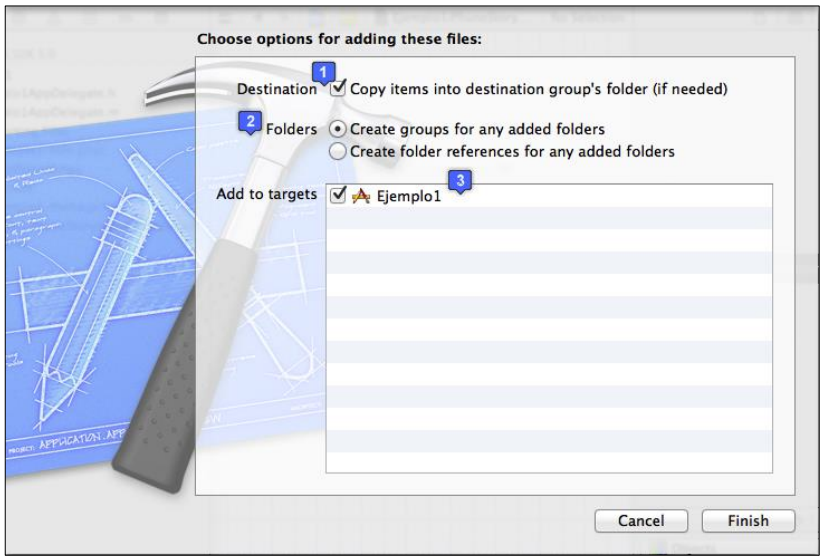


Fig. 2.54: Opciones de agregado de imágenes en un proyecto

1. ***Destination:*** Se debe marcar esta opción para que la imagen agregada al proyecto, se copie en la carpeta del mismo. Si no se lo hace, la imagen se agregará al proyecto pero no a su carpeta contenedora, estableciéndose únicamente una referencia a la ubicación de la imagen, lo cual no es recomendado.
2. ***Folders***: Esta opción es útil cuando se agregan carpetas de archivos a un proyecto, al agregar una imagen, se debe dejar marcada la opción por defecto

"*Create groups for any added folders*". A continuación se explica cada una de las posibilidades.

a. ***Create groups:*** Al seleccionar esta opción, se copia la carpeta de archivos dentro de la carpeta del proyecto.

b. ***Create folder references:*** Al seleccionar esta opción, únicamente se establece una referencia hacia la carpeta de archivos que se desea agregar al proyecto.

3. ***Add to targets:*** Si se trabaja con mas de un proyecto a la vez, se puede agregar los archivos a todos los proyectos deseados simultáneamente. Se debe marcar cada proyecto en el cual se requiera agregar los archivos, en este caso "Ejemplo1". Continuar presionando el botón "Finish".

Se puede verificar que el archivo fue agregado exitosamente en el proyecto como se indica en la Fig. 2.55.

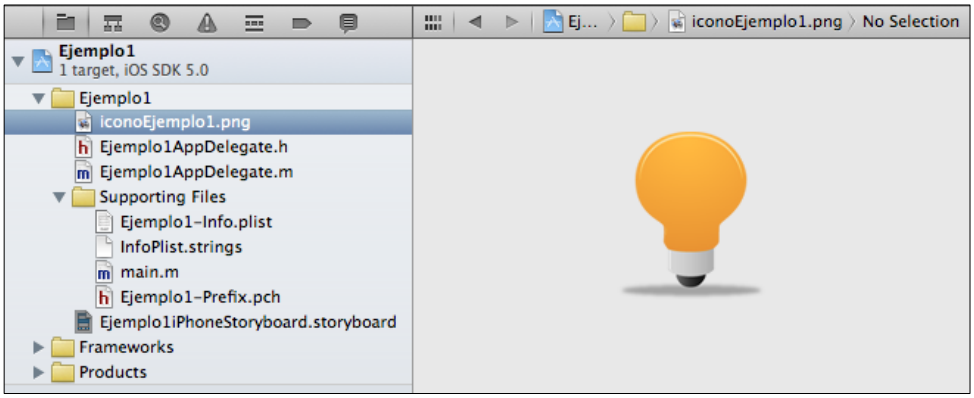


Fig. 2.55: Archivo de imagen agregado con éxito

Para asignar una imagen como ícono de la aplicación se puede referir al apartado *"2.3.2 Asignar un ícono para la aplicación"*.

2.3.2 Asignar un ícono para la aplicación

El ícono es la presentación de una aplicación. Se recomienda utilizar un ícono llamativo y que tenga que ver con la funcionalidad de la aplicación a desarrollar, así como lo hace la empresa *Apple Inc.* con sus aplicaciones incluidas en los dispositivos *iOS* como puede observar en la Fig 2.56.

Fig. 2.56: Íconos de las aplicaciones de *Apple Inc.*

A partir del *iPhone4* e *iPad3,* los dispositivos *iOS,* presentan una pantalla de retina, cuya resolución es muy superior con respecto a dispositivos predecesores. Es por esto que la aplicación debería tener dos íconos iguales pero de distintas dimensiones: un ícono con una resolución de 114x114 pixeles para pantallas de retina y otro con resolución de 57x57 pixeles para pantallas de dispositivos anteriores.

En el apartado anterior se añadió al proyecto "*Ejemplo1*" una imagen de 114x114 pixeles, ahora se debe agregar la misma pero con dimensiones de 57x57 pixeles. Si se requiere ayuda se debe referir al apartado "*2.3.1 Cómo agregar un fichero a un proyecto*" específicamente al sub apartado "*C) Agregar un archivo de imagen*".

Se debe verificar que se dispone de los dos archivos de imagen agregados en el proyecto como se indica en la Fig. 2.57.

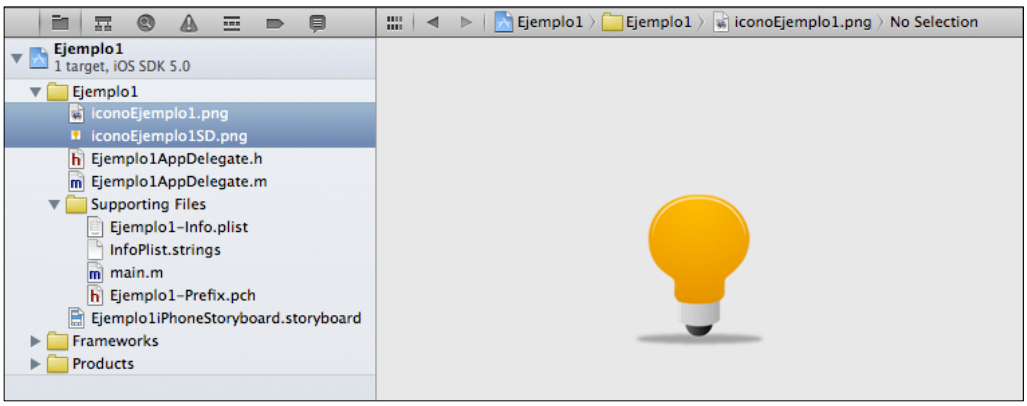


Fig. 2.57: Archivos de imagen para ícono agregados al proyecto

En el Área de Navegación, se debe seleccionar el archivo de propiedades del proyecto "*Info.plist*" para añadir una nueva propiedad sobre el mismo. En cualquier área en blanco de este archivo, se debe dar un *click* derecho para poder agregar una nueva propiedad o fila "*Add Row*" en el mismo, tal como se muestra en la Fig. 2.58.

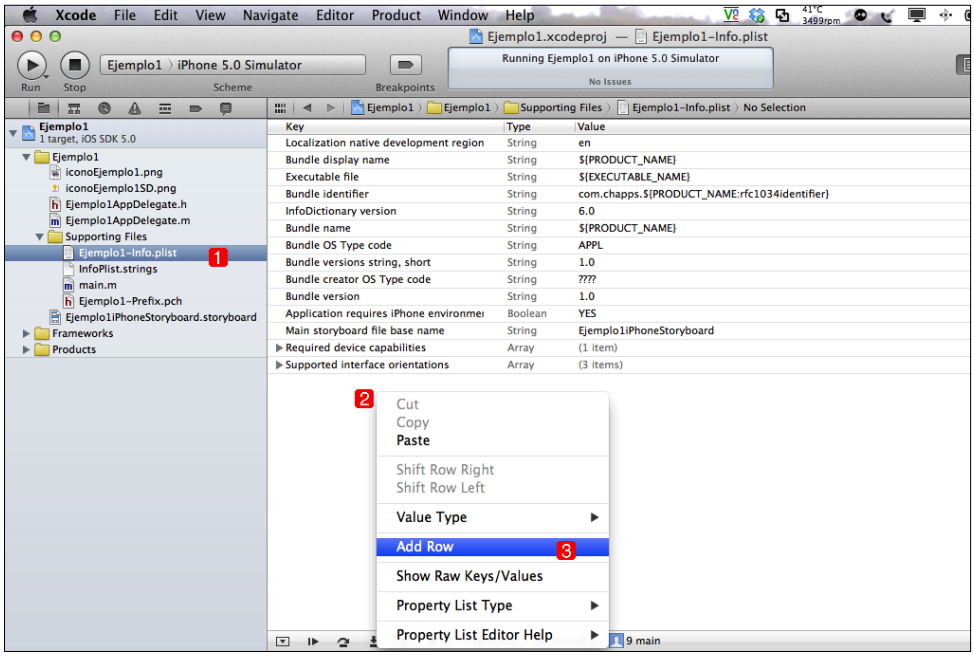


Fig. 2.58: Insertar una propiedad o fila en el archivo de propiedades del proyecto

A continuación se despliega una lista de propiedades que se pueden agregar sobre este archivo en la cual se debe seleccionar el arreglo "*Icon files*" como se muestra en la Fig. 2.59.

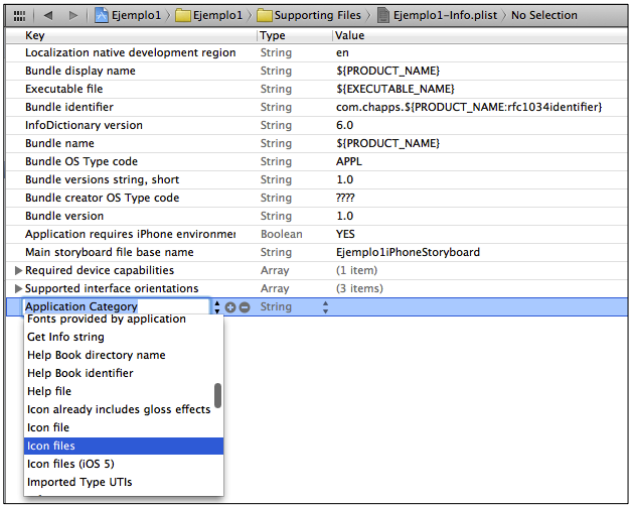


Fig. 2.59: Agregar propiedad *Icon files*

Esta propiedad o arreglo, permite agregar varios íconos, aquí se agregará el ícono estándar y el ícono en alta definición de la aplicación.

Para esto se debe presionar 

para poder ingresar cada ícono en un ítem del arreglo. En el item0 se debe escribir el nombre incluyendo la extensión del archivo imagen agregado, acto seguido se debe realizar el mismo procedimiento para el item1 en donde se ingresará el nombre con la extensión del otro ícono; como se indica en la Fig. 2.60.

Fig. 2.60: Agregado de iconos en el archivo de propiedades

Si se presiona en este momento el botón "*Run*" 

para compilar y ejecutar el proyecto, se ejecutará la aplicación en el simulador *iOS*, presentando una ventana completamente blanca o negra dependiendo de la versión de *XCode* que se utilice, esto

se debe a que la aplicación aún no tiene una vista o ventana inicial de interface gráfica de usuario para mostrar. Se debe presionar el botón home del *iPhone* en el simulador para poder visualizar los íconos de las aplicaciones instaladas. Se podrá ver el ícono asignado para la aplicación como se muestra en la Fig. 2.61. Si se revisan las propiedades del proyecto como se muestra en la Fig. 2.62, se podrá verificar también la correcta asignación del ícono para la aplicación.

Fig. 2.61: Ícono de la aplicación asignado con éxito

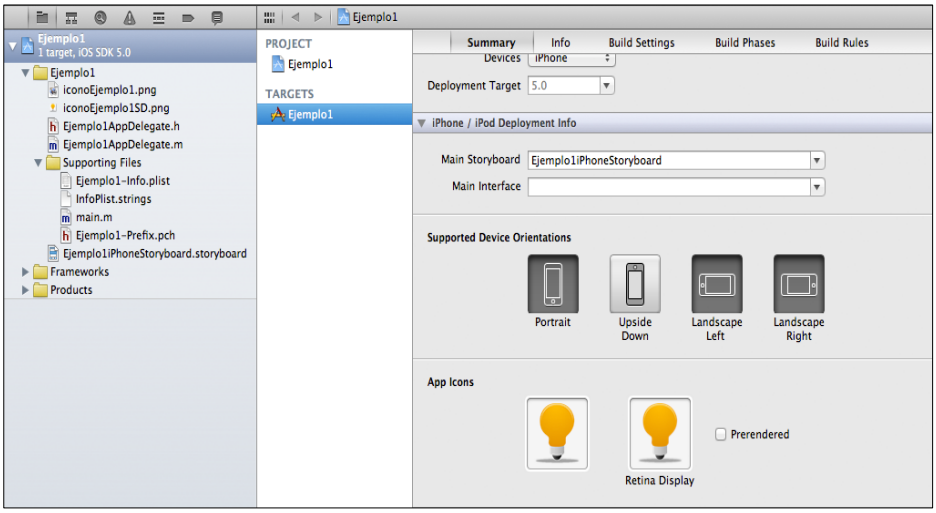


Fig. 2.62: Ícono de la aplicación asignado al proyecto exitosamente

2.3.3 Agrupamiento de ficheros dentro de un proyecto

Cuando se dispone de varios archivos dentro de un proyecto, conviene agruparlos para encontrarlos fácilmente y poder trabajar con ellos. Para esto, se debe dar *click* derecho sobre el Área de Navegación y seleccionar "*New Group*" como se indica en la Fig. 2.64.

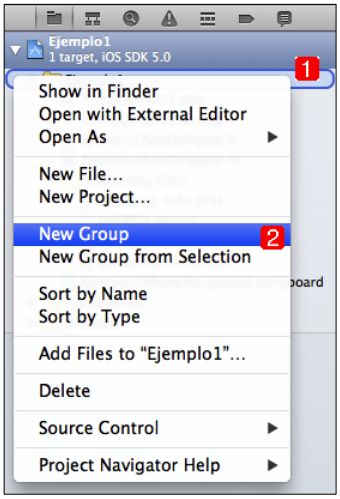


Fig. 2.64: Creación de un nuevo grupo

Fig. 2.63: Archivos imagen agrupados dentro del proyecto

A continuación se debe nombrar al nuevo grupo "*Imágenes*" y arrastrar los archivos imagen dentro de ésta carpeta o grupo como se muestra en la Fig. 2.63.

Se puede crear un grupo a partir de una selección de archivos, para esto se deben seleccionar los archivos a agrupar, en este caso los archivos controladores de la clase *AppDelegate,* dar *click* derecho sobre los mismos y seleccionar "*New Group from Selection*" como se indica en la Fig. 2.65. Nombrar al grupo contenedor de estos archivos "*Controlador*".

Finalmente se debe crear un grupo llamado "*Vista*" para agrupar los archivos de interface gráfica de usuario, que, en este caso es únicamente el *Storyboard.* El proyecto, por el momento debe tener los archivos y grupos de la Fig. 2.66.

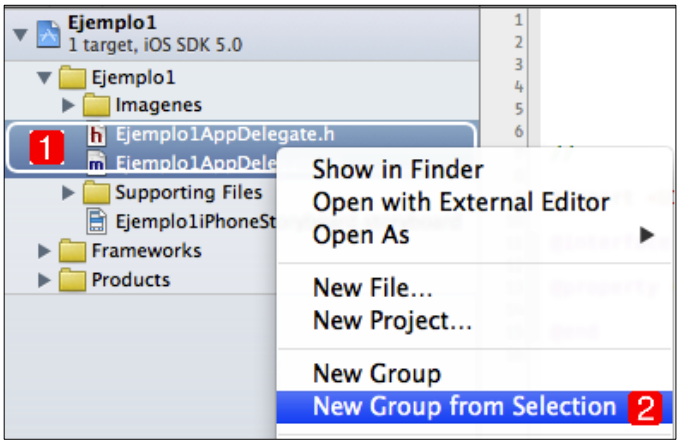


Fig. 2.65: Nuevo grupo a partir de selección de archivos

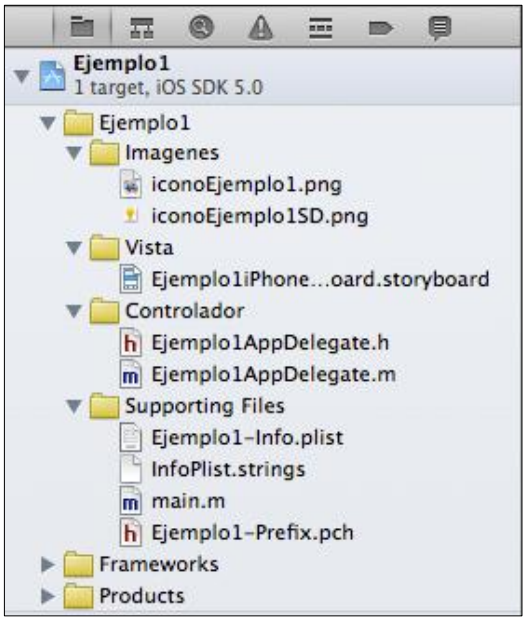


Fig. 2.66: Archivos agrupados

2.4 Cómo crear interfaces gráficas de usuario

Crear interfaces de usuario para aplicaciones *iOS* mediante *XCode* es relativamente sencillo. El proceso consiste en arrastrar los componentes sobre las ventanas o vistas que tendrá la aplicación. A partir de la versión 4.2 de *XCode* se han implementado *Storyboards* que permiten visualizar el flujo de ventanas y componentes dentro de la aplicación.

Antes de construir interfaces gráficas de usuario es importante conocer algunos de los componentes que se pueden utilizar dentro de una aplicación, por lo tanto, en el siguiente apartado, se realizará un análisis de los componentes que comúnmente se encuentran en aplicaciones *iOS*.

2.4.1 Análisis de componentes comunes

Los componentes que se presentarán a continuación, se encuentran en el Área de Utilidades de *XCode* (Fig. 1.2). Para visualizarlos; sobre esta área, se debe dirigir a la barra de selección de librería (Fig. 2.67) y presionar el botón librería de objetos. 

Fig. 2.67: Barra de selección de librerías

De esta forma se presentará una lista de objetos que se pueden agregar a una aplicación *iOS* como se muestra en la Fig. 2.68.

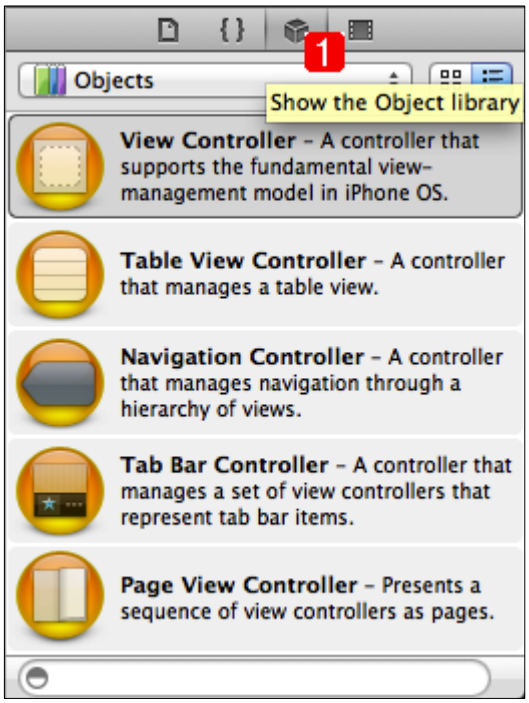


Fig. 2.68: Mostrar librería de objetos de *GUI* de una aplicación *iOS*

A continuación se presentan los principales componentes:

***View Controller*:** Controlador que soporta el manejo de vistas o ventanas en *iOS*. Permite gestionar una vista así como sus barras de herramientas y de navegación. La clase *UIViewController* soporta también vistas modales y rotativas cuando se cambia la orientación del dispositivo *iOS*. El objeto *View Controller* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.69.

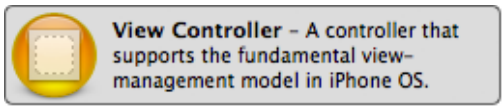


Fig. 2.69: Ícono *View Controller*

Al construir una interface gráfica de usuario, un *View Controller* vacío es igual al que se presenta en la Fig. 2.70. Se puede personalizar y cambiar el color de fondo, así como agregar componentes imagen, botón, etiquetas, entre otros. Un *View Controller* personalizado se presenta en la Fig. 2.71.

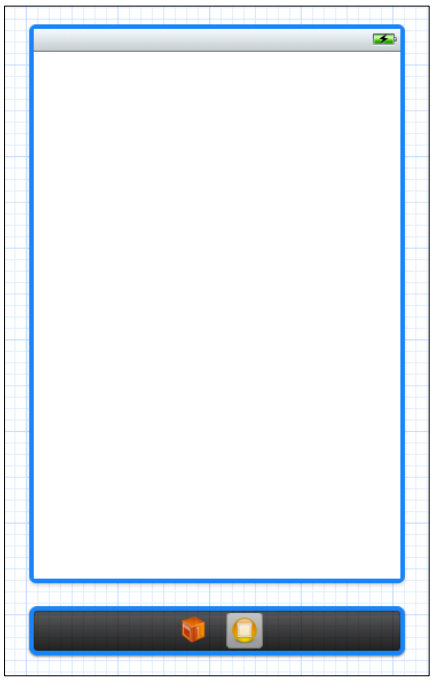


Fig. 2.70: *View Controller* vacío

Fig. 2.71: *View Controller* que presenta una *GUI*

Table View Controller: Controlador que permite manejar una vista o ventana que incorpora una tabla, generada automáticamente con dimensiones preestablecidas por el desarrollador, actúa como fuente de datos y delegado de la misma. La clase *UITableViewController* permite establecer los modos de edición de la tabla. El objeto *Table View Controller* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.72.

Fig. 2.72: Ícono Table View Controller

Al construir una interface gráfica de usuario, un *Table View Controller* vacío es como el de la Fig. 2.73. Se puede llenar la tabla con datos, agruparlos por categorías y filtrarlos. La aplicación nativa de *iOS* "Contactos" usa un *Table View Controller* para mostrar los contactos almacenados, como se puede observar en la Fig. 2.74.

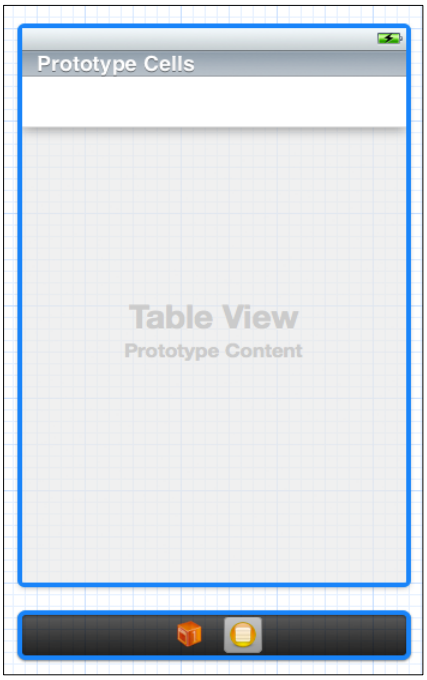


Fig. 2.73: Table View Controller vacío

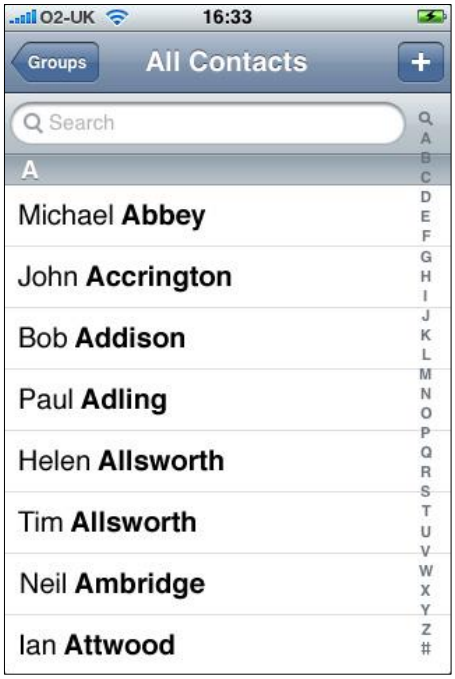


Fig. 2.74: Aplicación nativa de iOS "Contactos" usa *Table View Controller*

Navigation Controller: Controlador que maneja la navegación entre una jerarquía de vistas o ventanas. Maneja una pila de *View Controllers*, cada uno de los cuales contiene información sobre una vista, como el título y su contenido. Cuando los *View Controllers* son insertados y extraídos de la pila, el *Navigation Controller* actualiza la barra de navegación y la vista apropiadamente. El objeto *Navigation Controller* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.75.

Fig. 2.75: Ícono Navigation Controller

Al construir una interface gráfica de usuario, un *Navigation Controller* vacío es como el que se presenta en la Fig. 2.76.

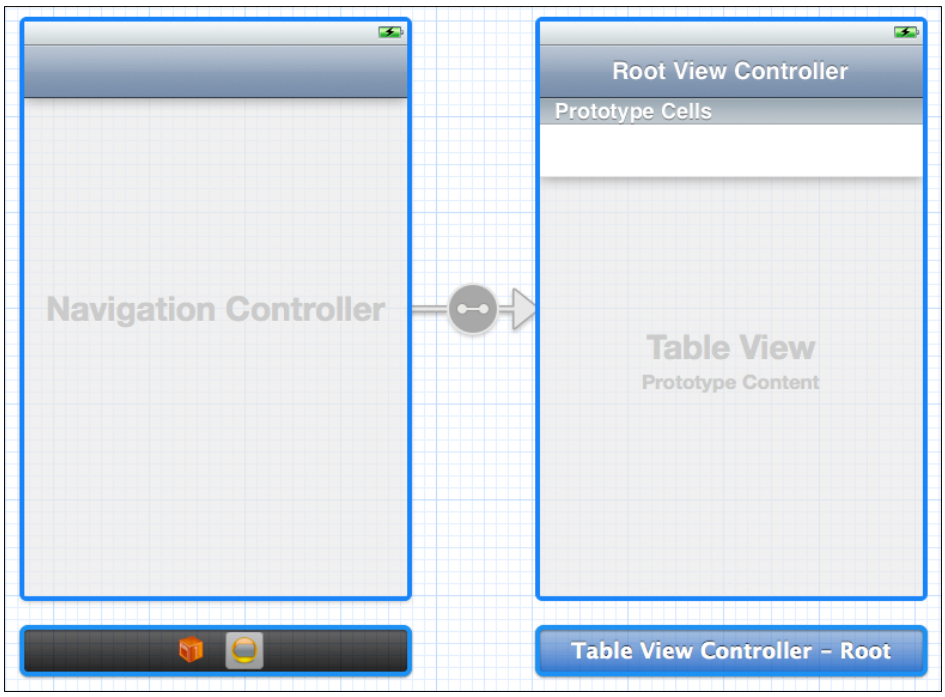


Fig. 2.76: Navigation Controller vacío

Un *Navigation Controller* sirve para interconectar varias vistas a la vez; es decir, cuando el usuario que interactúa con la aplicación presione un botón o elemento de tabla perteneciente a una vista actual, se presente otra vista. Al presentar esta vista, en la barra de navegación superior, aparecerá un botón para regresar a la vista anterior. En la Fig. 2.77 se presenta un ejemplo de *Navigation Controller* y a continuación la descripción del mismo.

Fig. 2.77: Aplicación nativa de dispositivos *iOS "*Ajustes" que incorpora un *Navigation Controller*

El ejemplo de la Fig. 2.77 corresponde a la aplicación nativa de los dispositivos *iOS* "Ajustes". Si el usuario presiona "General", se presentan los ajustes generales. Después, si el usuario presiona "*Auto-Lock*", se presentarán los rangos de tiempo para auto bloquear el dispositivo *iOS*. En cualquier momento el usuario puede regresar a la vista anterior presionando el botón correspondiente en la barra superior de navegación.

Tab Bar Controller: Controlador que maneja un conjunto de *View Controllers* que representan elementos del *Tab Bar* o barra de pestañas. Cada *View Controller* presenta información sobre la pestaña a la que corresponde y suministra la vista a ser presentada cuando se seleccione la pestaña del *Tab Bar*. El objeto *Tab Bar Controller* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.78.

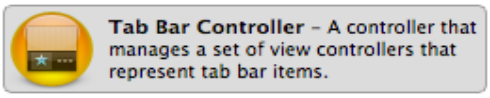


Fig. 2.78: Ícono Tab Bar Controller

Al construir una interface gráfica de usuario un *Tab Bar Controller* de dos pestañas, es como el presentado en la Fig. 2.80. Puede incorporar hasta 5 pestañas. En la Fig. 2.79, el *Tab Bar* está compuesto de 5 pestañas. En la Fig. 2.81 se ha implementado un *Tab Bar* de 4 pestañas.

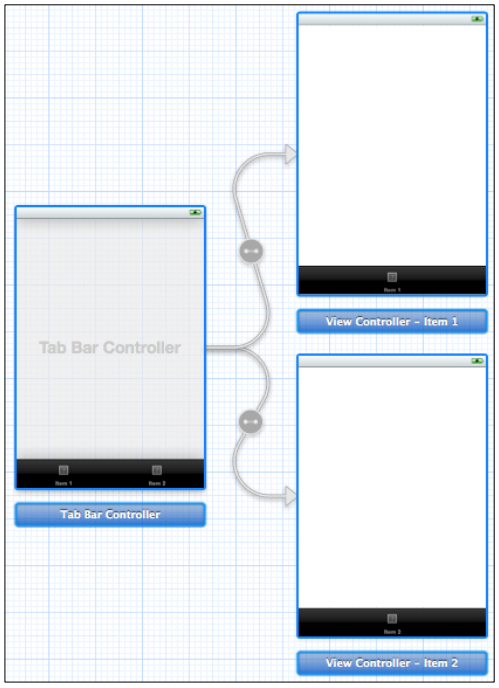


Fig. 2.80: *Tab Bar Controller* de 2 pestañas

Fig. 2.79: *Tab Bar Controller* de la aplicación nativa de *Apple* "Teléfono" (5 pestañas)

Fig. 2.81: *Tab Bar Controller* de 4 pestañas

***Label* o etiqueta:** Es una porción de texto de tamaño variable. Implementa una vista de texto de lectura únicamente. La clase *UILabel* puede reducir, envolver o truncar el texto, dependiendo del tamaño del rectángulo delimitador y las propiedades que establezca el desarrollador; el mismo que puede controlar el tipo de letra, color, alineamiento, resaltado y sombreado del texto en la etiqueta. El objeto *Label* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.82.

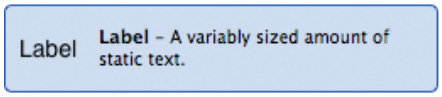


Fig. 2.82: Ícono *Label*

Al construir una interface gráfica de usuario, las etiquetas o *Labels* dentro de una vista, se ven como en la Fig. 2.83, donde se pueden diferenciar tres: *"Label 1"* ubicada en la esquina superior izquierda, *"Label 2"* ubicada en la esquina inferior derecha y *"Etiqueta de Texto"* ubicada en el centro de la vista.

Fig. 2.83: *View Controller* con 3 *Labels* personalizados

***Round Rect Button* o botón:** Implementa un botón que intercepta los eventos de presión sobre la pantalla y envía un mensaje de acción cuando es pulsado. El desarrollador puede establecer el título, imagen y otras propiedades de apariencia. Adicionalmente, se puede especificar una apariencia diferente para cada estado del botón. El objeto *Round Rect Button* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.84.

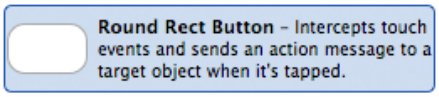


Fig. 2.84: Ícono *Round Rect Button*

Al construir una interface gráfica de usuario, los botones o *Round Rect Buttons* dentro de una vista, se ven como en la Fig. 2.85, donde se pueden diferenciar cinco: El tradicional *Round Rect Button*, seguido de un botón personalizado con una imagen "*Push Me*", y los botones tradicionales de interfaces de usuario en *iOS*, para añadir, desplegar detalles e información respectivamente.

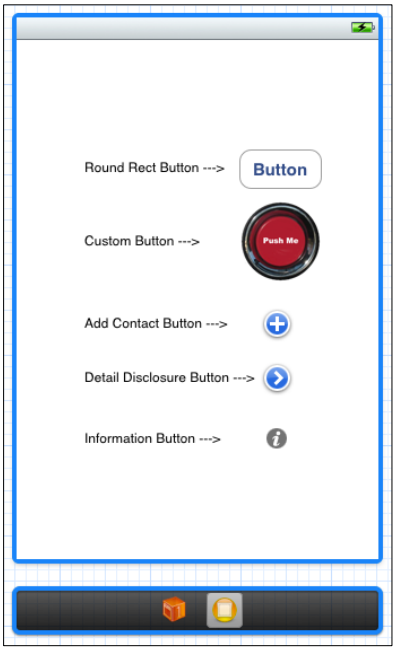


Fig. 2.85: *View Controller* con 5 botones

Segmented Control: Elemento que comprende varios segmentos, cada uno de los cuales funciona como un botón discreto. Cada segmento puede mostrar texto o una imagen, pero no ambos. La clase *UISegmentedControl* asegura que el ancho de cada segmento sea proporcional, basado en el numero total de segmentos, a menos que el desarrollador establezca un ancho específico. El objeto *Segmented Control* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.86.

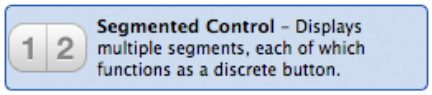


Fig. 2.86: Ícono *Segmented Control*

Al construir una interface gráfica de usuario, los *Segmented Control* en una vista se ven como en la Fig. 2.87, donde se pueden diferenciar tres: El primero, compuesto de tres segmentos con imágenes: un velocímetro, un auto y una llave. El segundo, tiene un estilo "Barra" y consta de dos segmentos. El ultimo, se compone de 3 segmentos de texto.

Fig. 2.87: View Controller con 3 Segmented Control

Text Field o* Campo de Texto*: Rectángulo redondeado que puede contener texto editable. Al presionarlo, aparece el teclado; si el usuario presiona "*Return*" sobre éste, desaparece y la aplicación puede manejar la entrada. La clase *UITextField* soporta la superposición de vistas para mostrar información adicional y provee un control para limpiar el contenido del *Text Field*. El objeto *Text Field* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.88.

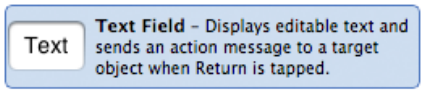


Fig. 2.88: Ícono *Text Field*

Al construir una interface gráfica de usuario, los *Text Field* en una vista se ven como en la Fig. 2.89, donde se pueden diferenciar dos: El verde, para ingresar un nombre de usuario, presenta el control de borrado rápido. El anaranjado, para ingresar la contraseña; indica mediante el texto "*Type Here*" que debería presionarlo, al hacerlo, éste texto desaparece. Si se marca la propiedad "*Secure*" del *Text Field*, se asegura de que nadie verá la contraseña presentando solo caracteres "·".

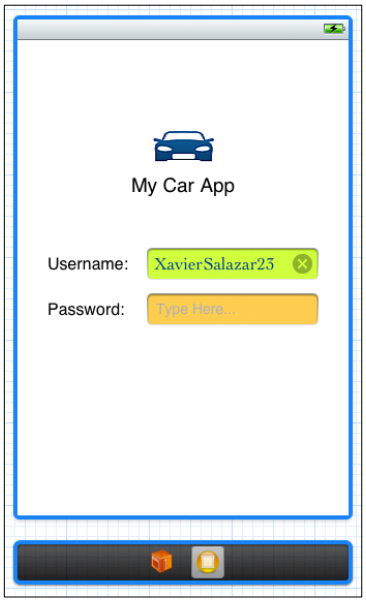


Fig. 2.89: View Controller con 2 Text Field

***Switch*:** Presenta un elemento que muestra el estado booleano de un valor. Si el usuario presiona el control, puede alternar entre los estados ON/OFF. El objeto *Switch* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.90.

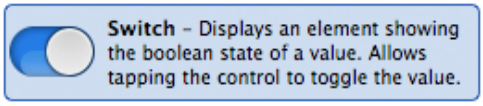


Fig. 2.90: Ícono *Switch*

Al construir una interface gráfica de usuario, los *Switch* dentro de una vista se ven como en la Fig. 2.91, donde se pueden diferenciar tres: Un *Switch* con su coloración por defecto; azul, en estado ON, en la parte superior izquierda de la vista. Un segundo *Switch* en ubicado en la mitad de la misma en estado OFF, y; un último switch en la parte inferior derecha de la vista en estado ON, pero personalizado de color rojo.

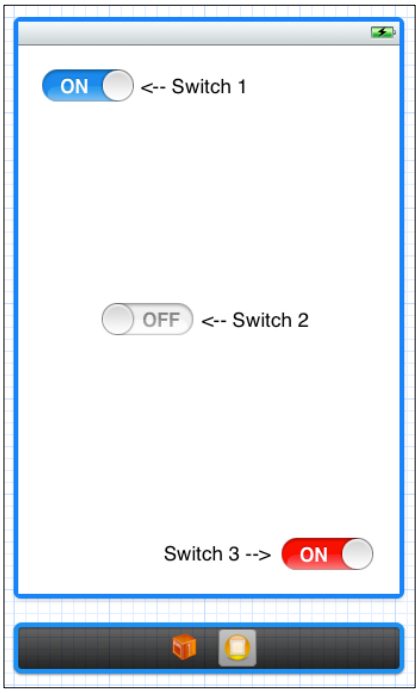


Fig. 2.91: View Controller con 3 Switches

***Activity Indicator View*:** Indicador de actividad que informa la ejecución de una tarea o proceso de duración desconocida (Si se conoce la duración de la actividad se recomienda que el desarrollador utilice en su lugar un *Progress View*). Mientras la tarea o proceso se encuentre en ejecución, el indicador de actividad girará. El usuario no interactúa con un indicador de actividad. El objeto *Activity Indicator View* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.92.

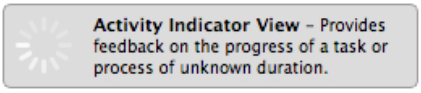


Fig. 2.92: Ícono Activity Indicator View

Al construir una interface gráfica de usuario, un *Activity Indicator View* en una vista se ve como en la Fig. 2.93, mientras no se termine de ejecutar la tarea o proceso, los indicadores de actividad no dejan de girar. Sirven para informar al usuario que debe esperar hasta que se terminen de ejecutar las actividades o procesos, es decir, el indicador deje de girar o en su defecto, desaparezca.

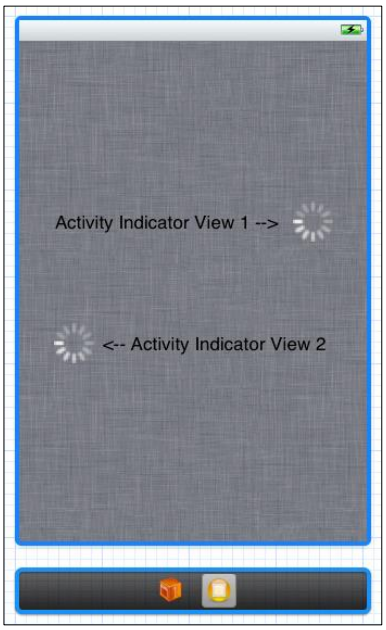


Fig. 2.93: View Controller con 2 Activity Indicator Views

***Progress View*:** Elemento que representa el progreso de una tarea sobre el tiempo. El progreso actual es representado por un valor decimal entre 0.0 y 1.0 inclusive, en donde 1.0 indica la completitud de la tarea. El objeto *Progress View* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.94.

Fig. 2.94: Ícono *Progress View*

Al construir una interface gráfica de usuario los *Progress View* dentro de una vista se ven como en la Fig. 2.95, en donde se han clasificado de acuerdo a su estilo. Los cuatro primeros son *Progress View* con estilo por defecto. A continuación se puede visualizar un *Progress View* con estilo "barra" y finalmente un *Progress View* con estilo personalizado. Como se puede ver, es posible personalizar el valor y el color del *Progress View*.

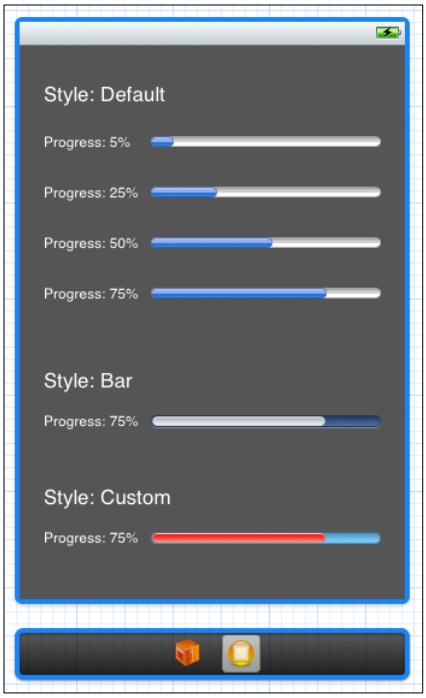


Fig. 2.95: *View Controller* con *Progress Views* clasificados por estilo

***Image View*:** Componente que sirve para presentar una imagen o una animación compuesta por un arreglo de imágenes. El objeto *Image View* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.96.

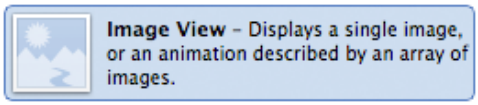


Fig. 2.96: Ícono *Image View*

Al construir una interface de usuario, un *Image View*, al cual aún no se le ha asignado ninguna imagen, se ve como el que se encuentra en la parte izquierda de la vista de la Fig. 2.97. Al asignarle una imagen, el rectángulo celeste con la leyenda *UIImageView* es reemplazado por la misma; por lo tanto la imagen de una bombilla eléctrica amarilla en la parte derecha de la vista, es un *Image View* al cual se le ha asignado esta imagen.

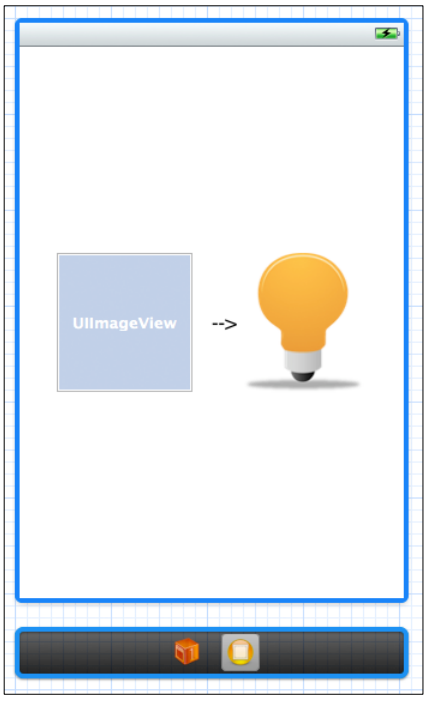


Fig. 2.97: *View Controller* con un *Image View* sin imagen asignada y otro con una imagen asignada

Web View: Presenta contenido web dentro de una vista y permite interactuar con este. El desarrollador puede enviar solicitudes de carga de contenido web en el objeto *Web View*. La clase *UIWebView* permite desplazarse a través del historial web, y ajustar propiedades del contenido mediante programación. El objeto *Web View* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.98.

Fig. 2.98: Ícono *Web View*

Al construir una interface gráfica de usuario, un *Web View* dentro de una vista se ve como un rectángulo celeste con la leyenda *UIWebView* como se muestra en la Fig. 2.99. Cuando el usuario ejecute la aplicación, dentro de este rectángulo podrá visualizar e interactuar con el contenido web que el desarrollador haya establecido, como se puede ver en la Fig. 2.100.

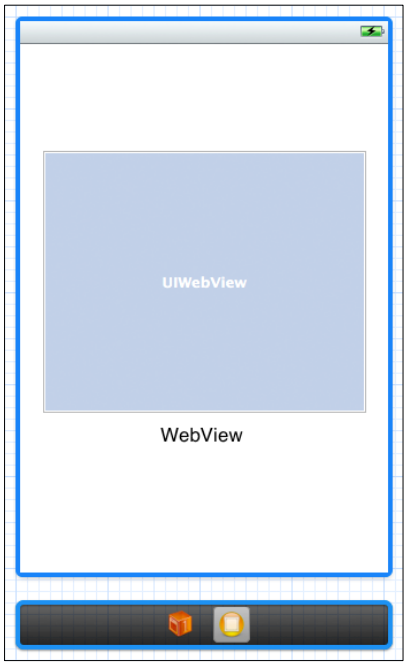


Fig. 2.99: *Web View* dentro de un *View Controller*

Fig. 2.100: *Web View* cargando contenido web

Map View: Presenta una interface gráfica de mapa dentro de una vista, similar a la aplicación nativa de *iOS* "Mapas". La clase *MKMapView* ayuda a presentar información y manipular los contenidos de un mapa. Se puede centrarlo en una coordenada dada, especificar el tamaño del área a mostrar; y, añadir pines con información personalizada. El objeto *Map View* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.101.

Fig. 2.101: Ícono *Map View*

Al construir una interface gráfica de usuario, un *Map View* dentro de una vista se ve como un rectángulo celeste con la leyenda *MKMapView* como se muestra en la Fig. 2.102. Cuando el usuario ejecute la aplicación, dentro de este rectángulo podrá visualizar el mapa con las características que el desarrollador haya establecido, como se puede ver en la Fig. 2.103.

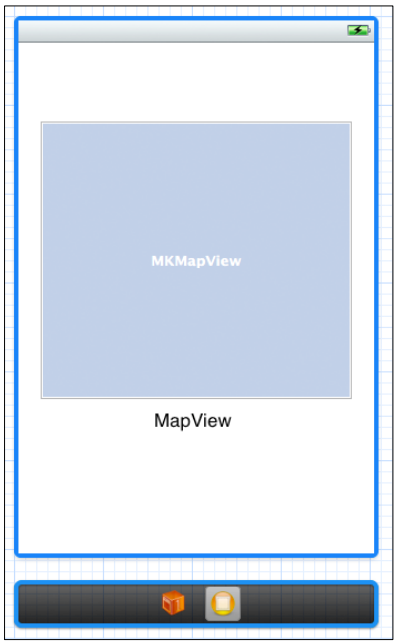


Fig. 2.102: *Map View* dentro de un *View Controller*

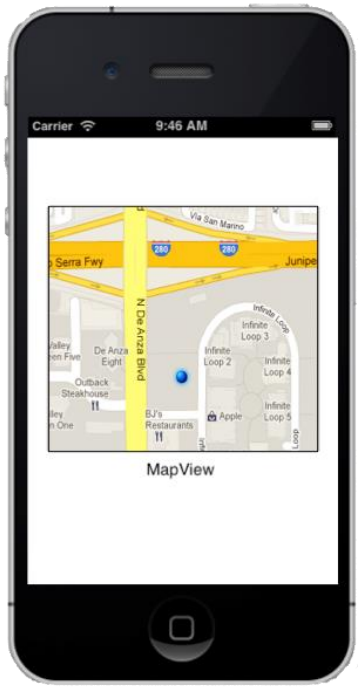


Fig. 2.103: *Map View* presentando información de mapa

Picker View: Elemento de interface gráfica de usuario potencialmente multidimensional, conformado por componentes y filas. Un componente es una rueda girante, que consta de una serie de elementos o filas. Cada fila en un componente tiene su propio contenido, que puede ser una cadena o un objeto de vista como una etiqueta (*label*) o una imagen. El objeto *Picker View* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.104.

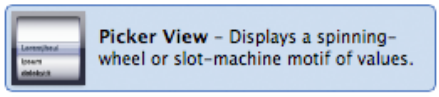


Fig. 2.104: Ícono *Picker View*

Al construir una interface gráfica de usuario, un *Picker View* en una vista se ve como en la Fig. 2.105. Este *Picker View* está conformado de un solo componente compuesto de filas de lugares. Dentro del componente se pueden seleccionar lugares como: *Mountain View*, *Sunnyvale*, *Cupertino*, etc. Un elemento *Date Picker* es un *Picker View* de 3 componentes para seleccionar una fecha y se lo presenta en la Fig. 2.106.

Fig. 2.105: *Picker View* de un solo componente

Fig. 2.106: *Picker View* de 3 componentes: *Date Picker*

Ad BannerView: La clase *ADBannerView* provee una vista que muestra banners publicitarios. Si el usuario presiona un banner, se ejecuta una acción programada en la publicidad. Por ejemplo, se puede presentar un video, otra publicidad adicional, abrir la aplicación *Safari* para mostrar una pagina web, entre otras. El desarrollador es remunerado cuando un usuario ve o interactúa con la publicidad. El objeto *Ad BannerView* se encuentra en la librería de objetos de la Fig. 2.68. Su ícono se presenta en la Fig. 2.107.

Fig. 2.107: Ícono *Ad BannerView*

Al construir una interface gráfica de usuario, un *Ad BannerView* en una vista se ve como un rectángulo celeste con la leyenda *ADBannerView* como se muestra en la Fig. 2.108. Cuando el usuario ejecute la aplicación, dentro de este rectángulo visualizará publicidad como se muestra en la Fig. 2.109 y podrá interactuar con la misma.

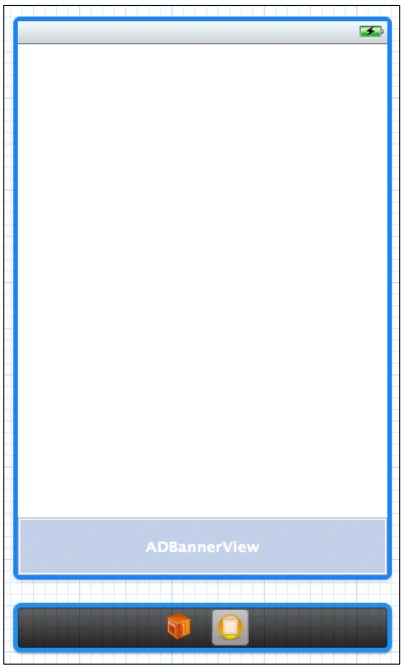


Fig. 2.108: Ad BannerView dentro de un View Controller

Fig. 2.109: *Ad BannerView* presentado publicidad

2.4.2 Construcción de una interface gráfica de usuario

Ahora que se han presentado los principales componentes que pueden conformar una interface gráfica de usuario en *iOS,* se explicará como crear una, utilizando algunos de estos componentes y ajustando sus propiedades de acuerdo a los requerimientos de la interface gráfica de usuario a construir.

A continuación se construirá una *GUI* para presentar los vehículos disponibles en la empresa *Fadem's Motor Cía. Ltda*. Los requerimientos son los siguientes:

1. Un banner de la empresa en la parte superior de la vista
2. Información del vehículo: descripción, año, kilometraje y ciudad de matrícula o placa.
3. Seis fotos del vehículo con un filtro para visualizar:
    a. 3 fotos del interior o
    b. 3 fotos del exterior del mismo.
4. Un botón para llamar directamente a un representante de la empresa para solicitar mayor información.
5. Un botón para solicitar el precio del vehículo.
6. Un campo de texto para escribir el correo del usuario que solicita el precio, al cual, la empresa responderá.
7. Flechas de desplazamiento que permitan visualizar el vehículo siguiente y anterior.

Se implementará la solución de la Fig. 2.110, que como se puede observar, satisface todos los requerimientos anteriormente descritos.

Fig. 2.110: Interface gráfica de usuario para presentar los vehículos disponibles en la empresa

Construcción de la interface gráfica de usuario.

Paso 1: Crear un proyecto vacío; en caso de requerir ayuda, se puede referir al apartado *"2.2 Cómo crear un nuevo proyecto"* y nombrarlo como se desee. En el tutorial se lo nombrará *EjemploGUI*.

Paso 2: Agregar un *Storyboard* al proyecto, denominado *EjemploGUIiPhoneStoryboard.storyboard* y asignarlo a la aplicación; si se requiere ayuda, se puede referir al apartado *"2.3.1 Cómo agregar un fichero a un proyecto"* específicamente al subapartado *"a) Agregar un Storyboard"*

Paso 3: Agregar al proyecto los archivos de imágenes necesarios para construir la interface, los cuales se encuentran en el archivo comprimido *ImagenesEjemploGUI.zip*; si se requiere ayuda, se puede referir al apartado *"2.3.1 Cómo agregar un fichero a un proyecto"* específicamente al subapartado *"c) Agregar un archivo de imagen"*

Paso 4: Agrupar los ficheros del proyecto en carpetas: imágenes, vista y controlador; si se requiere ayuda, se puede referir al apartado *"2.3.3 Agrupamiento de ficheros dentro de un proyecto".*

Paso 5: Agregar al proyecto una clase *UIViewController* y nombrarla *EjemploGUIViewController;* si se requiere ayuda, se puede referir al apartado *"2.3.1 Cómo agregar un fichero a un proyecto"* específicamente al subapartado *"b) Agregar una clase UIViewController"*

Hasta el momento, los ficheros en el proyecto, deberían ser los siguientes:

Fig. 2.111: Ficheros del proyecto *EjemploGUI*

Paso 6: Asignar la clase controladora, *EjemploGUIViewController,* a una vista de interface gráfica o *View Controller*. Para esto, se debe seleccionar el archivo *EjemploGUIiPhoneStoryboard.storyboard* y agregar un *View Controller* sobre el mismo; esto se consigue al arrastrar el componente desde la librería de objetos de la Fig. 2.68, hacia el *Storyboard* como se muestra en la Fig. 2.112.

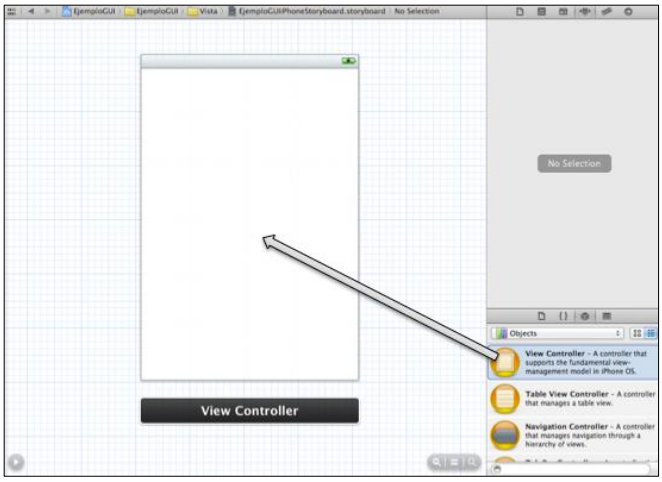


Fig. 2.112: Arrastrar un componente View Controller sobre un *Storyboard*

Las propiedades de los componentes de interface gráfica de usuario, se pueden visualizar el en Área de Utilidades de *XCode* (Fig. 1.2). Para esto, en ésta área, se debe dirigir a la barra de selección de inspector (Fig. 2.113) y se debe presionar el botón correspondiente a las propiedades que se deseen visualizar del componente.

Fig. 2.113: Barra de selección de inspector

Para asignar la clase controladora al *View Controller*, se lo debe seleccionar; y presionar el botón que muestra el inspector de identidad () en la barra de selección de inspector. En la parte inferior de esta barra, se puede seleccionar la clase que será la controladora del *View Controller*, en este caso *EjemploGUIViewController*, como se puede ver en la Fig. 2.114.

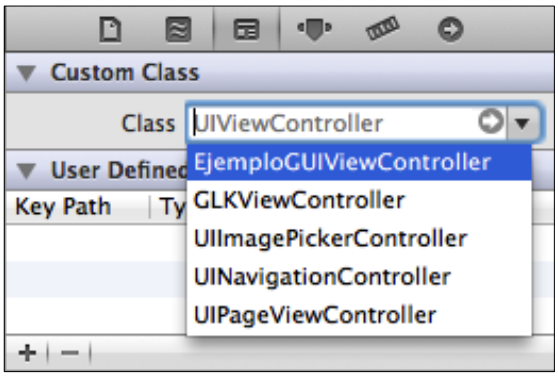


Fig. 2.114: Asignación de clase controladora del *View Controller*

Paso 7: Arrastrar y ajustar los componentes necesarios sobre el *View Controller*. Para empezar, se debe arrastrar un elemento *Image View*. Ver Fig. 2.115.

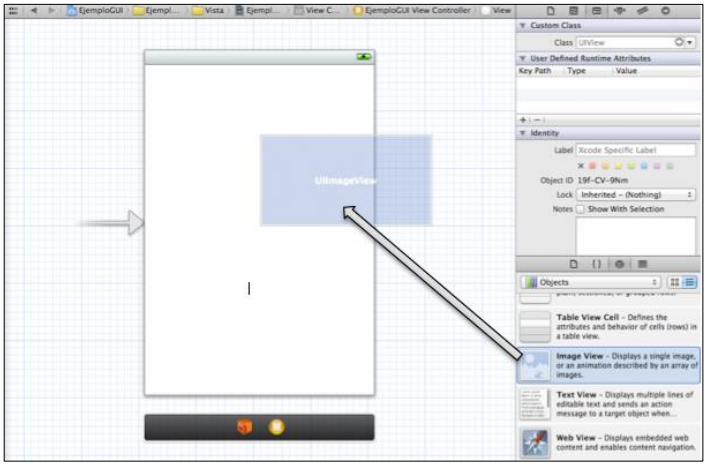


Fig. 2.115: Arrastrar un *Image View* sobre el *View Controller*

Se lo debe posicionar en la parte superior de la vista y redimensionar a 280 x 44 pixeles, para mostrar el banner de la empresa. Las líneas punteadas azules que aparecen a medida que se desplaza un componente dentro de una vista, son límites que no se deberían sobrepasar para generar una vista agradable para el usuario, según recomienda la empresa *Apple Inc*. para mantener un estándar en las interfaces *iOS*, pero; si se requiere utilizar este espacio adicional en una vista, se lo podría hacer sin ningún problema. Estas líneas son utilizadas también para indicar el alineamiento de los componentes como puede ver en la Fig. 2.116.

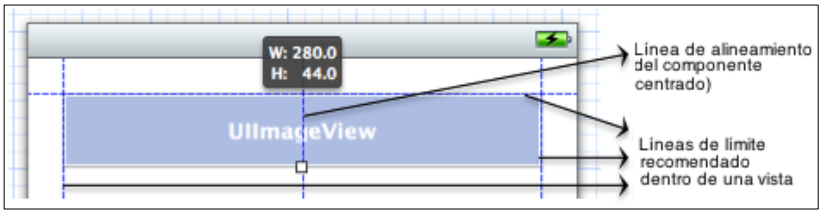


Fig. 2.116: Redimensionado y alineamiento de un componente

Finalmente se debe asignar el archivo de imagen que se presentará en el componente, para esto, se debe seleccionar el *Image View* y en la barra de selección de inspector, seleccionar el inspector de atributos  .

En el panel de atributos, en el apartado *Image*, se debe seleccionar la imagen que se desea mostrar, en este caso *FademsMotorBanner.png*. El *Image View* debe mostrar ahora el banner de la empresa como se puede ver en la Fig. 2.117.

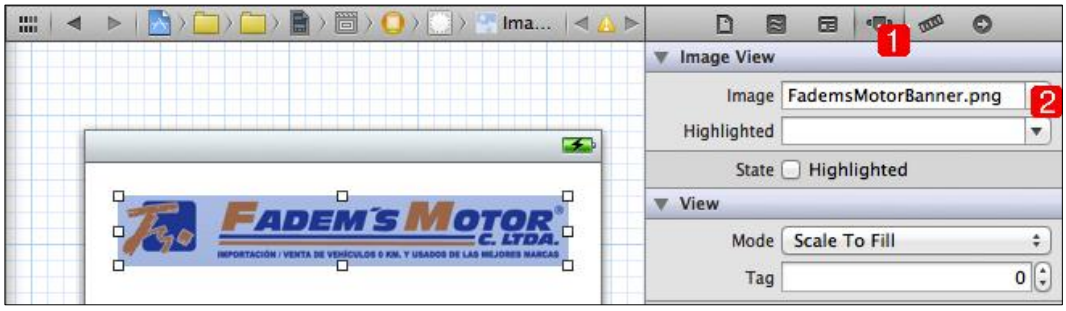


Fig. 2.117: Asignado de una imagen al componente *Image View*

A continuación se agregarán *Labels* o etiquetas de texto en donde se presentará la información del vehículo como: descripción, año, kilometraje y placa. Para esto se debe arrastrar una etiqueta sobre el *View Controller* y posicionarlo como se muestra en la Fig. 2.118.

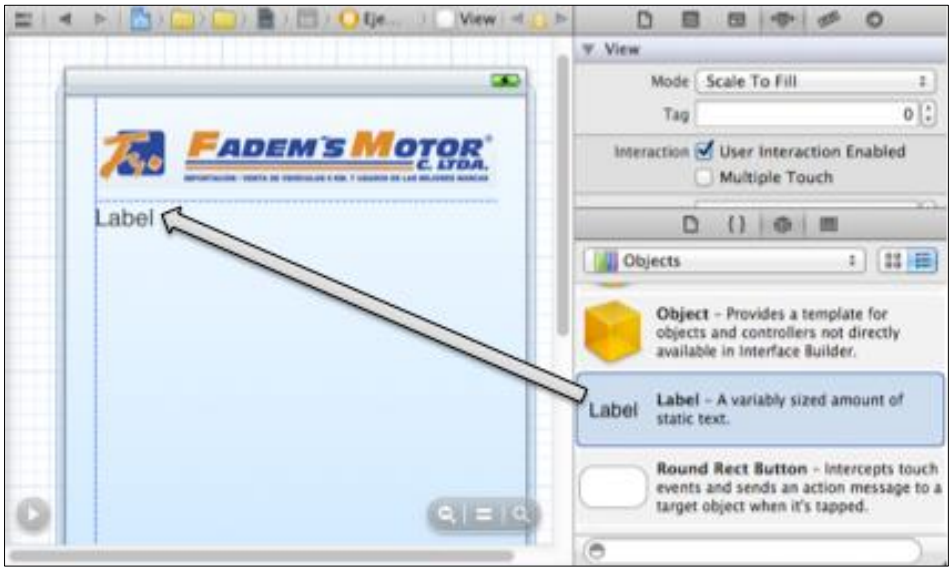


Fig. 2.118: Arrastrar un *Label* sobre el *View Controller*

Cuando el *Label* se encuentre dentro del *View Controller* se puede editar su contenido, haciendo doble *click* sobre el mismo. Escribir "*Vehículo:*" y finalizar presionando el botón "*enter*" o "*return*". Ver Fig. 2.119.

Repetir el paso anterior y arrastrar hacia el *View Controller* siete *Labels* más. Escribir en estas etiquetas de texto la información del vehículo, como se muestra en la Fig. 2.120.

Fig. 2.119: *Label* "*Vehículo:*" en vista de EjemploGUI

Fig. 2.120: Información del vehículo en *Labels*

Se pueden personalizar las etiquetas de texto mediante el panel de atributos. Para esto, se debe seleccionar la etiqueta "*Vehículo:*" y, en el panel de atributos, en el apartado *Font* presionar el botón para poder personalizarlo como se muestra en la Fig. 2.121.

En el apartado *Font*, se debe seleccionar "*System Bold"* y; en el apartado *Size*, escribir "16". Finalizar presionando el botón "*Done*".

A continuación en el apartado *Text Color* se debe seleccionar el color anaranjado llamado "*Tangerine*" como se muestra en la Fig. 2.122.

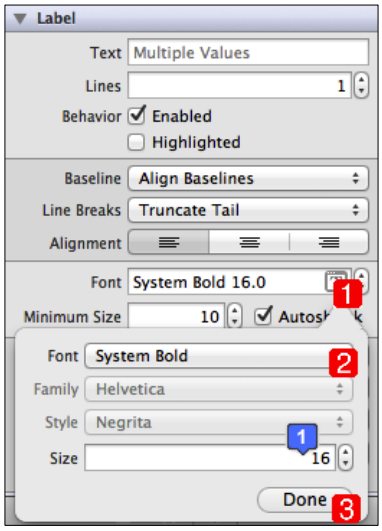


Fig. 2.121: Tipo de letra de un *Label*

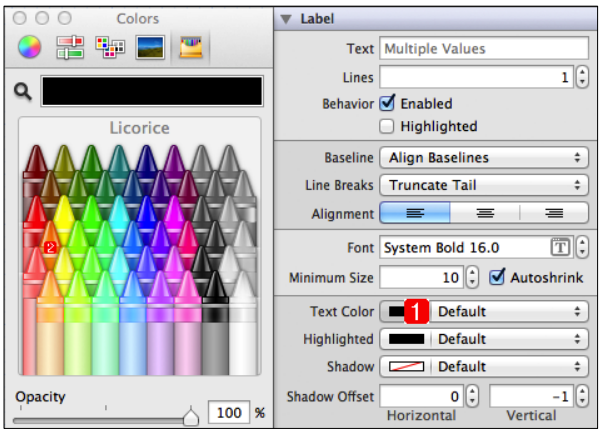


Fig. 2.122: Color de texto de un *Label*

Repetir el paso anterior de ajuste de fuente y color de texto para todos los *Label* de título. Al hacerlo, el *View Controller* se verá como el presentado en la Fig. 2.123.

Ajustar el color y la fuente de las etiquetas de descripción con las siguientes características: fuente *System Italic*, tamaño 16 y color azul o "*Midnight*" como se puede ver en la Fig. 2.124.

Fig. 2.123: *Labels* de titulo personalizados

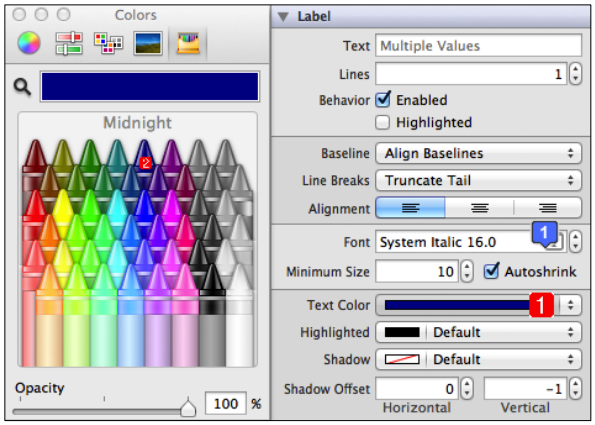


Fig. 2.124: Propiedades de *Labels* de descripción del vehículo

Hasta el momento, la interface gráfica de usuario construida debería verse como la presentada en la Fig. 2.125. Como se puede ver, las etiquetas de texto se han personalizado con los colores representativos de la empresa.

Fig. 2.125: *GUI* con una imagen y *Labels* personalizados

A continuación se implementará el botón "*Llamar*" con el objetivo de comunicarse con un representante de la empresa desde la aplicación. Para esto se debe arrastrar un *Round Rect Button* desde la librería de objetos hacia el *View Controller* como se indica en la Fig. 2.126.

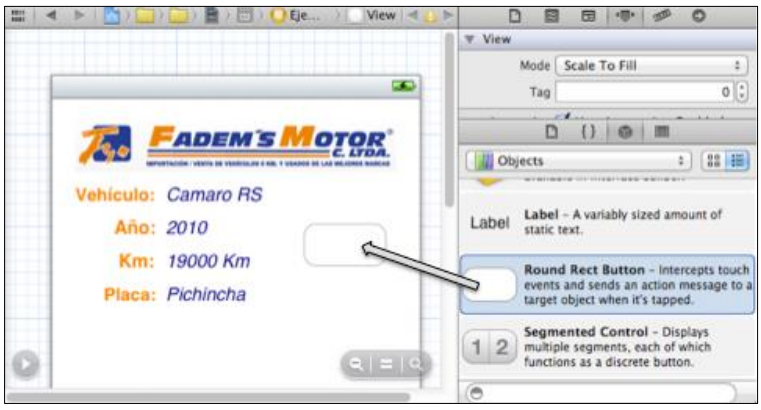


Fig. 2.126: Arrastrar un *Round Rect Button* sobre el *View Controller*

Seleccionar el botón. En sus atributos; para el apartado tipo, se debe seleccionar "*Custom*" y en el apartado imagen, seleccionar "*OrangePhone.png*". Un botón de tipo *Custom*, toma la forma de la imagen asignada; es decir, esta imagen de un teléfono

anaranjado será un botón, el cuál, podría ser programado para realizar una llamada a un representante de la empresa, cuando sea presionado. Ver Fig. 2.127.

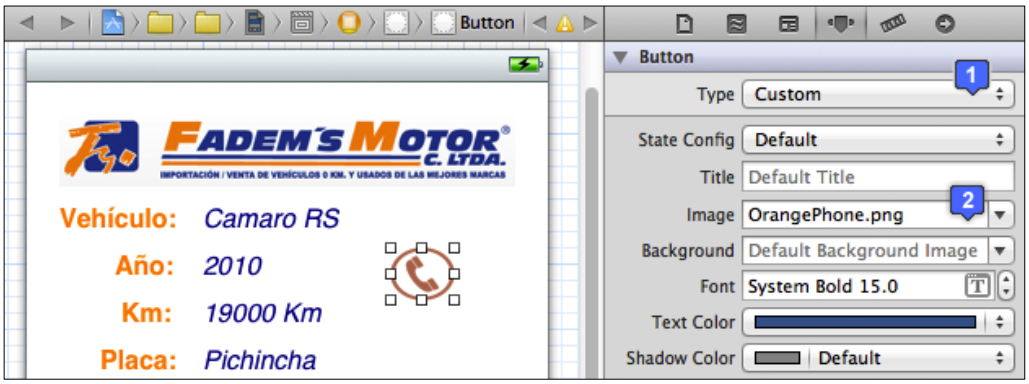


Fig. 2.127: *Round Rect Button* personalizado en forma de un teléfono

Se puede utilizar un *Label* denominado "*Llamar*", personalizarlo y posicionarlo debajo del botón teléfono como se ha hecho en la Fig. 2.128.

Fig. 2.128: Botón teléfono y *Label*

A continuación se agregarán las imágenes del vehículo, para esto se empleará un *Image View* en donde se presentará una imagen grande y 3 botones *Custom* o personalizados en donde se presentarán las imágenes pequeñas. La idea es la siguiente: cuando el usuario presione una imagen pequeña (botón), el *Image View* presentará esta imagen en un tamaño superior. Para esto se debe arrastrar un *Image View* y 3 *Round Rect Button* sobre el *View Controller* como se indica en la Fig. 2.129.

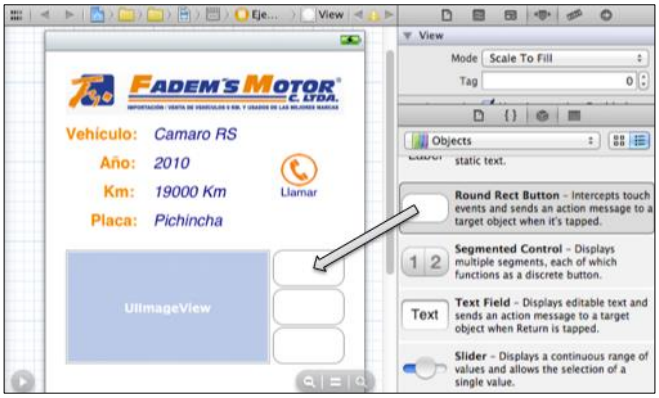


Fig. 2.129: Agregar componentes *Image View* y botones

Fig. 2.130: Componentes con imágenes asignadas

Para el *Image View* se debe asignar la imagen *CamaroBig.png*. Para los botones, seleccionar el tipo *Custom* y utilizar las imágenes *CamaroOut1.png*, *CamaroOut2.png* y *CamaroOut3.png* respectivamente. De esta forma la interface se verá como la presentada en la Fig. 2.130.

Para filtrar las fotos y visualizar únicamente fotos del interior o del exterior del vehículo a la vez, se implementará un *Segmented Control*, de esta forma, cuando el usuario presione un segmento, se presentarán solo fotos del exterior; y, si selecciona el otro segmento, se presentarán las fotos del interior. Para esto se debe arrastrar un *Segmented Control* y dos *Round Rect Buttons* sobre el *View Controller* como se indica en la Fig. 2.131.

Fig. 2.131: Arrastrar un *Segmented Control* sobre el *View Controller*

Ajustar los botones para que sean de tipo *Custom* y asignarles las imágenes "*BackOrangeArrow.png"* y "*NextOrangeArrow.png*" respectivamente. Estos botones servirán para ver los diferentes vehículos de la empresa.

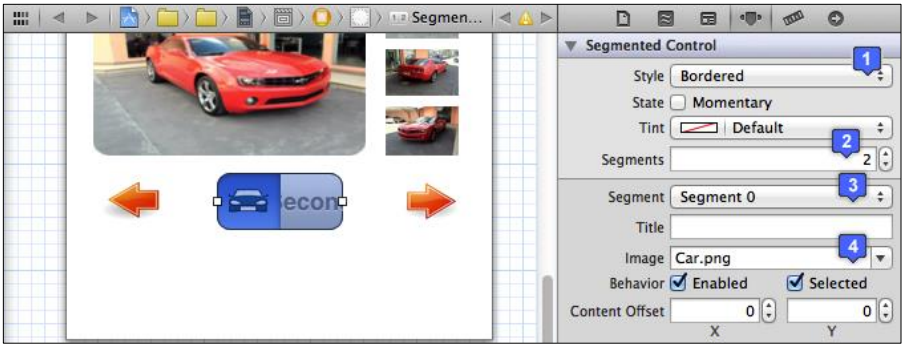


Fig. 2.132: Personalización de un *Segmented Control*

Seleccionar el *Segmented Control*, y establecer que tenga estilo "*Bordered*" y dos segmentos. Para establecer la imagen del primer segmento, se debe seleccionar "*Segment 0*" y en el apartado imagen, seleccionar "*Car.png*". Como se puede ver en la Fig. 2.132.

Se debe hacer lo mismo para establecer la imagen del segundo segmento: seleccionar el *Segmented Control*, en el apartado segmento seleccionar "*Segment 1*", y; en el apartado imagen, seleccionar "*Key.png*". Como se indica en la Fig. 2.133.

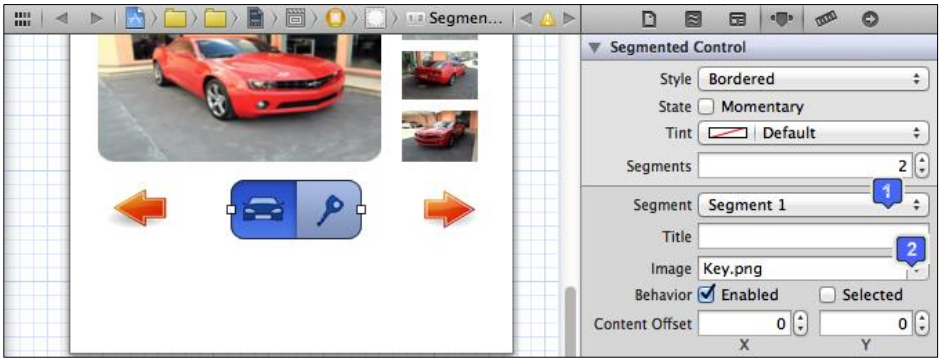


Fig. 2.133: Personalización del segundo segmento del *Segmented Control*

Agregar y personalizar etiquetas de texto debajo del *Segmented Control* para indicar al usuario la funcionalidad de cada segmento. Ver Fig. 2.134.

Fig. 2.134: Segmented Control y Labels

Para finalizar se implementará la consulta del precio del vehículo. Para esto, se debe arrastrar un *Text Field* y un *Round Rect Button* sobre el *View Controller*, como se indica en la Fig. 2.135.

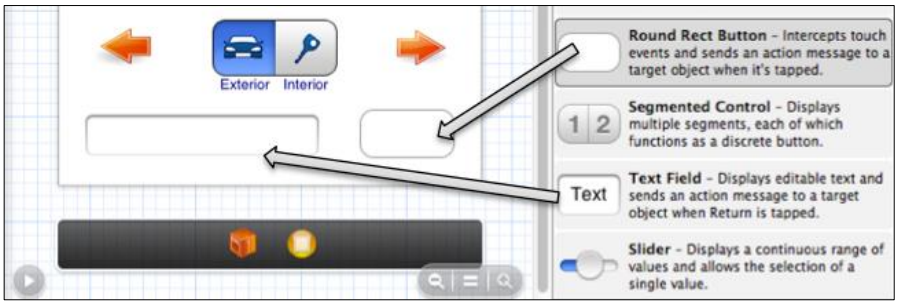


Fig. 2.135: Arrastrar un *Text Field* sobre el *View Controller*

El objetivo es que el usuario ingrese su correo electrónico en el *Text Field* y luego presione el botón para solicitar el precio del vehículo. La empresa recibirá la solicitud y responderá con el precio al correo del usuario solicitante. La validación, se podría hacer a nivel de código; por ejemplo, habilitar el botón "*Solicitar Precio*" cuando el usuario haya escrito un e-mail válido en el *Text Field.*

A continuación se debe seleccionar el *Round Rect Button* y escribir "*Solicitar Precio*" en la propiedad *Title*; seleccionar el color anaranjado o "*Tangerine*" en la propiedad

Text Color; ajustar sus dimensiones y establecer 11 para el tamaño de fuente. Ver Fig. 2.136.

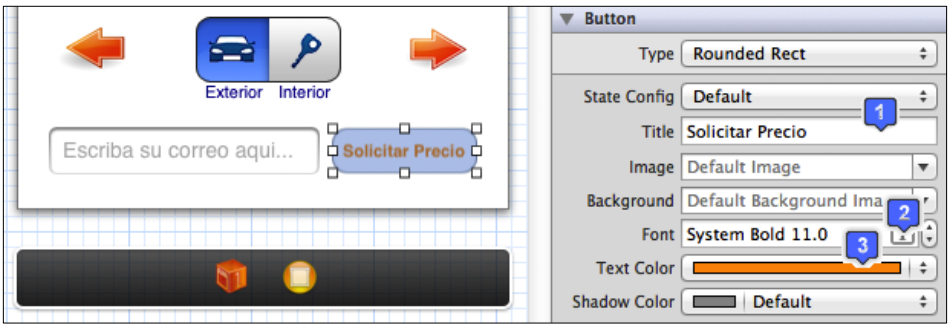


Fig. 2.136: Ajustar propiedades del *Round Rect Button*

Se debe seleccionar ahora el *Text Field* y en la propiedad *Placeholder* escribir "*Escriba su correo aquí...*". Esta propiedad sirve para indicar al usuario la información que debe escribir en el *Text Field*, su contenido desaparece cuando el usuario toca el *Text Field* para escribir en el mismo. Ver Fig. 2.137.

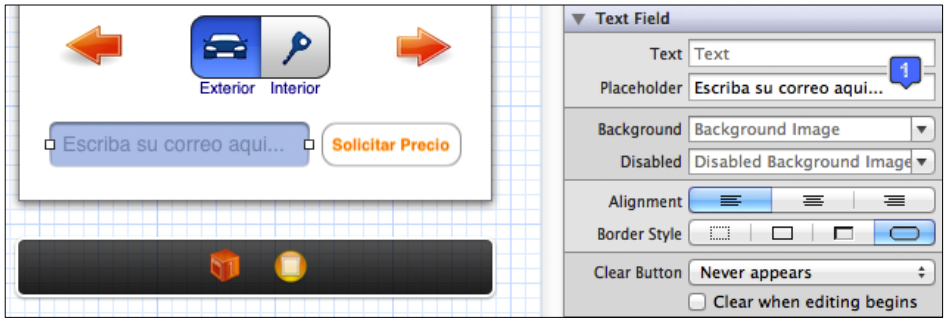


Fig. 2.137: Propiedad *Placeholder* de un *Text Field*

Para poder visualizar la interface construida en el simulador *iOS*, se deben hacer unos ajustes a nivel de código.

Ajuste 1: La clase "*EjemploGUIViewController*", que se ha agregado al proyecto para manejar el *View Controller*, presenta un método llamado *loadView* que debe ser anulado debido a que la interface fue construida mediante *Interface Builder* y no mediante código. Para esto, se debe seleccionar el archivo de implementación

EjemploGUIViewController.m, buscar el método *loadView* y comentarlo o removerlo debido a que no será requerido. El atajo para comentar una selección de código es *cmd+shift+7*. Observar la Fig. 2.138.

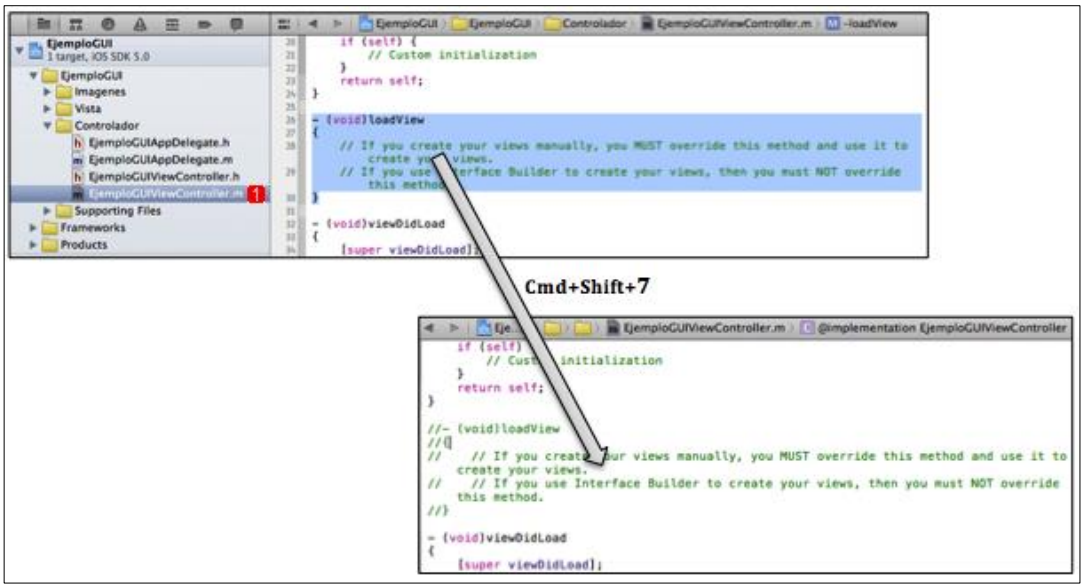


Fig. 2.138: Comentar una selección de texto (Método *loadView* de *EjemploGUIViewController.m)*

Ajuste 2: La clase *EjemploGUIAppDelegate* generada al crear el proyecto, incluye un método llamado "*application: didFinishLauchingWithOptions:"* que debe ser anulado parcialmente. Este método tiene el objetivo de crear una ventana principal, establecer un color de fondo blanco para la misma y hacerla visible. Por esta razón si este momento se ejecuta la aplicación en el simulador *iOS*, se presentará una ventana en blanco.

Para remover esta ventana y visualizar la vista que se ha construido en este apartado, se debe seleccionar el archivo *EjemploGUIAppDelegate.m*, buscar el método "*application: didFinishLaunchingWithOptions"* y eliminar las sentencias marcadas en la Fig. 2.139.

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)
    launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch.
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Fig. 2.139: Método "application: didFinishLaunchingWithOptions:"

Ahora se puede ejecutar la aplicación en el simulador *iOS*, para ello, se debe asegurar de que en la parte superior del proyecto, en "*Scheme*", se haya seleccionado "*iPhone 5.0 Simulator"* como se muestra en la Fig. 2.140, y presionar el botón 

Fig. 2.140: *EjemploGUI* se ejecutará en el simulador *iOS*

Después de que *XCode* compile el proyecto, si no se encuentra ningún error, se presenta el simulador *iOS* ejecutando la aplicación con la interface de usuario construida, como se puede ver en la Fig. 2.141.

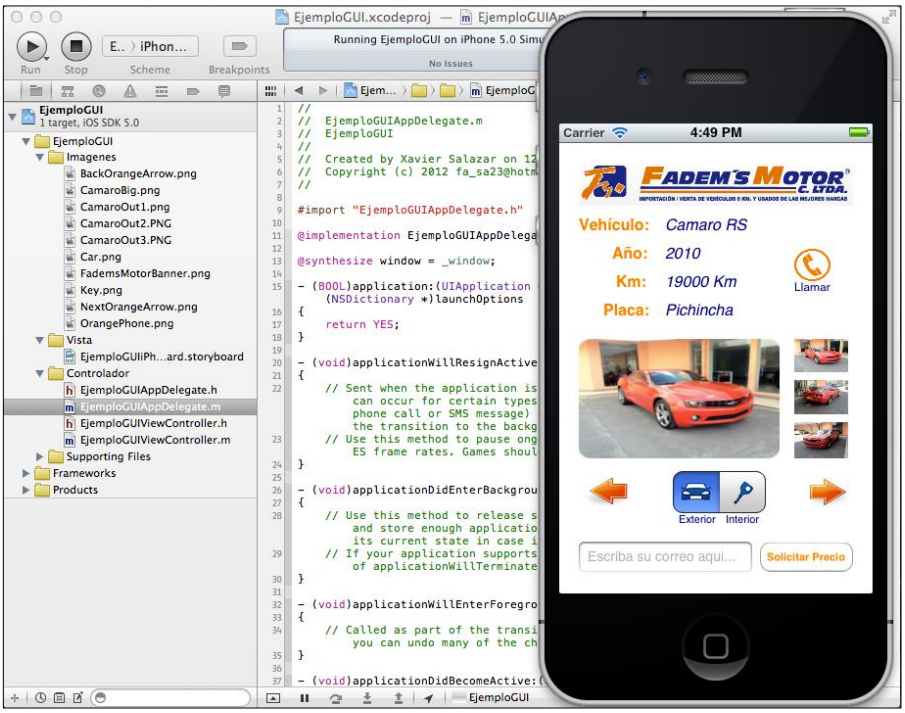


Fig. 2.141: *iOS Simulator* ejecutando la aplicación

2.4.3 Análisis de plantillas prediseñadas para un proyecto

Una vez comprendido como se puede crear un proyecto y conocido los posibles ficheros que pueden conformarlos, es necesario conocer que se puede desarrollar un proyecto a partir de plantillas prediseñadas de *XCode*. Es importante conocerlas, ya que significan una considerable reducción en el tiempo de desarrollo de una aplicación *iOS*. De esta forma el desarrollador puede construir una aplicación, pero no a partir de cero, sino a partir de una plantilla prediseñada.

Hasta ahora, al momento de crear un nuevo proyecto, apoyándose en el apartado "*2.2 Cómo crear un nuevo proyecto*", se seleccionaba siempre la plantilla aplicación vacía o *Empty Application*, pero existen otras alternativas. Para conocerlas, se debe abrir *XCode* y presionar el botón "*create a new XCode project*" de la Fig. 2.27. A continuación se debe seleccionar la categoría *iOS / Application* para mostrar las plantillas prediseñadas que se pueden utilizar, las mismas que se presentan en la Fig 2.142.

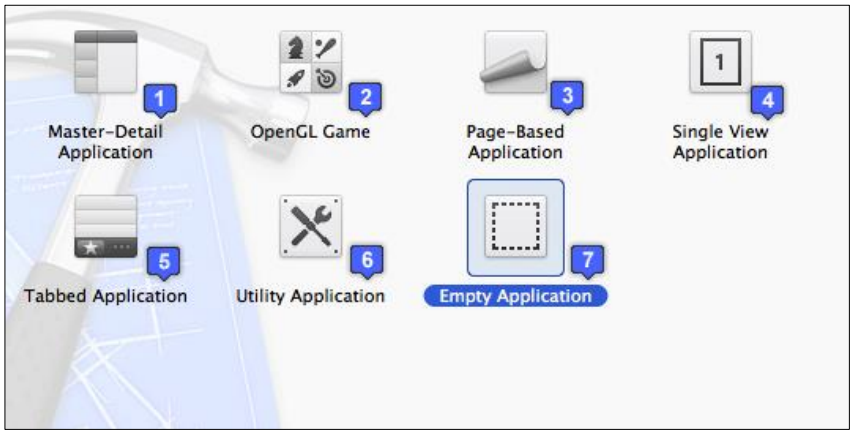


Fig. 2.142: Plantillas prediseñadas para una aplicación *iOS*

A continuación se presenta un análisis de las plantillas que se pueden seleccionar:

1. Master-Detail Application: Provee un punto inicial para una aplicación maestro-detalle. Presenta una *GUI* con un *Navigation Controller* y un *Table View* para mostrar una lista de elementos.

Al crear un proyecto utilizando esta plantilla, constará de los siguientes archivos:

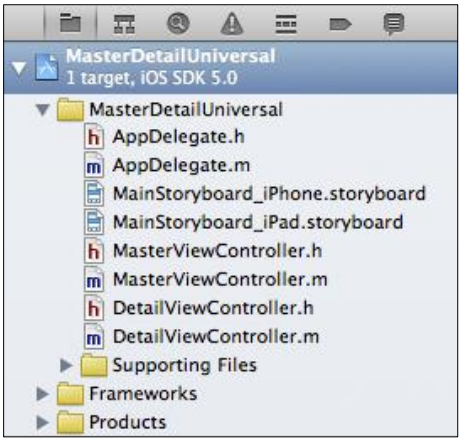


Fig. 2.143: Archivos de una aplicación *Master-Detail*

- La clase *AppDelegate* (con sus archivos cabecera e implementación)
- Un *Storyboard* para *iPhone* o un *Storyboard* para *iPad*; o ambos
- Dos clases *UIViewController* (con sus archivos cabecera e implementación): una para manejar la "vista-maestro" y otra para manejar la "vista-detalle".
- Archivos de soporte, *frameworks* y el producto

Funcionamiento: En el caso de *iPhone*, la "vista-maestro", es conformada por un *Table View* en la que se pueden añadir nuevos registros o filas. Por defecto, los registros constan de la fecha del dispositivo. Al seleccionar alguno de los registros añadidos, se presenta la "vista-detalle" con su mismo contenido. A continuación se presenta el *Storyboard* de *iPhone* de una aplicación *Master-Detail.*

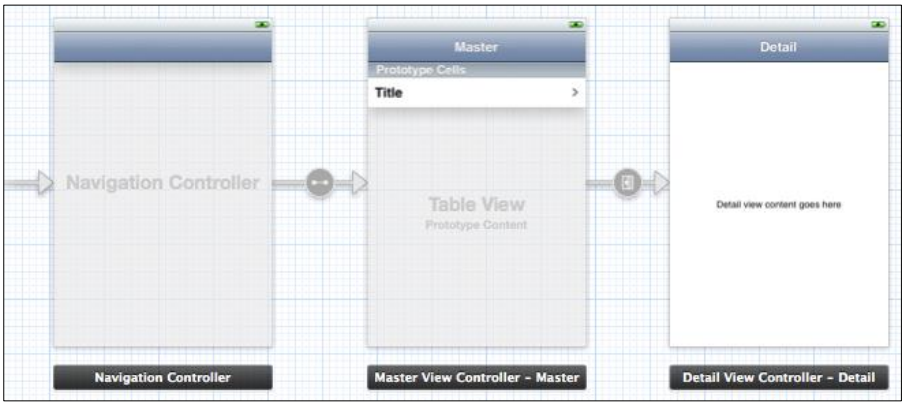


Fig. 2.144: *iPhone Storyboard* de una aplicación *Master-Detail*

Para comprender de mejor manera el funcionamiento de una aplicación *Master-Detail* en un dispositivo *iPhone*, se presenta a continuación la secuencia que tendría la misma al ser ejecutada.

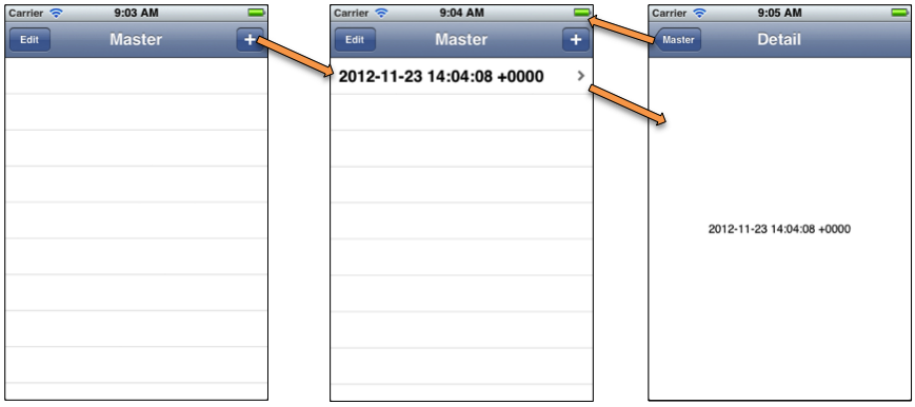


Fig. 2.145: Secuencia de una aplicación *Master-Detail* en *iPhone*

En el caso de *iPad*, las vistas tienen la misma funcionalidad, pero la forma de presentación es distinta: la "vista-maestro" aparece y desaparece sobrepuesta sobre la "vista-detalle".

A continuación se presenta el *Storyboard* de *iPad* de una aplicación *Master-Detail.*

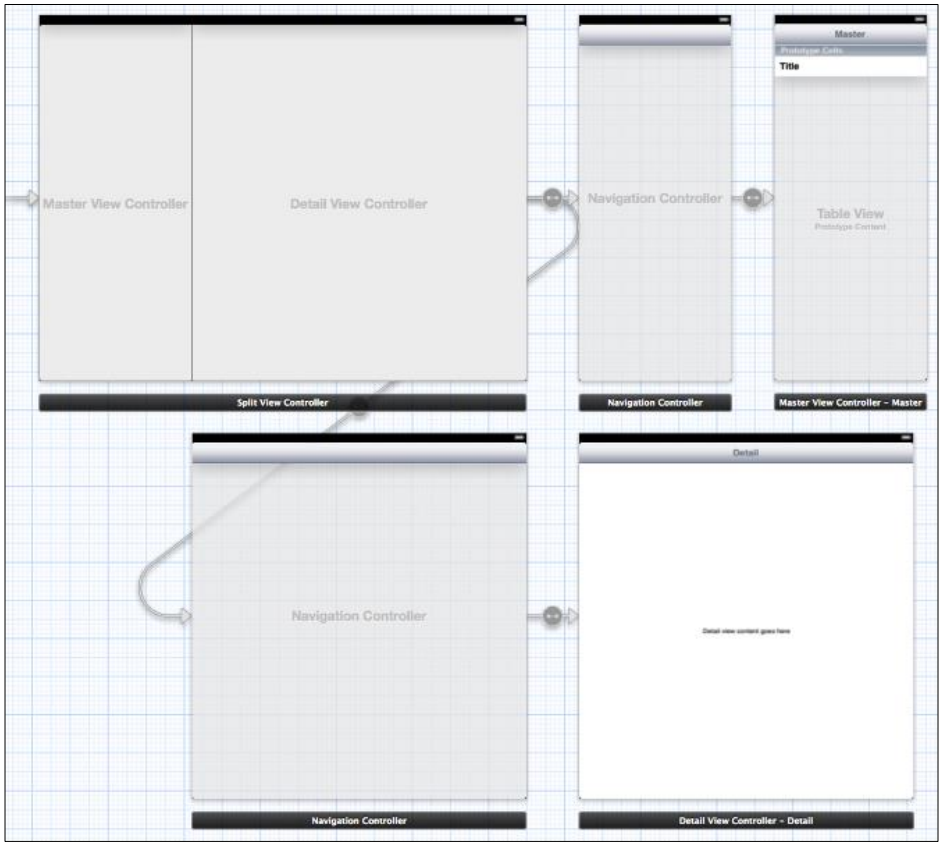


Fig. 2.146: *iPad Storyboard* de una aplicación *Master-Detail*

Para comprender de mejor manera el funcionamiento de una aplicación *Master-Detail* en un dispositivo *iPad*, se presenta a continuación la secuencia que tendría la misma al ser ejecutada.

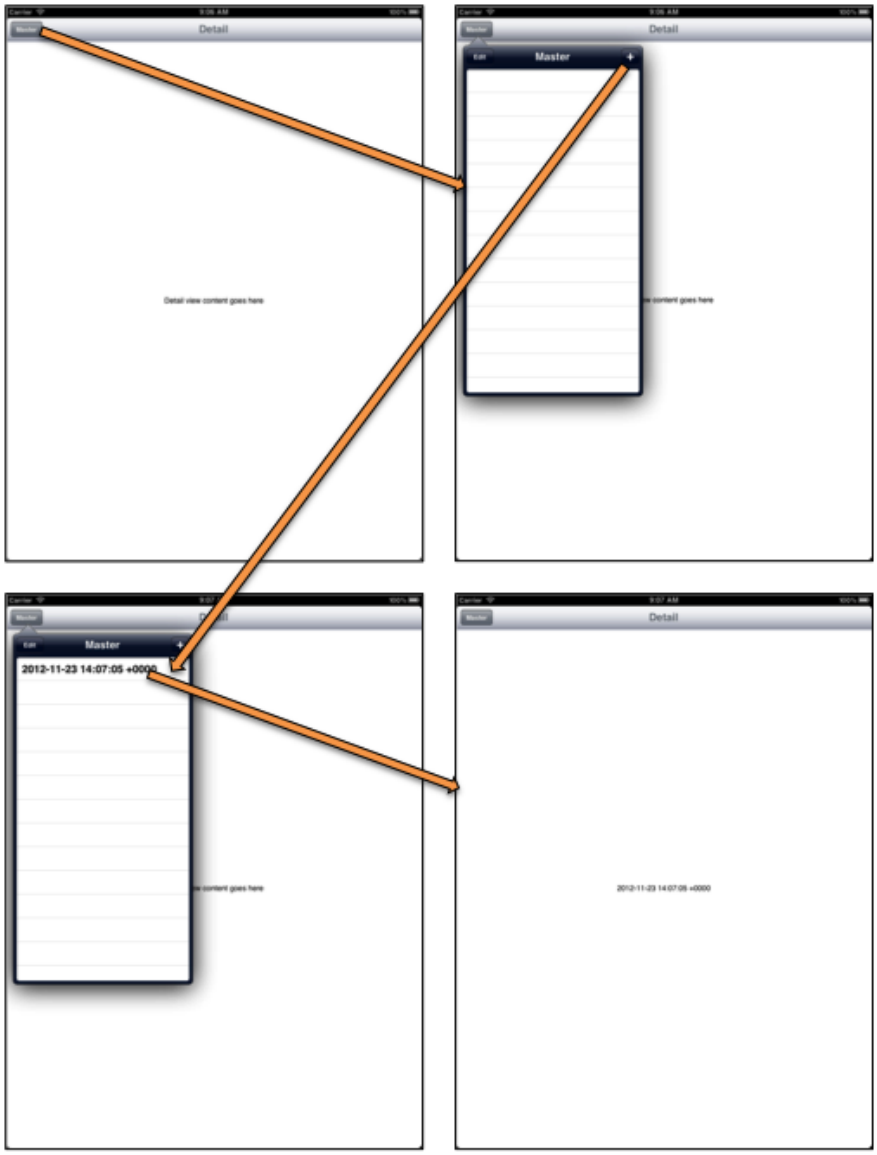


Fig. 2.147: Secuencia de una aplicación *Master Detail* en *iPad*

Si se desea, se puede analizar el código fuente auto generado de cada vista para comprender los métodos y el funcionamiento de las mismas. Esto se explicará detalladamente en el apartado "*2.5 Asignación de variables y eventos a componentes en View Controllers y Table View Controllers*" y sus sub apartados.

2. OpenGL Game: Plantilla que provee un punto de partida para un juego basado en *OpenGL*. Provee una vista en la cual se puede renderizar una escena *OpenGL*, y un temporizador o *timer* para animar la vista.

Al crear un proyecto utilizando esta plantilla, constará de los siguientes archivos:

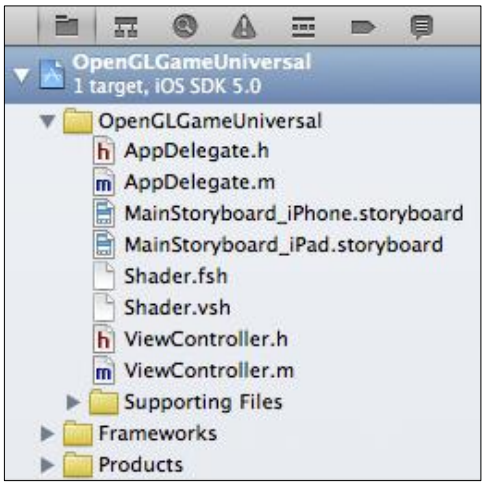


Fig. 2.148: Archivos de un juego *OpenGL*

- La clase *AppDelegate* (con sus archivos cabecera e implementación).
- Un *Storyboard* para *iPhone* o *Storyboard* para *iPad*; o ambos.
- Ficheros *Shader.fsh* y *Shader.vsh.* que son programas llamados por *OpenGL* para ejecutar el código en tiempo de renderización. El fichero *Shader.fsh* es el sombreador de fragmentos y el fichero *Shader.vsh* es el sombreador de vértices. Tienen una función similar pero son llamados en diferente tiempo durante el proceso de renderización[52]
- Una clase *UIViewController* (con su archivos cabecera e implementación) para manejar la vista que se presenta al usuario
- Archivos de soporte, *frameworks* y el producto

[52] EndoDigital. (2012). *Part Three: Getting Started in XCode*. Recuperado el 21 de Octubre de 2012, de Part Three: Getting Started in XCode: http://www.endodigital.com/opengl-es-2-0-on-the-iphone/part-three-getting-started-in-xcode/

Funcionamiento: La aplicación utiliza los archivos *Shader.fsh* y *Shader.vsh*, para renderizar un objeto y animarlo dentro de una vista. Si se analiza el archivo de implementación de la clase *UIViewController*, se pueden encontrar varios métodos que son los responsables de construir el objeto y animarlo. El *Storyboard* tanto de *iPhone* como de *iPad*, consta únicamente de un objeto *View Controller* vacío.

Para comprender de mejor manera el funcionamiento de un juego basado en *OpenGL*, se presenta a continuación el Simulador *iOS* ejecutando el mismo, sin haber realizado modificación alguna a la plantilla utilizada.

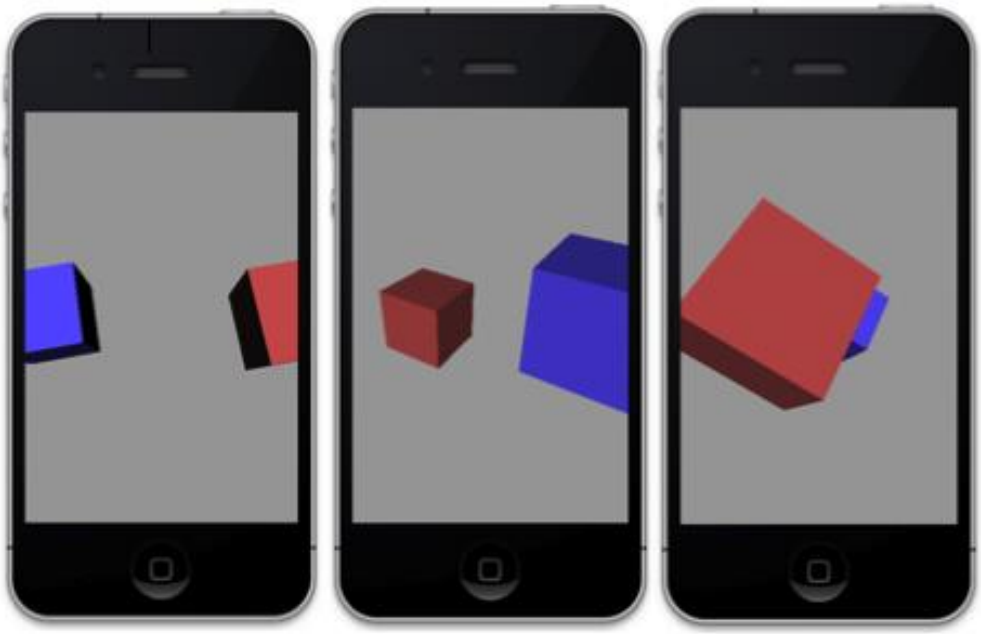


Fig. 2.149: Juego basado en OpenGL en ejecución

Si se desea, se puede analizar el código fuente auto generado de los archivos de sombreado y controlador de la vista para comprender sus métodos y funcionamiento. Lamentablemente, el tema de desarrollo de juegos para *iOS* no se abordará en este tutorial debido a que no fue desarrollado con este fin.

3. Page-Based Application: Plantilla que provee un punto inicial para una aplicación basada en páginas que utiliza un componente *Page View Controller.*

Al crear un proyecto utilizando esta plantilla, constará de los siguientes archivos:

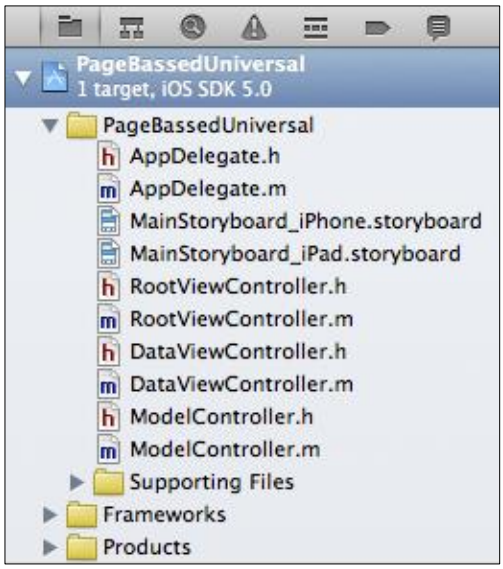


Fig. 2.150: Archivos de una aplicación basada en páginas

- La clase *AppDelegate* (con sus archivos cabecera e implementación).
- Un *Storyboard* para *iPhone* o *Storyboard* para *iPad*; o ambos.
- Dos clases *UIViewController* (con sus archivos cabecera e implementación): una para la manejar la "vista raíz" y otra para manejar la "vista de información".
- Una clase *ModelController* que maneja un modelo simple: una colección de nombres de los meses. Un controlador sirve como fuente de información para el *Page View Controller,* ya que no es necesario crear un *View Controller* para cada página, por que al hacerlo, se produciría una saturación de memoria del dispositivo *iOS.*
- Archivos de soporte, *frameworks* y el producto.

Funcionamiento: El controlador de la vista raíz, contiene el objeto *Page View Controller*. El controlador de la vista de información, en este caso contiene un *Label*

que presenta el nombre del mes y un objeto de información que puede ser de cualquier tipo (tipo *id)*, en este caso es una vista. El controlador del modelo contiene una colección de nombres de los meses y los métodos para crear, configurar y devolver la nueva vista solicitada. A continuación se presenta el *Storyboard* de *iPhone* y de *iPad* de una aplicación basada en páginas.

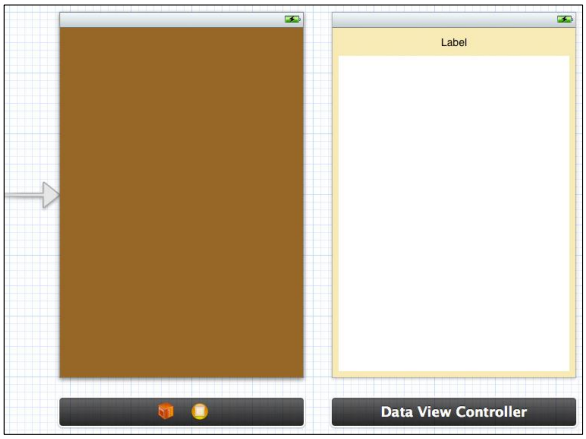


Fig. 2.151: *iPhone Storyboard* de una aplicación basada en páginas

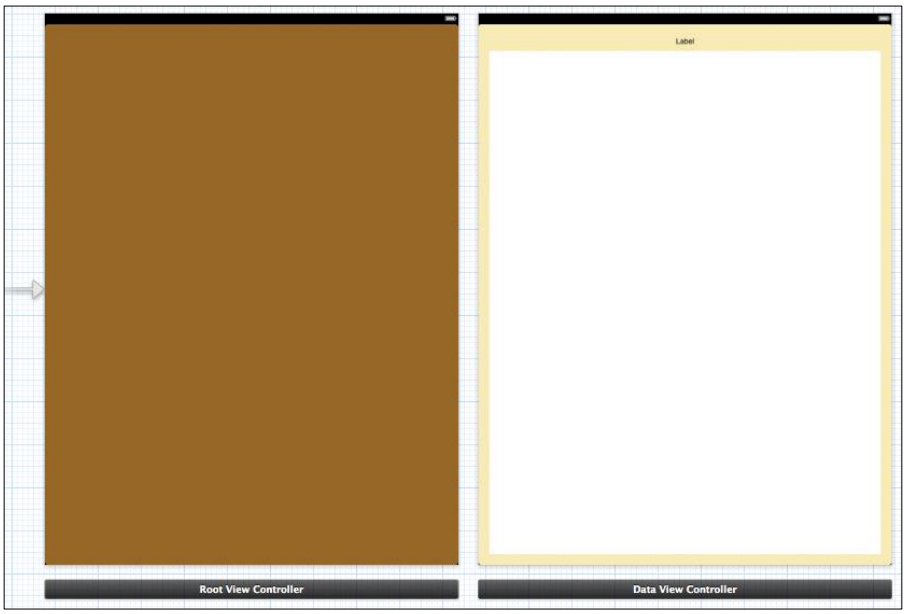


Fig. 2.152: *iPad Storyboard* de una aplicación basada en páginas

Para comprender de mejor manera el funcionamiento de una aplicación basada en páginas, se presenta a continuación la secuencia que tendría la misma al ser ejecutada:

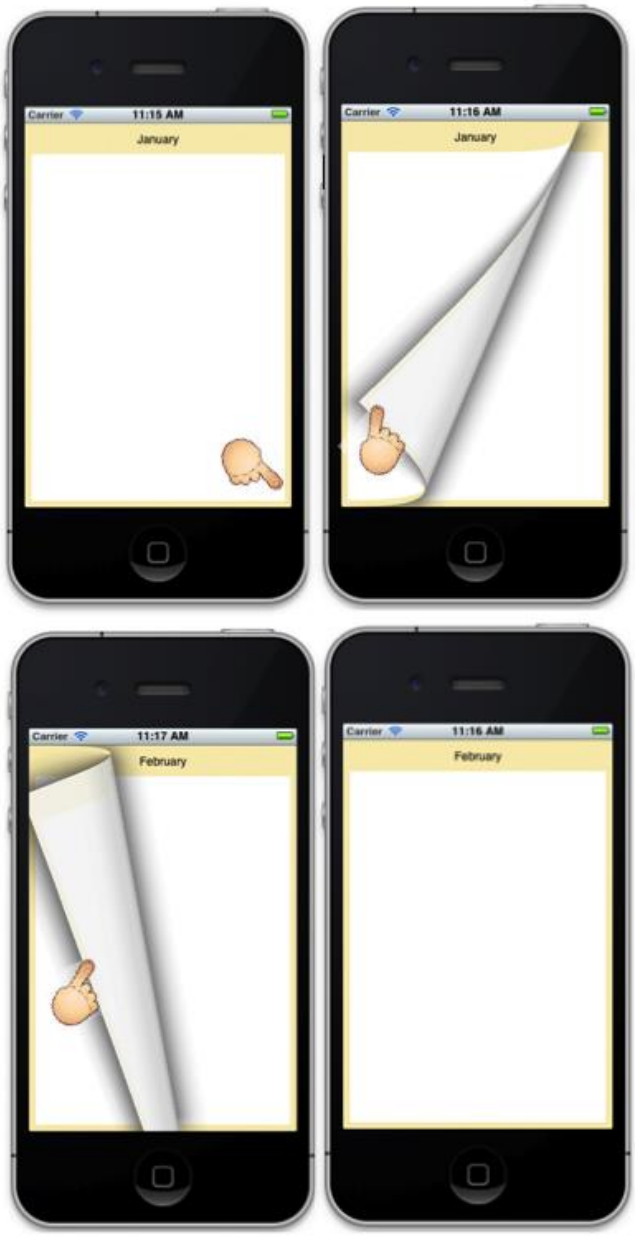


Fig. 2.153: Secuencia de una aplicación basada en páginas

Si se desea, se puede analizar el código fuente auto generado de cada vista para comprender los métodos y el funcionamiento de las mismas y de esta forma poder construir una aplicación basada en páginas personalizada.

4. Single View Application: Plantilla que provee un punto inicial para una aplicación que presenta solo una vista. Provee un *View Controller* para manejar la vista, y un *Storyboard* que contiene la misma.

Al crear un proyecto utilizando esta plantilla, constará de los siguientes archivos:

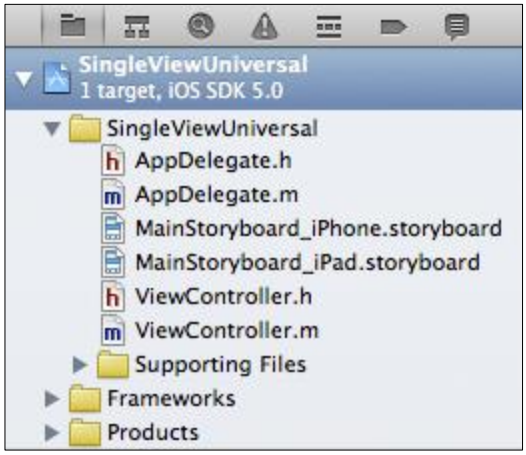


Fig. 2.154: Archivos de una aplicación de una sola vista

- La clase *AppDelegate* (con sus archivos cabecera e implementación).
- Un *Storyboard* para *iPhone* o *Storyboard* para *iPad*; o ambos.
- Una clase *UIViewController* (con su archivos cabecera e implementación) para manejar la vista que se incluye dentro del *Storyboard.*
- Archivos de soporte, *frameworks* y el producto.

Funcionamiento: Una aplicación de una sola vista incluye por defecto un *Storyboard* con un *View Controller* y su respectiva clase controladora para manejar sus posibles eventos y variables. La vista se presenta completamente en blanco tanto para *iPad* como para *iPhone*. La aplicación construida en el apartado "*2.4.2 Construcción de una interface gráfica de usuario*", podría haber sido construida utilizando esta plantilla, debido a que se creó un proyecto vacío y después se agregó un *Storyboard*, un *View Controller* y una clase *UIViewController* que son las que conforman a esta plantilla.

A continuación se presenta el *Storyboard* de una aplicación de una sola ventana tanto para *iPhone* como para *iPad.*

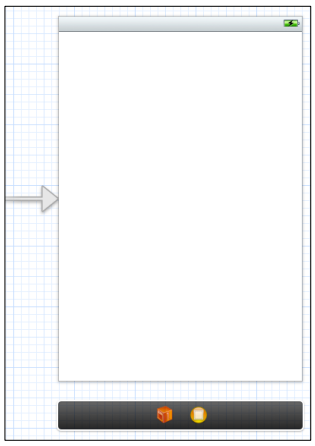


Fig. 2.155: *iPhone Storyboard* de una aplicación de una sola ventana

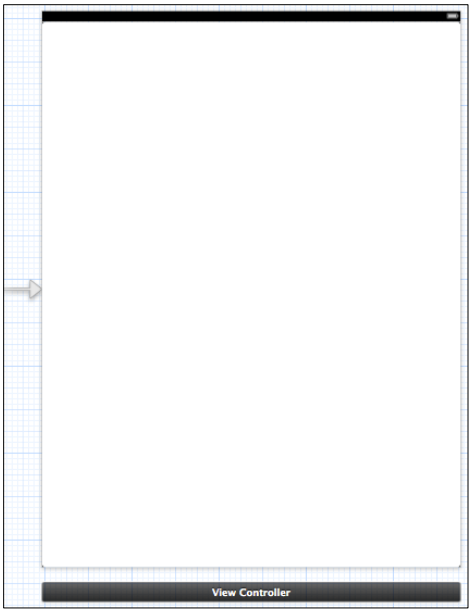


Fig. 2.156: *iPad Storyboard* de una aplicación de una sola ventana

Para esta plantilla no se presenta su secuencia de ejecución debido que al probarla en el simulador *iOS*, se presenta únicamente una ventana en blanco en el dispositivo. No existe código fuente autogenerado por esta plantilla a más de los métodos incluidos por defecto en el archivo de implementación de la clase *UIViewController*.

5. Tabbed Application: Plantilla que provee un punto inicial para una aplicación que usa un *Tab Bar* o Barra de Pestañas. Provee una interface de usuario configurada con un *Tab Bar Controller*, y *View Controllers* para cada elemento del *Tab Bar*.

Al crear un proyecto utilizando esta plantilla, constará de los siguientes archivos:

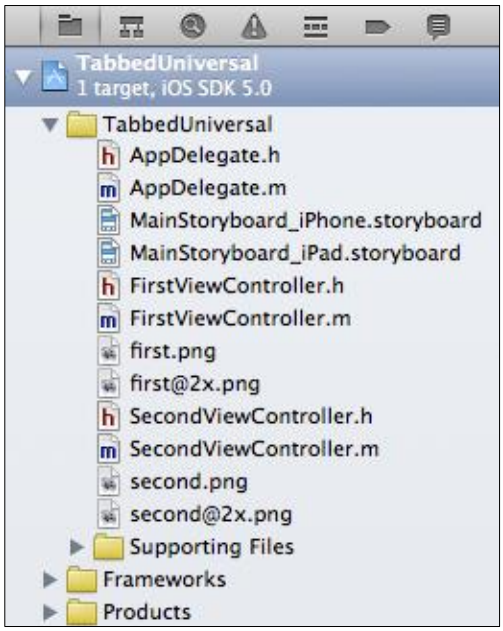


Fig. 2.157: Archivos de una aplicación de barra de pestañas

- La clase *AppDelegate* (con sus archivos cabecera e implementación).
- Un *Storyboard* para *iPhone* o *Storyboard* para *iPad*; o ambos.
- Dos clases *UIViewController* (con sus archivos cabecera e implementación): una para la manejar la "primera vista" y otra para manejar la "segunda vista", debido a que la barra de pestañas constará de dos elementos, es decir dos *View Controllers.*
- Cuatro archivos de imagen, dos en una resolución (*first.png* y *second.png*) y otros dos, que son las mismas imágenes anteriores pero con una resolución superior (*first@2x.png* y *second@2x.png*). Estas imágenes se presentan en los elementos de la barra de pestañas. Las imágenes de resolución inferior se presentan en

dispositivos *iPhone/iPod* y las de mayor resolución cuando la aplicación se ejecuta en dispositivos *iPad*.

- Archivos de soporte, *frameworks* y el producto.

Funcionamiento: Consta de un *Tab Bar Controller* o barra de pestañas ubicada en la parte inferior de la aplicación. La conforman dos pestañas (podrían ser hasta cinco). Estas pestañas son "*First*" y "*Second*", con sus respectivos íconos; un círculo y un cuadrado respectivamente. Al seleccionar alguna de ellas, se presenta el *View Controller* que corresponde a la misma. Al ejecutar la aplicación se presenta el primer *View Controller* que tiene un color de fondo blanco, una etiqueta "*First View*" y otra etiqueta con una leyenda sin importancia. Si el usuario presiona la pestaña "*Second*", se presentará el segundo *View Controller,* que es igual al primero, con la diferencia de que la etiqueta que presenta contiene la leyenda "*Second View*".

A continuación se presenta el *Storyboard* de *iPhone* e *iPad* de una aplicación basada en pestañas.

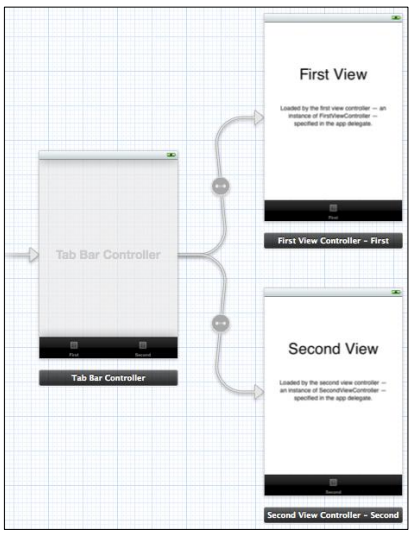


Fig. 2.158: *iPhone Storyboard* de una aplicación basada en pestañas

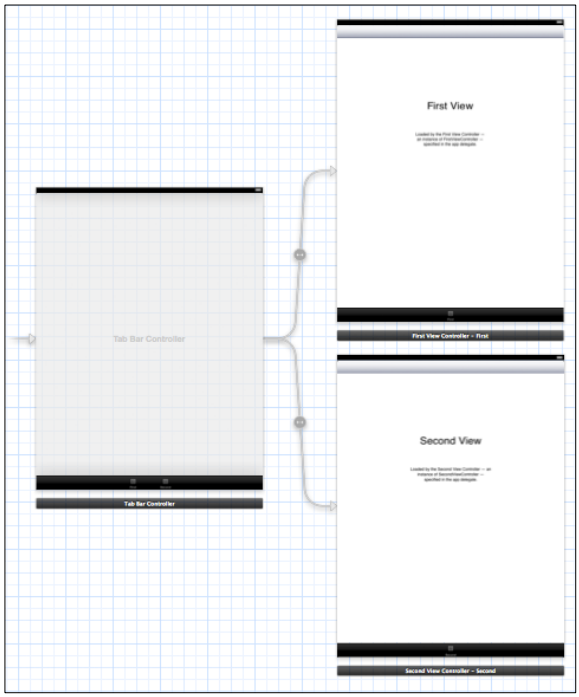


Fig. 2.159: *iPad Storyboard* de una aplicación basada en pestañas

Para comprender de mejor manera el funcionamiento de una aplicación basada en pestañas, se presenta a continuación la secuencia que tendría la misma al ser ejecutada:

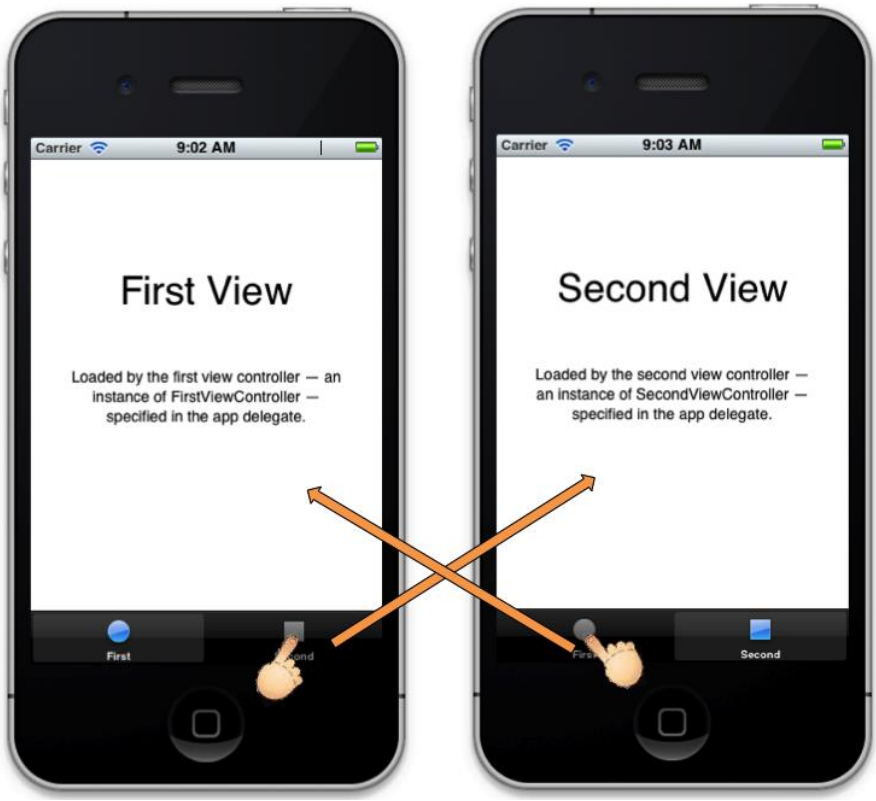


Fig. 2.160: Secuencia de una aplicación basada en pestañas

Gracias al uso de *Storyboards*, no se requiere escribir código fuente para implementar una barra de pestañas, y tampoco para asignar una vista a cada pestaña de esta barra. Se explicará detalladamente cómo implementar una barra de pestañas en una aplicación, cómo asignar las vistas correspondientes y cómo asignar un ícono para cada pestaña en el apartado "*2.6.2 Implementación de un Tab Bar Controller*".

6. Utility Application: Plantilla que provee un punto inicial para una aplicación de utilidades que tiene una vista principal y una vista secundaria. Para *iPhone*, se configura un botón de información para saltar de la vista principal a la vista secundaria. En el caso de *iPad*, se configura un botón de barra de información que muestra la vista secundaria en una vista sobrepuesta a la vista principal.

Al crear un proyecto utilizando esta plantilla, constará de los siguientes archivos:

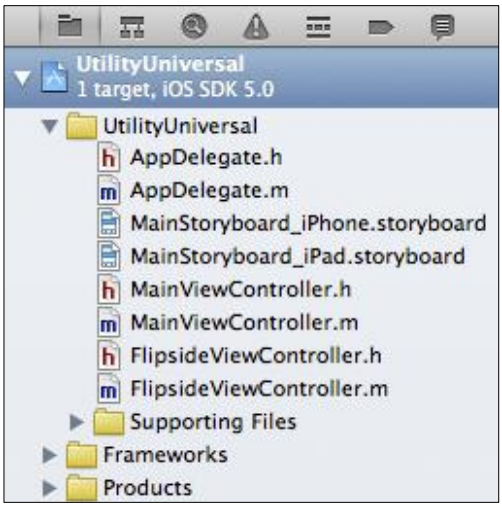


Fig. 2.161: Archivos de una aplicación de utilidades

- La clase *AppDelegate* (con sus archivos cabecera e implementación).
- Un *Storyboard* para *iPhone* o *Storyboard* para *iPad*; o ambos.

- Dos clases *UIViewController* (con sus archivos cabecera e implementación): una para la manejar la "vista principal" y otra para manejar la "vista volteada", que es llamada al presionar un botón que se encuentra en la "vista principal".
- Archivos de soporte, *frameworks* y el producto.

Funcionamiento: En el caso de *iPhone*, se presenta la vista principal, con un color de fondo negro y un único botón de información; al presionarlo, la vista realiza una transición de volteado para presentar su parte posterior, la cual es la "vista volteada" con un único botón (*Done)* que sirve para regresar a la vista principal. A continuación se presenta el *Storyboard* de *iPhone* de una aplicación de utilidades.

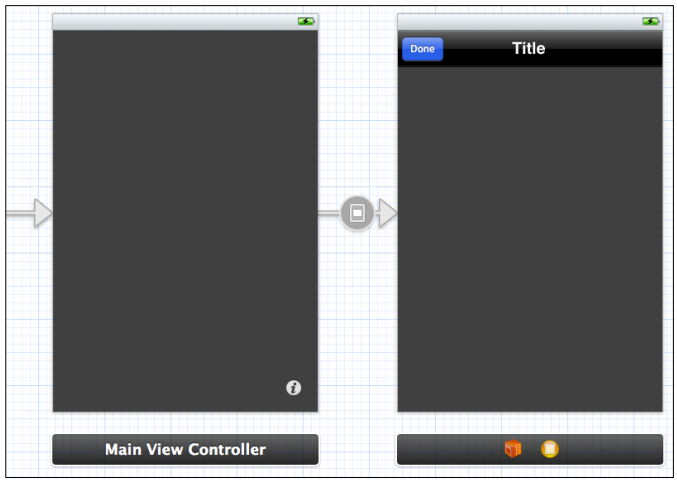


Fig. 2.162: *iPhone Storyboard* de una aplicación de utilidades

Para comprender de mejor manera el funcionamiento de una aplicación de utilidades en un dispositivo *iPhone*, se presenta a continuación la secuencia que tendría la misma al ser ejecutada.

Fig. 2.163: Secuencia de una aplicación de utilidades en *iPhone*

En el caso de *iPad*, las vistas tienen la misma funcionalidad, pero la forma de presentación es distinta: la vista principal implementa una barra superior con un botón que al presionarlo permite que la vista volteada aparezca y desaparezca sobrepuesta sobre la misma.

A continuación se presenta el *Storyboard* de *iPad* de una aplicación de utilidades.

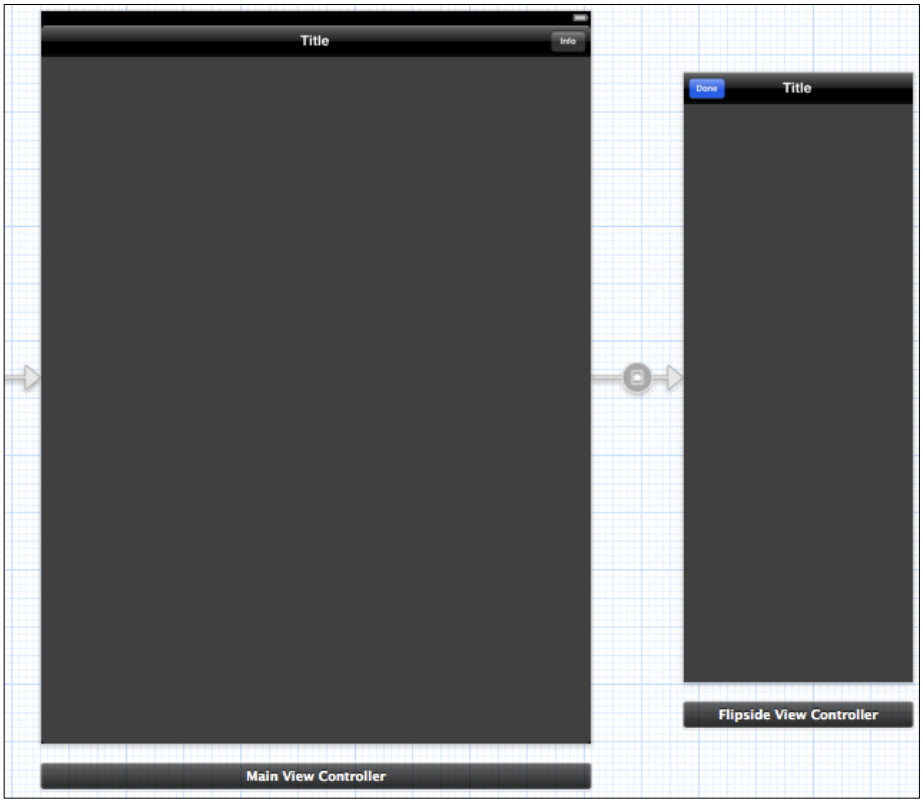


Fig. 2.164: *iPad Storyboard* de una aplicación de utilidades

Para comprender de mejor manera el funcionamiento de una aplicación de utilidades en un dispositivo *iPad*, se presenta a continuación la secuencia que tendría la misma al ser ejecutada.

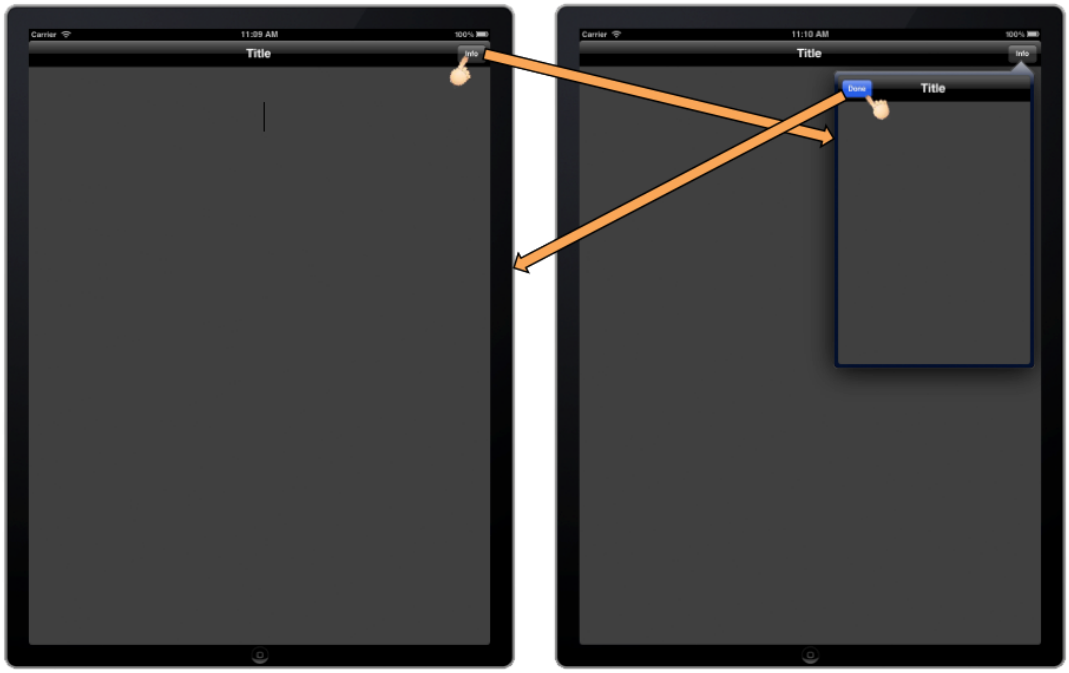


Fig. 2.165: Secuencia de una aplicación de utilidades en *iPad*

Si se desea, se puede analizar el código fuente autogenerado de cada vista para comprender los métodos y el funcionamiento de las mismas. Esto, con el objetivo de comprender la forma de llamar a una segunda vista a partir de una primera, y; para comprender las transiciones con las que se puede cambiar de una vista a otra. Se explicará detalladamente los distintos tipos de transiciones existentes en el apartado "*2.6.1 Análisis de tipos de transiciones*".

7. Empty Application: Plantilla que provee un punto inicial para cualquier tipo aplicación. Provee únicamente un *AppDelegate*.

Al crear un proyecto utilizando esta plantilla, constará de los siguientes archivos:

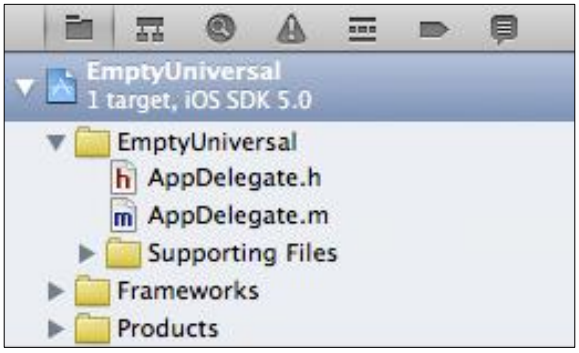


Fig. 2.166: Archivos de una aplicación vacía

- La clase *AppDelegate* (con sus archivos cabecera e implementación).
- Archivos de soporte, *frameworks* y el producto.

Funcionamiento: Al ejecutar la aplicación se presenta una vista completamente vacía de color blanco.

Fig. 2.167: Aplicación vacía

No existe ningún *Storyboard,* la vista es instanciada mediante código fuente escrito en la clase *AppDelegate*.

A partir de esta plantilla, se puede desarrollar cualquier tipo de aplicación. Un ejemplo claro, se puede observar en el apartado "*2.4.2 Construcción de una interface gráfica de usuario*", en el cual, a partir de esta plantilla, se agregó un *Storyboard*, se construyó una interface gráfica de usuario y se agregó una clase para controlar la única vista.

2.4.4 Rotación de interfaces y sus componentes

Una característica muy atractiva de los *Smartphones* en la actualidad es el acelerómetro o capacidad de detección del movimiento. Las aplicaciones son mas agradables cuando presentan esta característica; por esto, a continuación se explicará como se pueden rotar las interfaces gráficas de usuario.

Paso 1: Se creará un nuevo proyecto, usando la plantilla "*Single View Application*" del apartado anterior, y así, se evitará tener que agregar un *Storyboard*, una vista y su clase controladora. Para esto se debe abrir *XCode* y presionar el botón "*Create a new XCode project*" de la Fig. 2.27. Seleccionar la categoría *iOS / Application* para ver las plantillas prediseñadas (Fig. 2.142). Se debe seleccionar "*Single View Application*" y continuar de la misma forma en la que se creaba un nuevo proyecto vacío; el proyecto será nombrado "*RotationExample*". Si se requiere ayuda, se puede referir al apartado "*2.2 Cómo crear un nuevo proyecto*".

Las imágenes necesarias para este apartado se encuentran dentro del comprimido "*imagenesRotationExample.zip*", por lo que se debe descomprimir el mismo y agregar estas imágenes dentro del proyecto. Si se requiere ayuda, se puede referir al apartado *"2.3.1 Cómo agregar un fichero a un proyecto"* específicamente al subapartado *"c) Agregar un archivo de imagen"*. Finalizar agrupando los ficheros del proyecto como se indica en la Fig. 2.168.

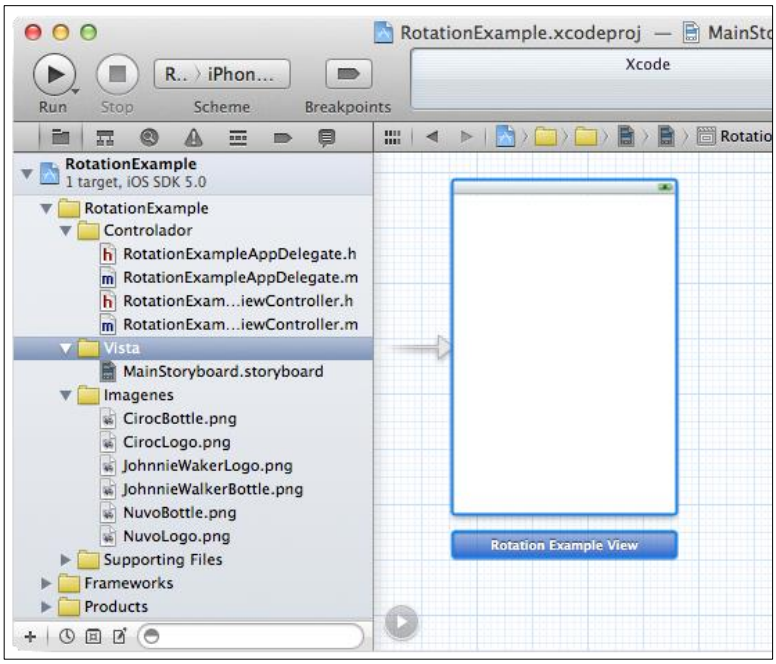


Fig. 2.168: Proyecto *RotationExample*

Paso 2: El siguiente paso consiste en configurar la aplicación para que sea capaz de responder a las orientaciones deseadas. Para esto, primero se presentan las cuatro posibles orientaciones del dispositivo:

Fig. 2.169: Orientaciones de un dispositivo *iOS*

Se desea que la aplicación responda a tres de las orientaciones: *Portrait*, *Landscape Left* y *Landscape Right*, debido a que la orientación *Upside Down* es poco común y no será la tomada en cuenta. Para configurar la aplicación se debe seleccionar el proyecto *RotationExample* en el Área de Navegación y seleccionar cada una de las orientaciones deseadas como se indica en la Fig. 2.170.

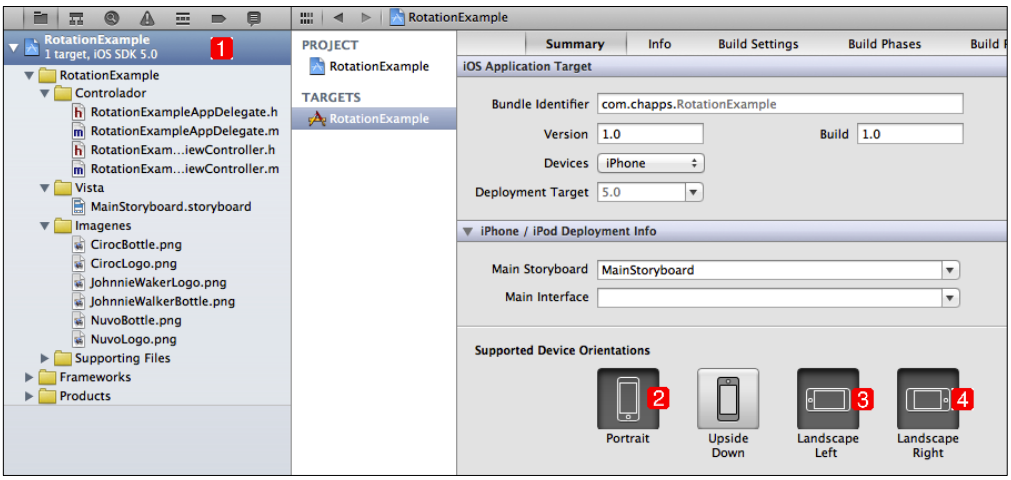


Fig. 2.170: Seleccionar las orientaciones deseadas para la aplicación

Paso 3: Construir una interface gráfica de usuario sobre la vista del *Storyboard*. Agregar seis *Round Rect Buttons* de tipo "*Custom*" y dimensiones 111*px* de ancho por 85*px* de alto*;* y asignarles las imágenes agregadas. El color de fondo de la vista es "*Light Gray Color*". Si se requiere ayuda, se debe referir al apartado "*2.4.2 Construcción de una interface gráfica de usuario*". Al finalizar este paso, la interface construida debería verse como la presentada a continuación.

Fig. 2.171: Interface gráfica de usuario de *RotationExample*

La idea es la siguiente: La aplicación presenta el logotipo de un licor en la parte izquierda y su botella de presentación correspondiente en la parte derecha. Cuando se rote el dispositivo, se debería mantener este orden, como se presenta en la siguiente figura.

Fig. 2.172: Objetivo de la aplicación *RotationExample*

Una vez construida la interface de usuario de la Fig. 2.171, se puede proceder a ejecutar la aplicación en el simulador *iOS*. Se presenta normalmente en orientación *Portrait*. Para visualizarla en modo *Landscape*, se debe dirigir a *Hardware* → Girar a la derecha / Girar a la izquierda, como se indica a continuación.

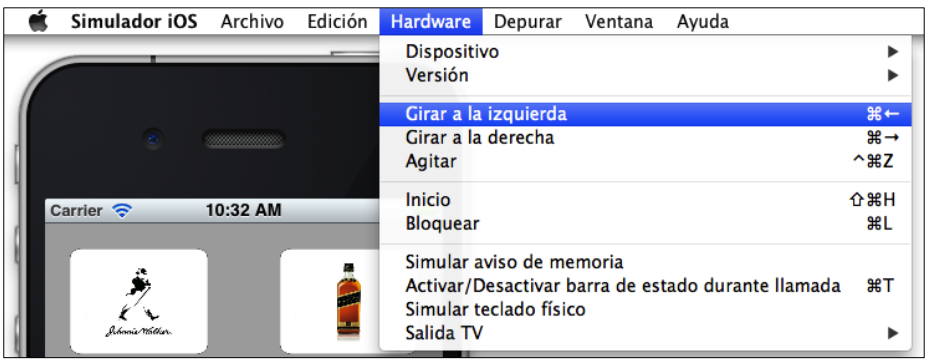


Fig. 2.173: Cambiar la orientación del *iOS Simulator*

Al visualizar la aplicación en modo *Landscape* se puede observar que existe un problema; algunos de los componentes de la *GUI*, han desaparecido como se puede observar en la Fig. 2.174. Este problema será corregido en el siguiente paso.

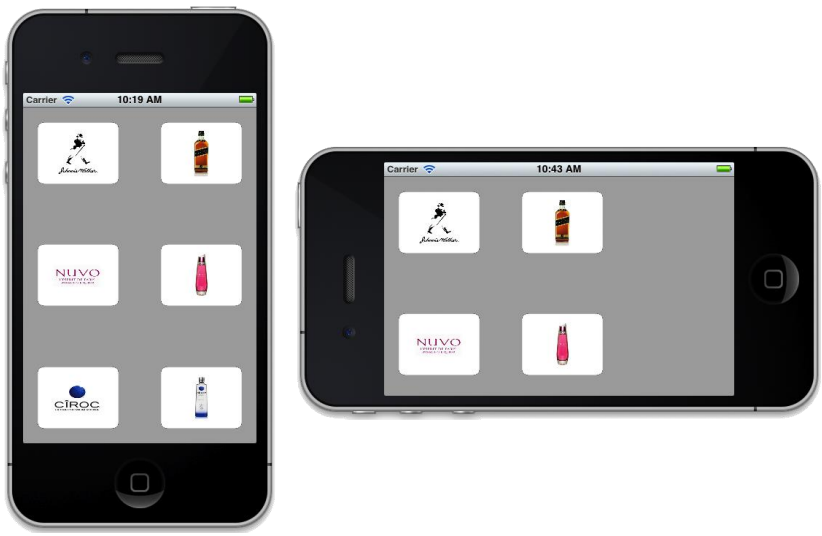


Fig. 2.174: Componentes desaparecidos al girar el dispositivo *iOS*

Paso 4: Para rotar correctamente la vista, se debe hacer un ajuste a nivel de código. Las clases controladoras de las vistas de un *Storyboard* que requieran ser rotadas, deben implementar el método *shouldAutorotateToInterfaceOrientation:*. Este método es el encargado de la rotación automática e indica que orientaciones soportará la vista controlada por la misma.

Para implementar o modificar este método, se debe seleccionar el archivo de implementación de la clase *RotationExampleViewController*, y analizar su código fuente. Se debe buscar el método *shouldAutorotateToInterfaceOrientation:*. Si no se lo encuentra, debe ser implementado. En este caso, se lo ha encontrado y se lo ha marcado en la siguiente figura:

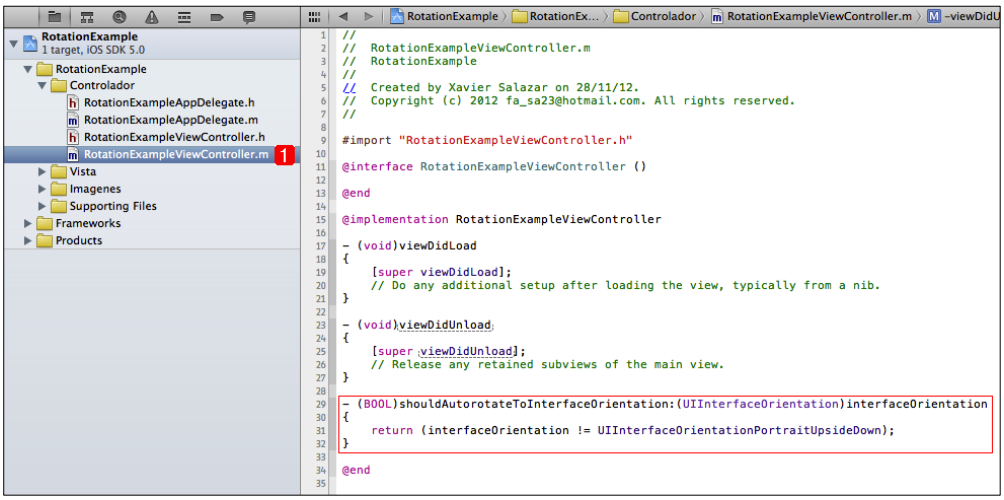


Fig. 2.175: Métodos del fichero *RotationExampleViewController.m*

Como se puede observar, el método devuelve un valor de tipo *BOOL*. Lo que se debe hacer es devolver "si" o (*YES*) para las orientaciones deseadas y "no" (*NO*) para las orientaciones no soportadas por la aplicación o por la vista específicamente.

Si dentro de este método se escribe la sentencia *"return YES"*, la vista soportará todas las orientaciones posibles; pero en este caso, se requiere soportar todas las orientaciones excepto "*Upside Down*" es por esto que el contenido del método será el siguiente:

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
    interfaceOrientation
{
    return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
}
```

Fig. 2.176: Contenido del método shouldAutorortateToInterfaceOrientation:

La sentencia "*return (interfaceOrientation ¡= UIInterfaceOrientationPortraitUpsideDown)*" significa que el método devolverá *YES* para todas las orientaciones (o las soportará) excepto para la orientación *Upside Down*.

Se debe implementar o modificar este método como se muestra en las Fig. 2.175 y Fig. 2.176 dentro del archivo correspondiente antes de proseguir.

A continuación se debe trabajar con cada objeto de la vista para prepararlos para la rotación automática. Para esto se debe seleccionar el *Storyboard* y seleccionar después el primer botón de la vista cuya imagen es el logotipo del licor "*Johnnie Walker*".

Después de seleccionar este botón, en la barra de selección de inspector de la Fig. 2.113, se debe presionar el botón para mostrar los atributos de tamaño y rotación del objeto () que son los presentados a continuación:

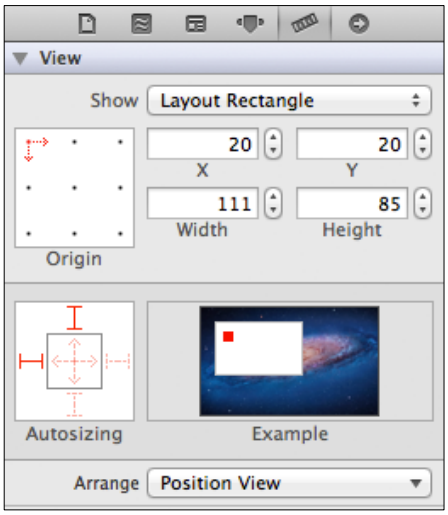


Fig. 2.177: Propiedades de tamaño del objeto

Lo que interesa ahora es el apartado de auto ajuste de tamaño o "*autosizing*" de los componentes, presentado en la Fig. 2.177 y enfocado en la Fig. 2.178, a partir de la cual se explicará su funcionamiento.

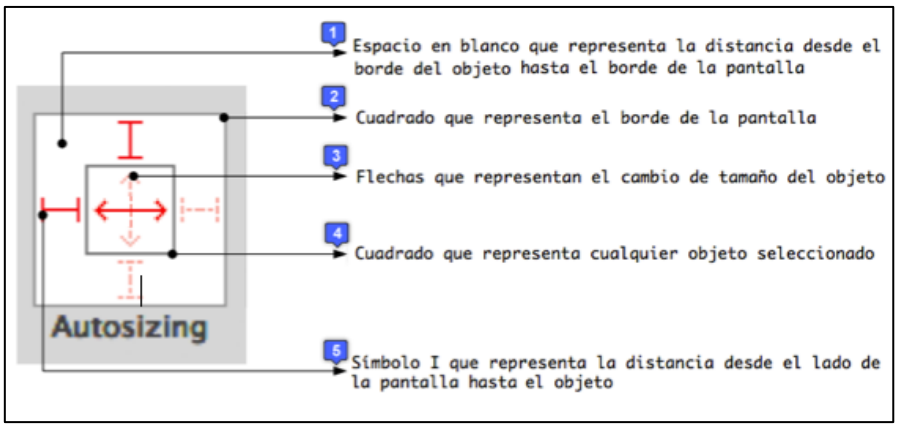


Fig. 2.178: Propiedad de auto ajuste de tamaño de componentes *iOS*

Propiedad *"autosizing":* Es una propiedad que permite auto ajustar la posición y tamaño de un componente seleccionado. Al seleccionar un componente, es representado por el recuadro interno (4) de la figura anterior. Las flechas internas del recuadro o componente (3) sirven para auto ajustar su tamaño a medida que se rota el dispositivo *iOS*. Los "símbolos I" (5) sirven para bloquear el espacio entre el componente y el borde deseado de la pantalla. Cuando las flechas y los "símbolos I" no se encuentran seleccionados son entrecortados, como el "símbolo I" de la parte inferior y la flecha vertical de la Fig. 2.178; al seleccionar cualquiera de los mismos, se marcan completamente de color rojo, como la flecha horizontal y el "símbolo I" de la parte superior de la figura antes mencionada. Para comprender mejor el funcionamiento de esta propiedad, a continuación se presenta una tabla de ejemplos de la variación de componentes de acuerdo a cambios en el patrón de *autosizing*:

Ejemplo #	Patrón	Muestra Portrait	Muestra Landscape	Portrait	Landscape
1					
2					
3					
4					
5					
6					
7					
8					
9					

Fig. 2.179: Tabla de ejemplos de la funcionalidad *autosizing*

A continuación se analizarán los ejemplos de la tabla de la figura anterior, en la cual, se modifica el patrón de *autosizing* para cada caso y se observa el comportamiento un objeto en una vista en orientación *Portrait* y *Landscape*. La tercera y cuarta columna de la tabla son una vista previa del comportamiento del objeto, localizado junto al patrón de *autosizing* como se observa en la Fig. 2.177.

Ejemplo #1: Ningún elemento seleccionado en el patrón. El patrón de *autosizing* no ha sido modificado. Por esto, el objeto en la vista conserva su misma posición en orientación *Portrait* y *Landscape.*

Ejemplo #2: Selección de un solo "símbolo I" del patrón. Al seleccionar el "símbolo I" superior del patrón de *autosizing,* se bloquea la distancia entre el objeto y la parte superior de la vista. Al rotar el dispositivo, el objeto y la parte superior de la vista mantienen la misma distancia.

Ejemplo #3, #4 y #5: Selección de un solo "símbolo I" del patrón. Estos ejemplos son iguales al anterior, con la única diferencia de que se bloquea la distancia entre el objeto y los otros lados de la vista: el lado izquierdo, derecho e inferior en el ejemplo #3, #4 y #5 respectivamente.

Ejemplo #6: Selección de una sola flecha del patrón. Al seleccionar la flecha vertical del patrón de *autosizing,* el objeto cambia de tamaño cuando se rota el dispositivo para mantener la misma distancia con el borde inferior y superior de la vista en cualquiera de las orientaciones. Al rotar el dispositivo a orientación *Landscape*, el componente se reduce, y mantiene la misma distancia que tenía con la parte superior e inferior de la vista en orientación *Portrait*.

Ejemplo #7: Selección de una sola flecha del patrón. Al seleccionar la flecha horizontal en el patrón de *autosizing,* el objeto cambia de tamaño cuando se rota el dispositivo para mantener la misma distancia con el borde izquierdo y derecho de la

vista en cualquiera de las orientaciones. Al rotar el dispositivo a orientación *Landscape*, el objeto se agranda, y mantiene la misma distancia que tenía con la parte izquierda y derecha de la vista en orientación *Portrait*.

Ejemplo #8: Selección de varios elementos del patrón. Al seleccionar las flechas horizontal y vertical, el objeto cambia de tamaño cuando se rota el dispositivo para mantener la misma distancia con todos los bordes de la vista en cualquiera de las orientaciones. Al rotar el dispositivo a orientación *Landscape*, el objeto se encoge y ensancha, y mantiene la misma distancia que tenia con los cuatro bordes de la vista en orientación *Portrait.*

Ejemplo #9: Selección de varios elementos del patrón. Al seleccionar los "símbolos I" superior y derecho, se bloquea la distancia entre el objeto y el borde superior derecho de la vista. Al rotar el dispositivo, el objeto mantiene la misma distancia con el borde superior derecho de la vista.

Como se puede observar, el reposicionamiento de componentes de una interface al rotar un dispositivo depende del patrón de *autosizing* que se le asigne.

Para continuar, se debe seleccionar cada componente de la interface construida y asignarle el patrón de *autosizing* correspondiente, como se indica en la siguientes tablas:

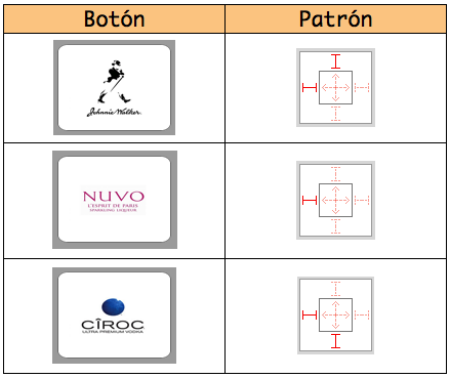


Fig. 2.180: Tabla de patrones de *autosizing*

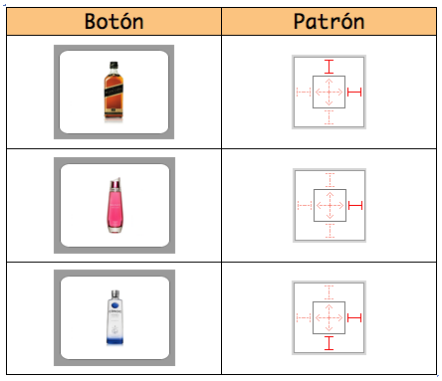


Fig. 2.181: Tabla de patrones de *autosizing*

Una vez finalizado este proceso, la interface de usuario en modo *Portrait* y *Landscape* es igual a la de la Fig. 2.172.

Importante: Al usar el patrón *autosizing*, se puede producir el solapamiento de componentes (Fig. 2.182) por lo que es esencial conocer el método de posicionamiento de objetos mediante pixeles.

Posicionamiento de objetos mediante pixeles: Si se agrandan las dimensiones de los botones, al ejecutar la aplicación y rotar el dispositivo, se puede observar que éstos se solapan.

Fig. 2.182: Componentes solapados al girar el dispositivo *iOS*

Para que los botones no se solapen cuando se rote el dispositivo a orientación *Landscape*, se mostrará el logotipo del licor en la parte superior y su botella de presentación en la parte inferior de la vista. Como se puede observar a continuación:

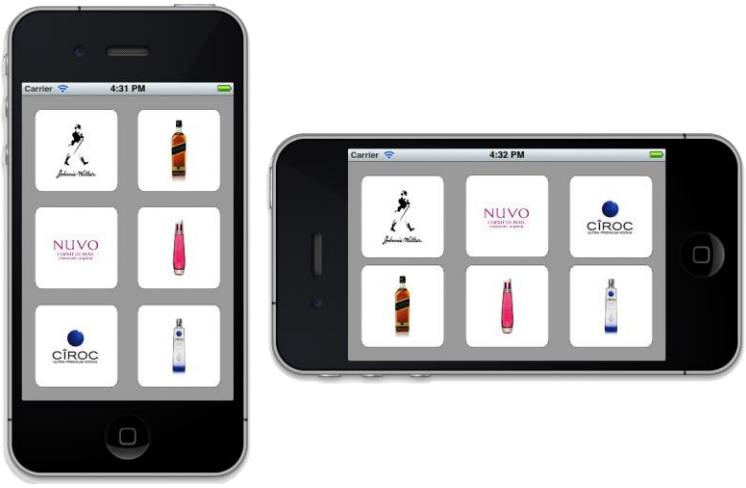


Fig. 2.183: Presentación de los botones en diferentes orientaciones del dispositivo

Para esto, se debe indicar mediante código fuente el lugar exacto dentro de la vista en donde se desea que este situado cada botón. Para empezar se debe tomar nota de las dimensiones y distancia con el eje x y eje y de la vista de cada botón; tanto en orientación *Portrait* como *Landscape*. Para conocer las dimensiones exactas, se debe seleccionar el botón deseado, y ver los atributos de tamaño del mismo (Fig. 2.177). Como se indica a continuación:

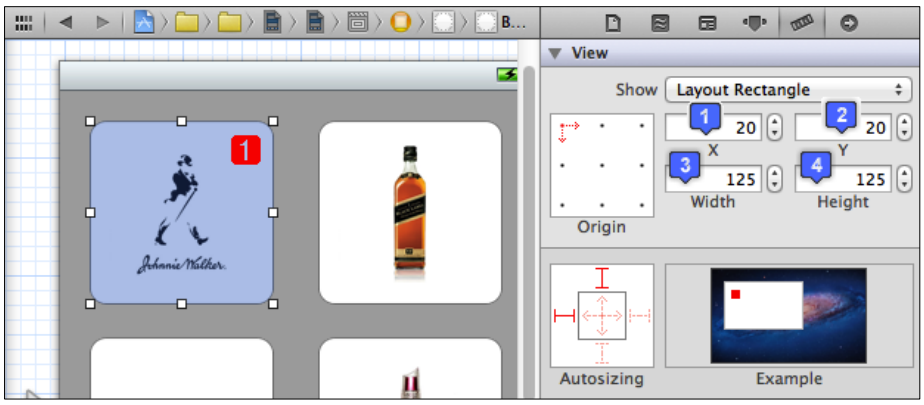


Fig. 2.184: Selección de un botón y visualización de sus atributos de tamaño

Al tomar nota de los valores de la figura anterior para cada componente, se obtiene la siguiente tabla:

Orientación	Botón	Dist. Eje X	Dist. Eje Y	Ancho	Alto
Portrait		20	20	125	125
		175	20	125	125
		20	167	125	125
		175	167	125	125
		20	315	125	125
		175	315	125	125

Fig. 2.185: Tabla de tamaño y posición de cada componente de la interface en orientación *Portrait*

Ahora, se debe hacer lo mismo para cada componente pero en orientación *Landscape*; para esto, se debe seleccionar la vista y, en sus atributos, en el apartado *Orientation* seleccionar *Landscape*, como se indica a continuación:

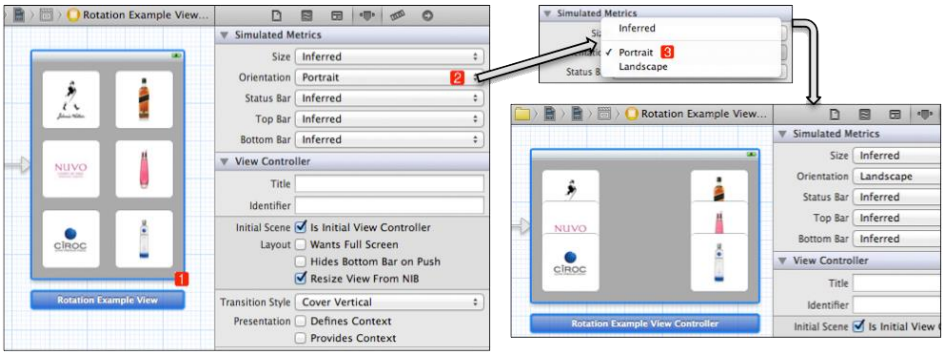


Fig. 2.186: Ajustar la orientación de una vista en un *Storyboard*

Como se puede observar, los botones se solapan; así que deben ser reposicionados como se indica en la Fig. 2.183 en la vista del *Storyboard* en orientación *Landscape*.

Acto seguido se debe realizar el mismo procedimiento anterior: seleccionar cada botón y tomar nota de sus dimensiones y distancia con el eje x y eje y de la vista como se indica en la Fig. 2.187.

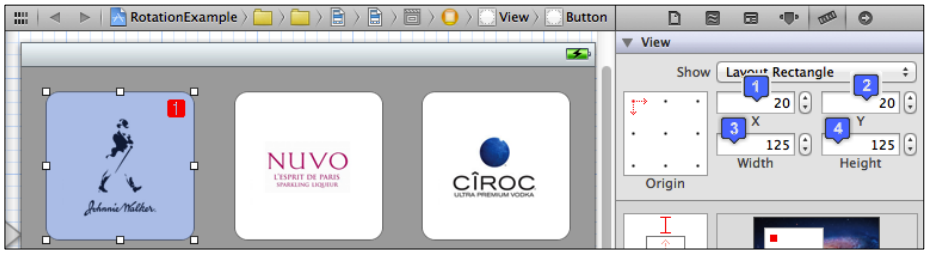


Fig. 2.187: Selección de un botón y visualización de sus atributos de tamaño.

Al tomar nota de los valores de la figura anterior para cada componente, se obtiene la siguiente tabla:

Orientación	Botón	Dist. Eje X	Dist. Eje Y	Ancho	Alto
Landscape		20	20	125	125
		20	155	125	125
		178	20	125	125
		178	155	125	125
		335	20	125	125
		335	155	125	125

Fig. 2.188: Tabla de tamaño y posición de cada componente de la interface en orientación *Landscape*

El siguiente paso consiste en posicionar los botones de acuerdo a la orientación del dispositivo mediante código fuente, empleando los valores de las tablas anteriores.

Para esto, se debe asignar una variable a cada botón como se indica en la siguiente tabla. Si se requiere ayuda, se puede referir al apartado *"2.5.1 Asignación de variables a componentes de GUI"*.

Botón	Variable
	buttonJohnnieWalkerLogo
	buttonJohnnieWalkerBottle
	buttonNuvoLogo
	buttonNuvoBottle
	buttonCirocLogo
	buttonCirocBottle

Fig. 2.189: Variables que se deben asignar a cada botón

A continuación se debe implementar un método dentro del archivo *RotationExampleViewController.m*. Para lo cual, se lo debe seleccionar como se indica a continuación.

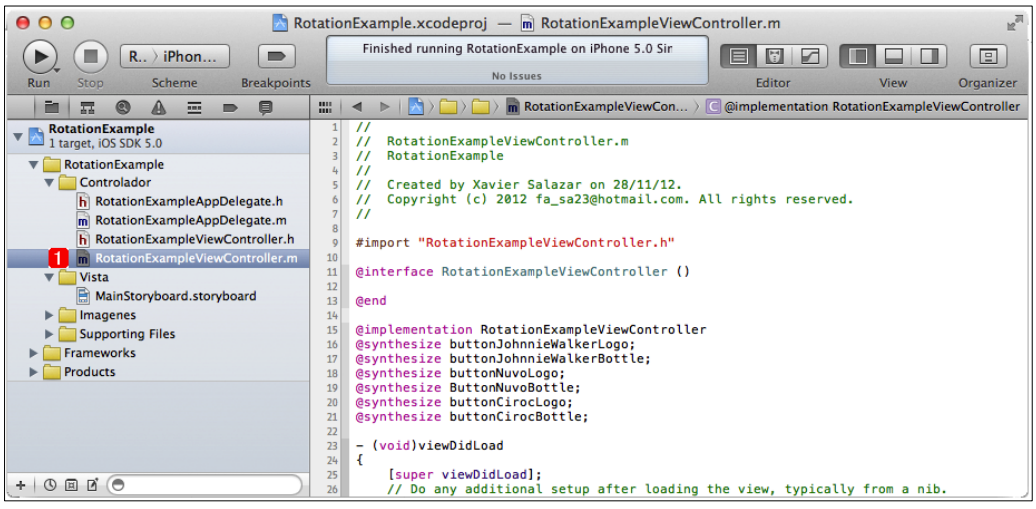


Fig. 2.190: Selección del archivo en donde se implementará el método

El método *willAnimateRotationToInterfaceOrientation:duration:* que se implementará, es llamado después de haber rotado el dispositivo *iOS* y lo que hace es detectar la orientación en la que se está sosteniendo al dispositivo; y, dependiendo de esto, presenta

los objetos de la interface de cierta manera. Al implementar este método se puede indicar la localización exacta de donde y cuando se desea posicionar cada objeto.

A continuación se debe dirigir a la parte inferior de este archivo y antes de la sentencia "*@end*" de la parte final escribir "*-will*" para que *XCode* presente todos los métodos que empiezan con esta palabra. Para implementar un método se debe hacer doble *click* sobre el mismo. El método a implementar es el siguiente:

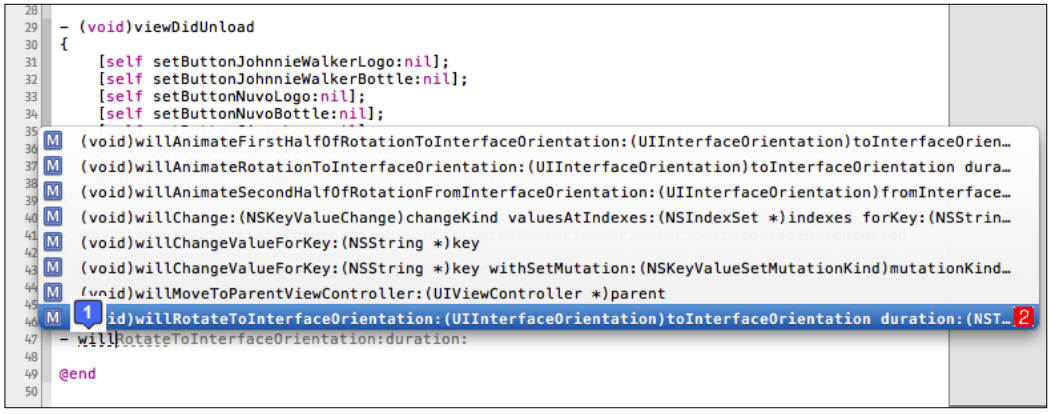


Fig. 2.191: Métodos que inician con la palabra *"-will"*

Una vez seleccionado este método se puede codificar dentro del mismo, es decir, dentro de sus llaves " { } " y debería verse de la siguiente manera:

```
- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)
    toInterfaceOrientation duration:(NSTimeInterval)duration{



}
```

Fig. 2.192: Método seleccionado

Como se puede ver, el método recibe un parámetro de tipo *UIInterfaceOrientation*, que indica a que orientación se rotará al dispositivo.

El método contendrá una condición "*if… else…*" en donde se indicará que si la orientación es *Portrait*, se debe presentar los botones en una posición, caso contrario (*Landscape Left* o *Landscape Right*), se debe reposicionar los mismos de forma

adecuada, apoyándose en los datos de las tablas de la Fig. 2.185 y Fig. 2.188. Se escribirá lo siguiente:

```
- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)
    toInterfaceOrientation duration:(NSTimeInterval)duration{

    if (toInterfaceOrientation == UIInterfaceOrientationPortrait){
        buttonJohnnieWalkerLogo.frame = CGRectMake(20, 20, 125, 125);
        buttonJohnnieWalkerBottle.frame = CGRectMake(175, 20, 125, 125);
        buttonNuvoLogo.frame = CGRectMake(20,167,125,125);
        ButtonNuvoBottle.frame = CGRectMake(175,167,125,125);
        buttonCirocLogo.frame = CGRectMake(20,315,125,125);
        buttonCirocBottle.frame = CGRectMake(175,315,125,125);
    }
    else {
        buttonJohnnieWalkerLogo.frame = CGRectMake(20,20, 125, 125);
        buttonJohnnieWalkerBottle.frame = CGRectMake(20,155,125,125);
        buttonNuvoLogo.frame = CGRectMake(178,20,125,125);
        ButtonNuvoBottle.frame = CGRectMake(178,155,125,125);
        buttonCirocLogo.frame = CGRectMake(335,20,125,125);
        buttonCirocBottle.frame = CGRectMake(335,155,125,125);
    }

}
```

Fig. 2.193: Método de reposicionamiento implementado

La propiedad *frame* de un botón, es el marco contenedor del mismo. La función *CGRectMake* es parte del *framework CoreGraphics* y en este caso es ocupada para dibujar objetos rectángulo invisibles; cada uno de los cuales contendrá un botón de la interface. Esta función consta de cuatro parámetros: distancia en X, distancia en Y, ancho y alto del rectángulo contenedor del botón respectivamente[53], como se indica en la Fig. 2.193. Ya se conocen estos datos del botón de acuerdo a la orientación, debido a que los mismos fueron almacenados previamente en las tablas de la Fig. 2.185 y Fig. 2.188.

Importante: Para posicionar los objetos de esta forma, es decir, mediante pixeles, es importante que el patrón *autosizing* de cada componente sea el patrón por defecto; presentado en la Fig. 2.177.

[53] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p126

2.5 Asignación de variables y eventos a componentes en *View Controllers* y *Table View Controllers*

Una variable es un espacio reservado en memoria, cuyo contenido puede "variar" mientras se ejecute la aplicación. Las variables contienen o apuntan a valores de tipo determinado, por lo que sus operaciones y métodos son determinados de acuerdo al tipo de dato en cuestión. En *Objective C* las variables pueden ser incluso de tipo "*id*", lo cual significa que éstas pueden contener cualquier tipo de dato.

Por otra parte, los eventos son acciones reconocidas por los objetos *GUI*; se activan como resultado de la interacción del usuario con los mismos.

Una vez que se ha aprendido a construir interfaces gráficas de usuario con los componentes deseados, corresponde conocer como asignar variables y eventos a cada componente, de manera de permitir la interacción del usuario con la interface gráfica.

2.5.1 Asignación de variables a componentes de GUI

El objetivo de asignar una variable a un componente *GUI* es el de poder manejar sus propiedades a través de código fuente. De esta forma se podría por ejemplo ocultar un botón "Registrar nuevo usuario" mientras el usuario no haya llenado correctamente los campos necesarios para el registro.

El proceso para asignar una variable a un componente *GUI* es relativamente sencillo. Para explicar este proceso, se creará un proyecto nuevo llamado *VariablesExample* usando la plantilla "*Single View*"; si se requiere ayuda se debe referir al paso 1 del apartado "*2.4.4. Rotación de interfaces y sus componentes*".

A continuación se agregará un objeto *Label* al *View Controller* del *Storyboard;* ajustar sus dimensiones a 280 pixeles de ancho por 135 pixeles de alto; y, centrar el *Label* en su contenedor con la propiedad "*alignment*" como se indica a continuación:

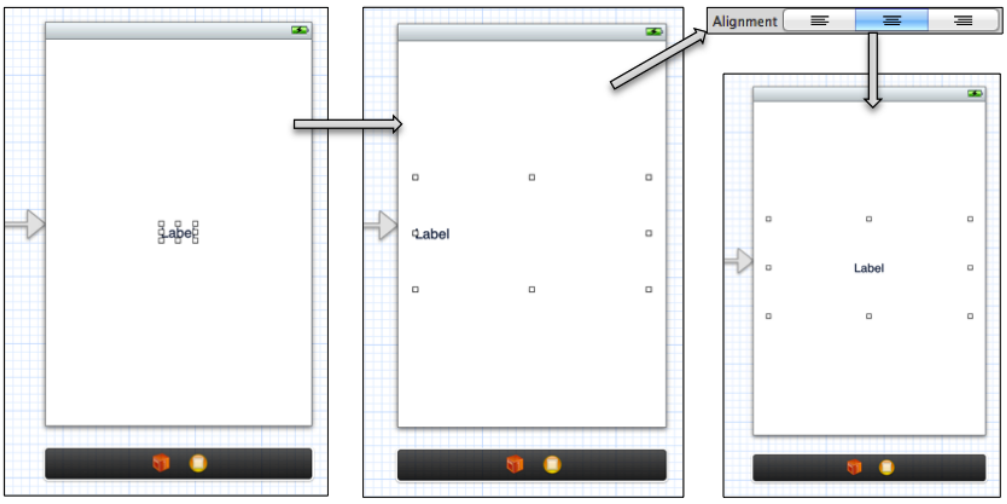


Fig. 2.194: Agregar un *Label* sobre el *View Controller* y ajustarlo

Acto seguido se asignará una variable al objeto *Label,* y se modificarán sus propiedades a través de código fuente. Para esto, se debe mostrar el "*Assistant editor*" ( ), que puede ser localizado en la parte superior del entorno de trabajo en la sección "*Editor*".

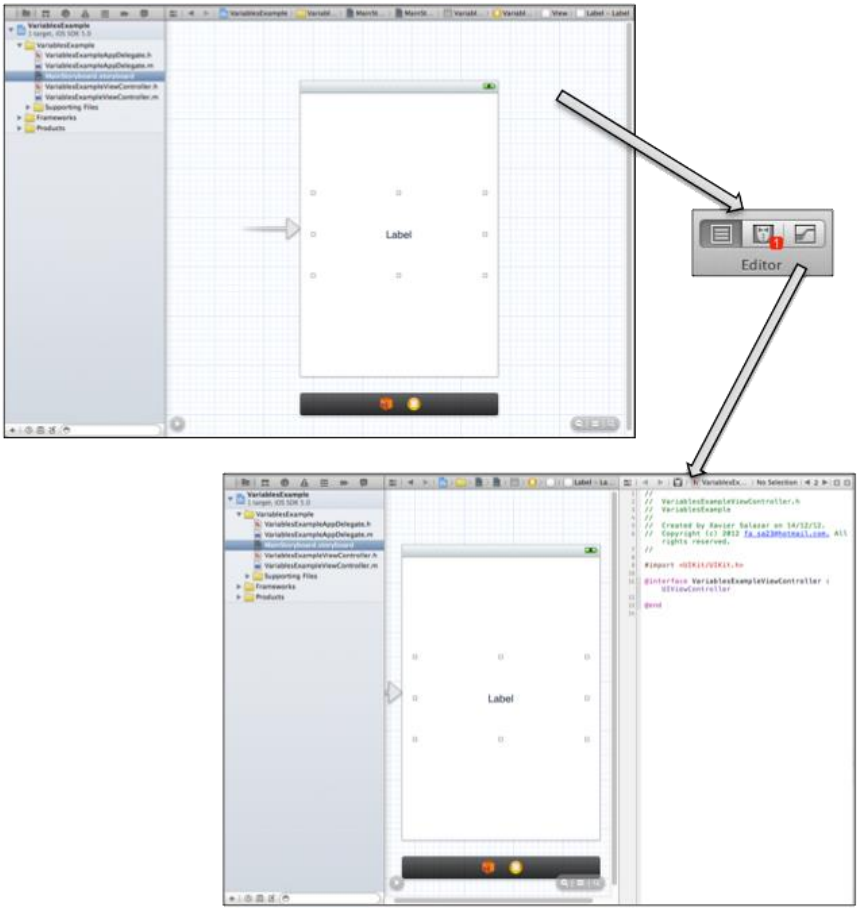


Fig. 2.195: Mostrar el *Assistant editor*

Como se puede observar en la figura anterior, se muestra en una misma ventana el *View Controller* del *Storyboard* y el archivo cabecera de su clase controladora. Acto seguido, se debe mantener presionada la tecla *ctrl*, dar *click* sobre el objeto *Label* y conectarlo con su clase controladora, entre *@interface* y *@end*, en donde se implementan las variables u *outlets* como se indica a continuación:

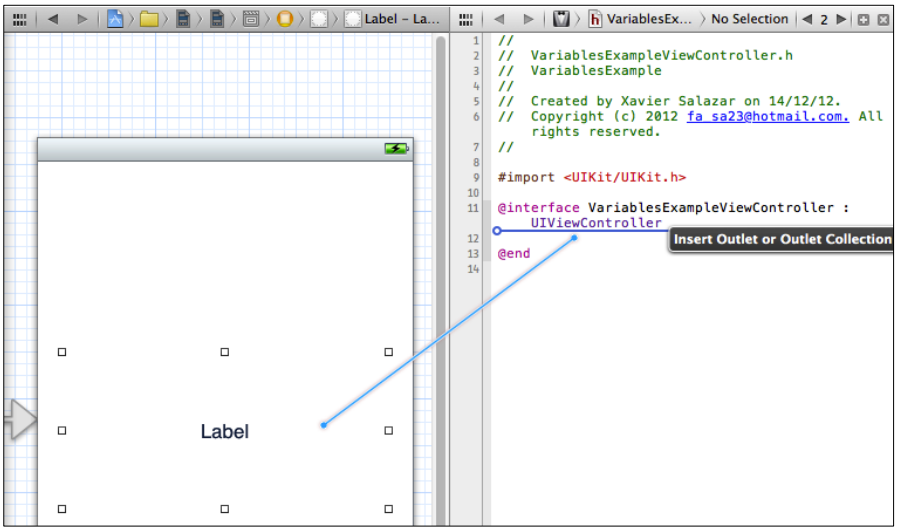


Fig. 2.196: Conexión de un objeto con su respectiva clase controladora

Al hacerlo, se muestra una pequeña ventana que presenta las propiedades de la conexión, en la cual, se puede especificar si se creará un método o una variable para el objeto, y su nombre representativo; como se puede ver a continuación:

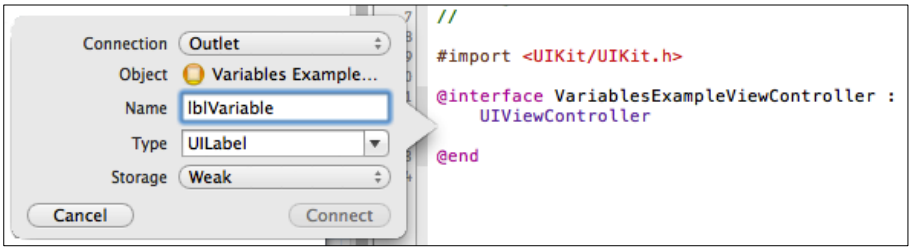


Fig. 2.197: Ventana de propiedades de la conexión (variable/*outlet*)

En esta ventana, en el apartado *Connection*, se indica si se desea asignar una variable o método al objeto; las variables en *XCode* son denominadas *Outlets*, por lo que se debe dejar como está. En el apartado *Name* se escribe el nombre de la variable; en este caso *lblVariable*. El apartado *Type* indica el tipo del objeto y por lo general no se lo puede cambiar. Por último, el apartado *Storage*, indica como se manejará el objeto en memoria, debido a que estos objetos suelen referenciarse entre ellos. Gracias al uso de *ARC*, ya no se deben retener y liberar espacios en memoria, como se hacía en versiones

anteriores de *XCode*. Existen dos alternativas para este apartado: *Strong* y *Weak;* conceptualmente una referencia *Strong* es en la cual el objeto que tiene la referencia del objeto apuntado, lo retiene; y, una referencia *Weak* es la que no lo hace[54].

En este caso para el apartado *Storage* se dejará seleccionado la opción *Weak.* Finalizar presionando el botón *"Connect"*.

De esta forma se ha creado la variable *lblVariable* que representa al objeto *Label* agregado al *View Controller,* sin haber escrito ninguna línea de código (que ha sido generado automáticamente). Esto se puede observar en el archivo cabecera e implementación de la clase *VariablesExampleViewController* en donde se han adicionado las siguientes líneas de código:

```
#import <UIKit/UIKit.h>

@interface VariablesExampleViewController : UIViewController

@property (weak, nonatomic) IBOutlet UILabel *lblVariable;

@end
```

Fig. 2.198: Línea adicionada al archivo cabecera

La línea de código generada en el archivo cabecera, crea la propiedad o variable. La instrucción *nonatomic* en esta sentencia, garantiza un desempeño mucho mas rápido de la variable (Debido a que no se preocupa de que ésta pueda ser requerida por varios procesos a la vez como lo haría la propiedad *atomic*, que no requiere ser escrita).

[54] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p402

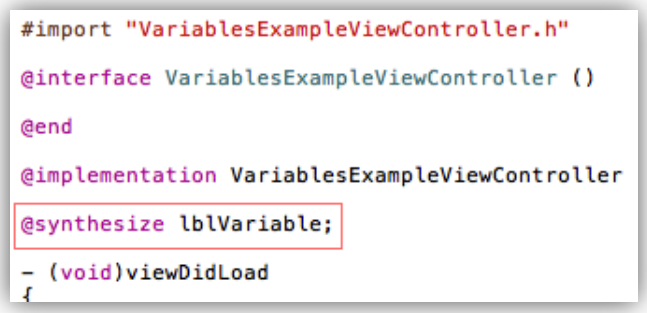


Fig. 2.199: Línea adicionada al archivo de implementación

Por otra parte la sentencia del archivo de implementación *@synthesize*, le indica al compilador que debe sintetizar los métodos *set* y *get* de la variable, como es conocido en programación orientada a objetos; lo cual representa una reducción en tiempo y líneas de código.

A partir de este momento se puede referenciar a la variable en cualquier método del archivo de implementación. Para ejemplificar, se trabajará dentro del método *viewDidLoad*, el cuál es llamado después de que la vista ha sido cargada en memoria; es decir, las sentencias escritas dentro de este método, se ejecutarán cuando se presente la vista en el simulador *iOS*. Se pueden encontrar otros métodos comunes de las vistas en el sub apartado *"2.5.3 Eventos comunes de Views"*.

En al archivo de implementación, en el método *viewDidLoad*, se escribirán las siguientes líneas de código para observar como cambia la variable; se puede notar que el *Label* dentro del *Storyboard* permanece igual; pero, al ejecutar la aplicación en el simulador se observan los cambios:

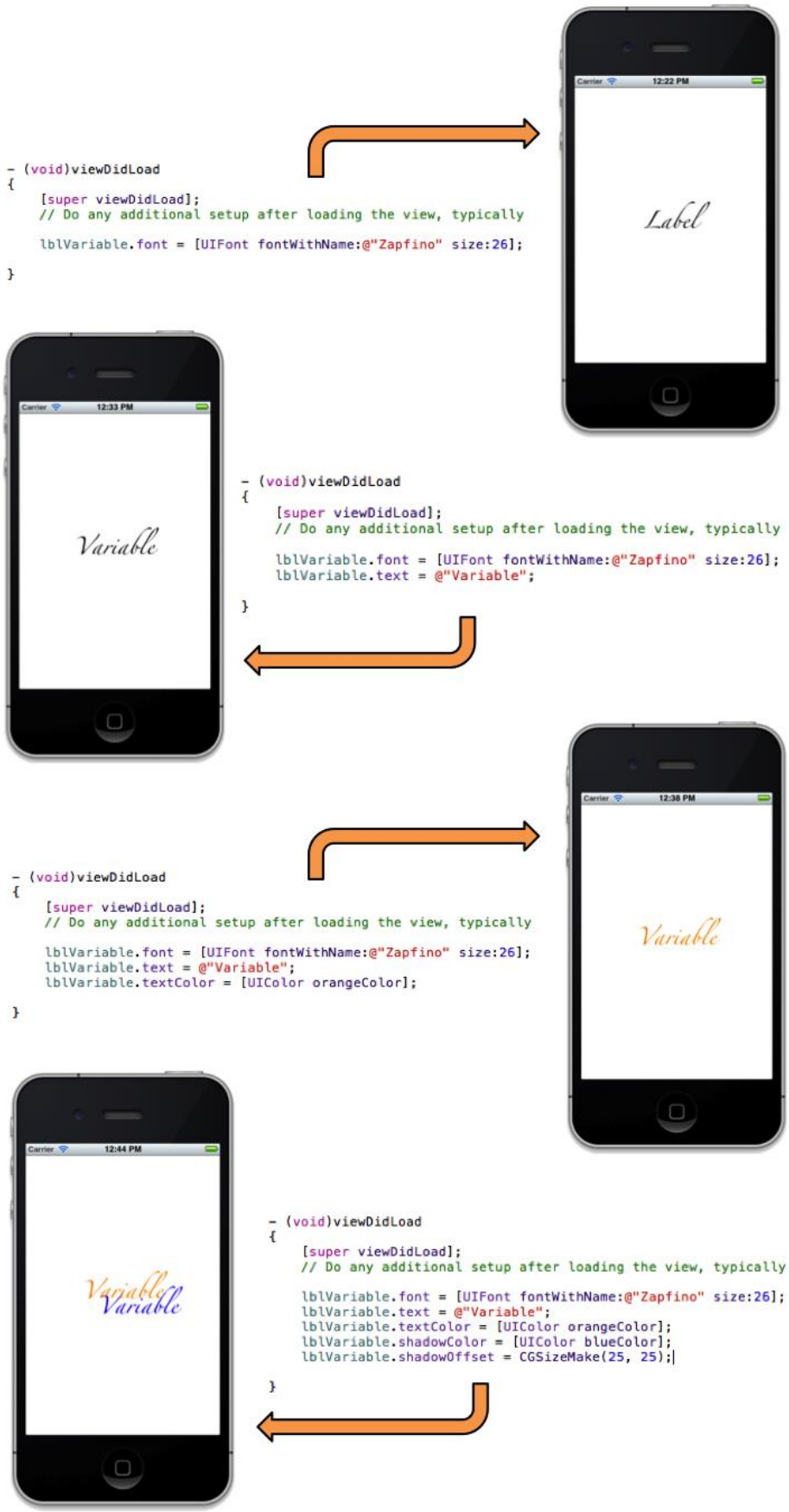


Fig. 2.200: Manejar las propiedades de una variable mediante código fuente

Inicialmente, se cambia el *Font* de la variable así como el tamaño del texto; después se cambia el contenido y color del texto, y; finalmente se indica el color de sombra del texto y el desplazamiento en pixeles del mismo; todo esto a través de código fuente ejecutado en el método *viewDidLoad,* llamado cuando se termina de cargar la vista.

Se puede consultar en la documentación de *Apple* otras propiedades de cada uno de los objetos que pueden conformar una *GUI* a los que se les puede asignar variables.

Se ha generado un proyecto "*AdvancedVariablesExample*" en donde se ha construido una interface gráfica para aprender inglés; en la que cada vez que se abra la aplicación, se presente: la fecha, el día de la semana actual y un número aleatorio en inglés. Se puede revisar este proyecto para conocer cómo asignar una imagen a un botón mediante código fuente, cómo obtener la fecha actual, entre otros. Se han asignado variables a distintos componentes de la interface.

Fig. 2.201: Funcionalidad de *AdvancedVariablesExample*

2.5.2 Asignación de métodos a componentes de GUI

El objetivo de asignar métodos a los componentes *GUI* es el de poder responder adecuadamente cuando el usuario interactúe con éstos. De esta forma se podría por ejemplo presentar un aviso cuando el usuario presione un botón. Los métodos en los componentes son conocidos como *Actions.*

El proceso para asignar un método a un componente *GUI* es relativamente sencillo. Para explicar este proceso, se creará un proyecto nuevo llamado *ActionsExample* usando la plantilla "*Single View*"; si se requiere ayuda se debe referir al paso 1 del apartado *"2.4.4 Rotación de interfaces y sus componentes"*.

A continuación se agregará un objeto *Round Rect Button* al *View Controller* del *Storyboard* como se indica a continuación:

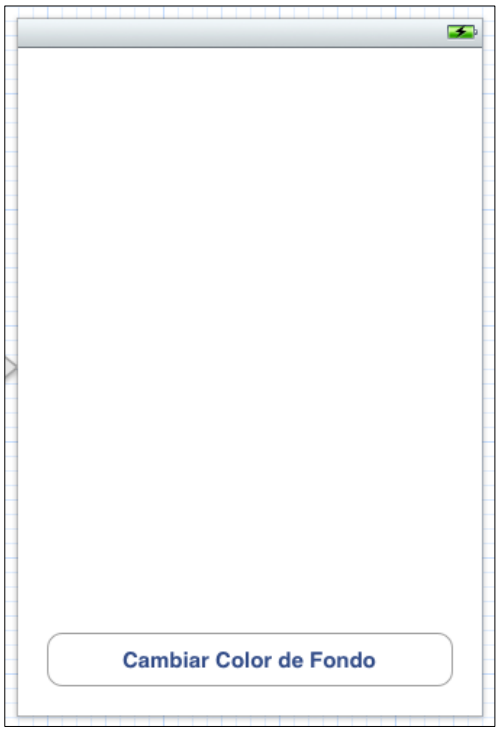


Fig. 2.202: Agregar un *Round Rect Button* y ajustarlo como se indica.

Acto seguido se asignará un método al objeto *Round Rect Button*, con el objetivo de responder a la acción "presionar el botón". Cada vez que se suscite este evento, se

cambiará el color de fondo de la vista. Para esto, se debe mostrar el "*Assistant editor*" (), que puede ser localizado en la parte superior del entorno de trabajo en la sección "*Editor*", como se indica en la Fig. 2.195.

De esta forma, se mostrará en la misma ventana el *View Controller* del *Storyboard* y el archivo cabecera de su clase controladora. Acto seguido, se debe mantener presionada la tecla *ctrl*, dar *click* sobre el objeto *Round Rect Button* y conectarlo con su clase controladora, entre *@interface* y *@end*, en donde se implementan los métodos o *actions*, como se indica a continuación:

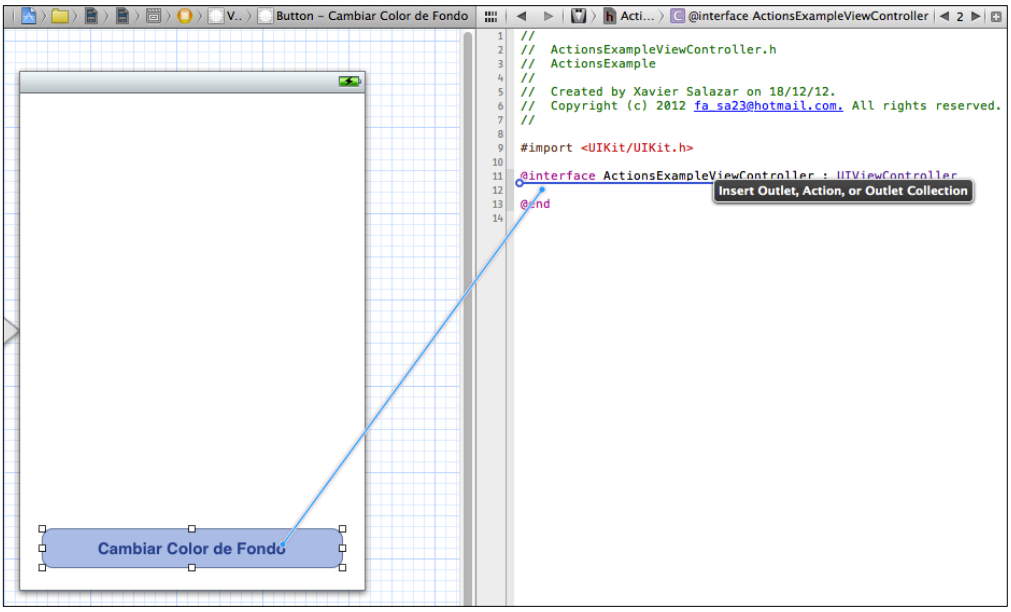


Fig. 2.203: Conexión del *Round Rect Button* con su clase controladora.

Al hacerlo, se muestra la ventana de propiedades de conexión (Fig. 2.204) en donde se debe indicar que se desea asignar un método al objeto, al seleccionar *(1) Action* en el apartado *Connection.* En el apartado *Name*, se debe nombrar al método; en este caso se lo ha llamado *cambiarColorFondo (2)* debido a su funcionalidad. En el apartado *Type*, es recomendado dejarlo como *id (3)*, debido a que de esta manera se pudiera asignar el mismo método para diferentes objetos y no solo para el botón.

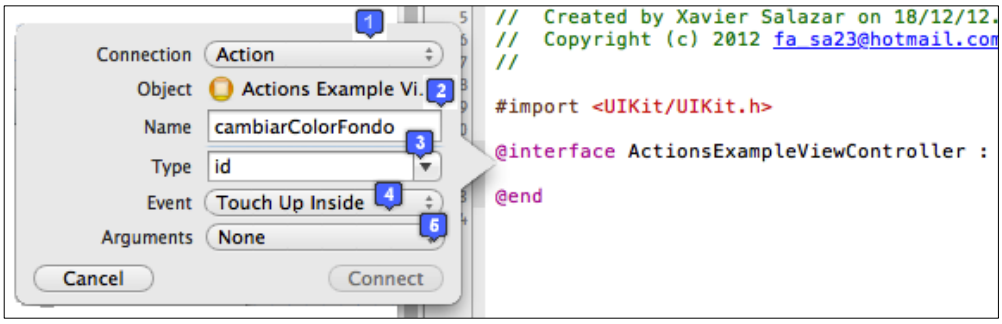


Fig. 2.204: Ventana de propiedades de la conexión (método/action)

El apartado *Event* presenta una serie de eventos o acciones comunes para la mayoría de objetos. El método *Touch Up Inside (4)* se dispara al presionar el botón, por lo que ha sido seleccionado.

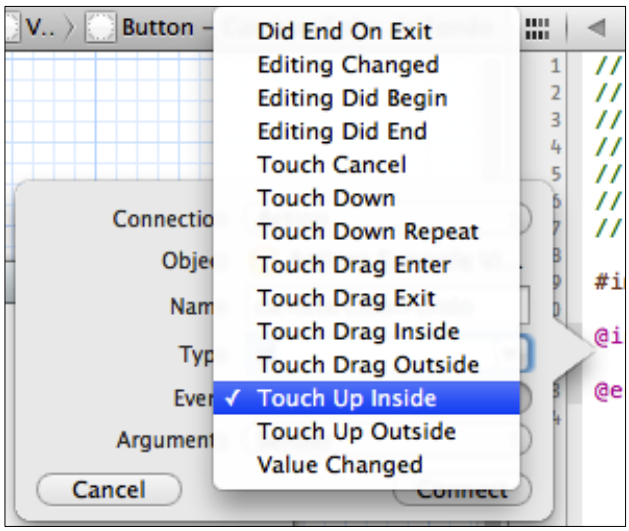


Fig. 2.205: Métodos del objeto *Round Rect Button*

Finalmente el apartado *Arguments* indica los parámetros que tendrá el método, que pueden ser: *None* o ninguno, *Sender* u objeto que envía o dispara el evento, en este caso el botón, o *Sender and Event* que es el objeto anterior mencionado y el evento disparado. Para cambiar el color de fondo de la vista no se requiere ninguno de los parámetros por lo que se ha seleccionado *None (5).*

Finalizar presionando el botón *"Connect"*.

De esta forma se ha creado el método *cambiarColorFondo* que se ejecutará al presionar el *Round Rect Button* agregado al *View Controller*, sin haber escrito ninguna línea de código (que ha sido generado automáticamente). Esto se puede observar en el archivo cabecera e implementación de la clase *ActionsExampleViewController* en donde se han adicionado las siguientes líneas de código:

```
#import <UIKit/UIKit.h>

@interface ActionsExampleViewController : UIViewController

- (IBAction)cambiarColorFondo;

@end
```

Fig. 2.206: Línea adicionada al archivo cabecera

La línea de código generada en el archivo cabecera, declara el método *cambiarColorFondo*. La instrucción *IBAction* en esta línea de código, indica que ésta es la declaración de un método.

```
}

- (IBAction)cambiarColorFondo {
}
@end
```

Fig. 2.207: Línea adicionada al archivo de implementación

Por otra parte dentro del archivo de implementación se ha agregado el método con llaves para escribir dentro de las mismas el código fuente que se debe ejecutar al dispararse este método.

Para cambiar el color de fondo de la vista cada vez que se presione el botón se debe escribir el siguiente código fuente dentro de la implementación del método generado:

```
- (IBAction)cambiarColorFondo {

    double aleatorioRojo = (rand() % 256) / 255.0;
    double aleatorioVerde = (rand() % 256) / 255.0;
    double aleatorioAzul = (rand() % 256) / 255.0;

    [self.view setBackgroundColor:[UIColor colorWithRed: aleatorioRojo  green:
        aleatorioVerde blue:aleatorioAzul alpha:1]];

}
```

Fig. 2.208: Código que se debe escribir para generar un color de fondo aleatorio

Este código se ejecutará cada vez que se presione el *Round Rect Button* agregado sobre el *View Controller* del *Storyboard.*

Para analizar este código, primero se debe comprender el sistema *RGB* (*red*, *green*, *blue*), el cual representa los colores que pueden ser visualizados en una computadora; estos colores pueden ser combinados en varias proporciones (representadas por un valor entre 0 y 255) para obtener cualquier color.

Se han creado tres variables de tipo *double aleatorioRojo, aleatorioVerde* y *aleatorioAzul* para almacenar los valores de rojo, verde y azul respectivamente, los cuales deben ser un valor aleatorio entre 0 y 255. Para obtener estos valores, se usó la función *rand()* seguido de *%256* que indica el límite no inclusive del valor aleatorio que ésta retornará.

Lamentablemente la función "*colorWithRed: green: blue: alpha:*" de la clase *UIColor*, no recibe directamente valores entre 0 y 255, sino que los recibe representados por un valor decimal entre 0 y 1 inclusive; por lo tanto, a cada variable anterior se la ha dividido para 255.0

Finalmente mediante la función *setBackgroundColor* de la clase *UIView*, se establece el color de fondo de la vista. Al ejecutar la aplicación se puede presionar el *Round Rect Button* y observar como cambia el color de la vista con cada presión del botón.

Fig. 2.209: Aplicación recién construida en ejecución

Se puede consultar en la documentación de *Apple* otros métodos o eventos que se les puede asignar a cada uno de los objetos de *GUI.*

Se ha generado un proyecto "*AdvancedActionsExample*" en donde se ha construido una *GUI* para aumentar o disminuir un número contenido en un *Label* de a cuerdo al botón pulsado. Se puede revisar este proyecto para conocer como modificar el texto de un *Label* al presionar un botón, asignándole un evento al mismo.

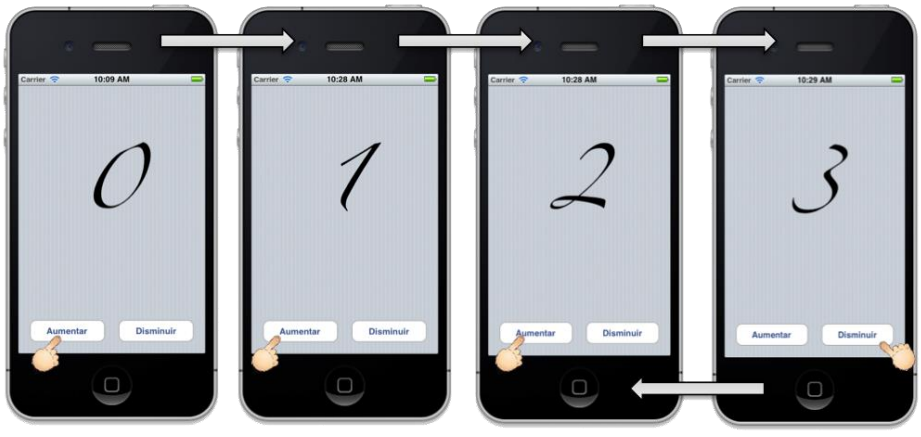


Fig. 2.210: Funcionalidad de *AdvancedActionsExample*

2.5.3 Eventos comunes de Views

Las vistas que forman parte de una aplicación, se componen de varios eventos o métodos de su clase "*UIView*" que pueden ser utilizados para realizar ajustes de la interface de usuario, cargar o grabar datos, entre otras funciones. Estos métodos no deben ser siempre implementados, se los puede implementar únicamente cuando sean necesarios; incluso, algunos se incluyen por defecto al generar la clase controladora de la vista. Los más importantes son los siguientes:

viewDidLoad: Evento que es llamado cuando la vista se ha cargado. Dentro de éste, se puede hacer cualquier configuración adicional después de haber cargado la vista, como cargar datos para presentarlos en pantalla.

viewDidUnload: Evento que es llamado cuando la vista se descarga, es decir, no es necesaria y se oculta. Dentro de éste, se pueden grabar datos ingresados por el usuario, así como liberar el espacio de memoria reservado para variables.

viewDidAppear: Evento que notifica al *View Controller* que su vista fue añadida a una ventana. Consta de un parámetro *BOOL* que indica si la vista fue añadida a la ventana usando una animación o no.

viewDidDisappear: Evento que notifica al *View Controller*, que su vista fue despedida, cubierta o escondida por otra vista. Usa un parámetro que *BOOL* que indica si la vista fue despedida con una animación o no.

ViewWillAppear: Evento que notifica al *View Controller* que su vista está punto de mostrarse. Usa un parámetro *BOOL* que indica si la vista será presentada utilizando una animación o no.

viewWillDisappear: Evento que notifica al *View Controller* que su vista esta a punto de ser despedida, cubierta o escondidgrea por otra vista. Usa un parámetro *BOOL* que indica si la vista será despedida utilizando una animación o no.

Se recomienda analizar estos métodos para comprender con exactitud "qué se debe implementar" y, "dentro de qué método".

2.5.4 Eventos comunes de Table Views

Las vistas de tabla o *Table Views* que pueden formar parte de una aplicación, se componen de varios eventos o métodos de su clase "*UITableView*" que son utilizados para ajustar las dimensiones de la tabla, agruparla, presentar datos en la misma, entre otras funciones. Estos métodos no deben ser siempre implementados; se los puede implementar únicamente cuando sean necesarios; incluso algunos, se incluyen por defecto al generar la clase controladora de la vista de tabla. Los más importantes son los siguientes:

numberOfSectionsInTableView: Método que pregunta por el numero de secciones que tendrá la tabla. Requiere de un parámetro que representa a la tabla.

tableView:numberOfRowsInSection: Método que pregunta el numero de filas que existirán por cada una de las secciones de la tabla. Requiere de un parámetro que representa a la tabla y de otro que representa a la sección específica.

tableView:cellForRowAtIndexPath: Método que personaliza cada celda a ser insertada dentro de una localización específica de la tabla. Utiliza un parámetro que representa la tabla y otro que indica el índice (*index*) dentro de la tabla, en donde será insertada la celda o fila.

tableView:didSelectRowAtIndexPath: Evento que indica a su delegado que se ha seleccionado una fila especifica. Requiere de un parámetro que representa a la tabla y de otro que indica el índice de la fila seleccionada.

tableView:heightForRowAtIndexPath: Método que pregunta por el alto (en pixeles) que se desea utilizar en una fila, en una localización específica. Requiere de un parámetro que representa a la tabla y de otro que indica la ubicación de la fila dentro de ésta.

tableView:titleForHeaderInSection: Método que pregunta por el título de la cabecera de una sección especifica de la tabla. Requiere de un parámetro que representa a la tabla y de otro que indica la sección específica de la misma.

tableView:titleForFooterInSection: Método que pregunta por el titulo del pie de una sección especifica de la tabla. Requiere de un parámetro que representa a la tabla y de otro que indica la sección específica de la misma.

tableView:accessoryTypeForRowWithIndexPath: Método que pregunta por el tipo de accesorio estándar que se usará como control de divulgación (*disclosure*) para una fila específica de la tabla. Requiere de un parámetro que representa a la tabla y de otro que indica la fila específica de la misma a la cual se le asignará el tipo de accesorio. Este método ha sido deprecado, es decir será anulado en futuras versiones de *XCode*, por lo tanto, si se desea establecer el accesorio de una fila de la tabla, se lo debe hacer estableciendo el atributo correspondiente del objeto.

tableView:accessoryButtonTappedForRowWithIndexPath: Evento que indica al delegado que el usuario presionó el accesorio asociado a la fila de la tabla. Requiere de un parámetro que representa a la tabla y de otro que indica la fila específica de la misma.

tableView:commitEditingStyle:forRowAtIndexPath: Método que permite manejar los eventos de inserción o borrado de una fila específica de la tabla. Requiere de un parámetro que representa a la tabla; otro, que indica la fila específica en cuestión y un último parámetro que indica el estilo de edición; es decir, indica si se trata de una inserción o borrado de fila.

Fig. 2.211: Eventos para configurar un *Table View*

El método *tableView:cellForRowAtIndexPath:* es uno de los más importantes, debido a que dentro del mismo se configura el contenido de cada una de las filas del *Table View*. Se puede personalizar el contenido, accesorio, tipo de letra, vista, entre otras características de las filas de un *Table View*. Se ha generado la aplicación de la fig. 2.211 y se la ha llamado *TableViewEvents;* se recomienda analizar el código fuente de la misma para comprender mejor el funcionamiento de los métodos mencionados en el presente apartado.

2.6 Conexión de vistas en Storyboards

En versiones anteriores de *XCode,* la conexión entre vistas que conforman una aplicación se realizaba exclusivamente mediante código fuente, lo que resultaba en grandes cantidades de líneas de código y dedicación de un tiempo considerable por parte de los desarrolladores.

A partir de la versión 4.2.1 de *XCode*, con la aparición de los *Storyboards,* el proceso de conexión de vistas se simplificó considerablemente. El proceso de conexión de vistas mediante *Storyboards* es exclusivamente gráfico, en el cual, el desarrollador no debe escribir ninguna línea de código, salvo para casos excepcionales; ahorrando así un tiempo considerable.

La forma de conexión entre vistas de una aplicación, depende de la plantilla que se utilice para crear la misma. Una aplicación puede tener distintos tipos de conexiones entre vistas a la vez.

En los siguientes sub apartados, se explicará como realizar la conexión entre vistas utilizando los distintos tipos de plantillas presentados en el sub apartado *"2.4.3 Análisis de plantillas prediseñadas para un proyecto"*.

2.6.1 Implementación de un *Navigation Controller* (Control de Navegación)

El *Navigation Controller* es el encargado de manejar la navegación entre una jerarquía de vistas, como se ha indicado en el análisis del mismo realizado en el apartado *"2.4.1 Análisis de componentes comunes"*.

Para comprender el proceso de implementación de un *Navigation Controller*, se debe crear un nuevo proyecto vacío y nombrarlo *NavigationControllerExample*. Si se requiere ayuda, se debe referir al apartado *"2.2 Cómo crear un nuevo proyecto"*.

A continuación se debe agregar un *Storyboard* en donde se construirá la interface utilizando el *Navigation Controller*.

Acto seguido se arrastrará un *Navigation Controller* sobre el *Storyboard* como se indica a continuación:

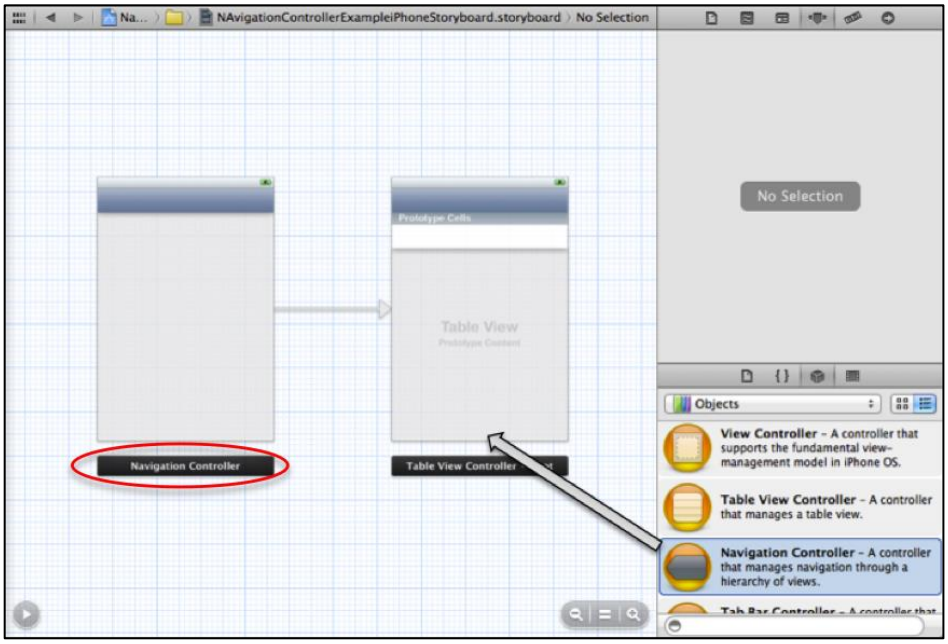


Fig. 2.212: Arrastrar un *Navigation Controller* sobre el *Storyboard*

El componente *Navigation Controller* se agrega por defecto asociado con un *Table View Controller*, el cual no es necesario por el momento por lo que será eliminado, conservando únicamente el componente *Navigation Controller*, así:

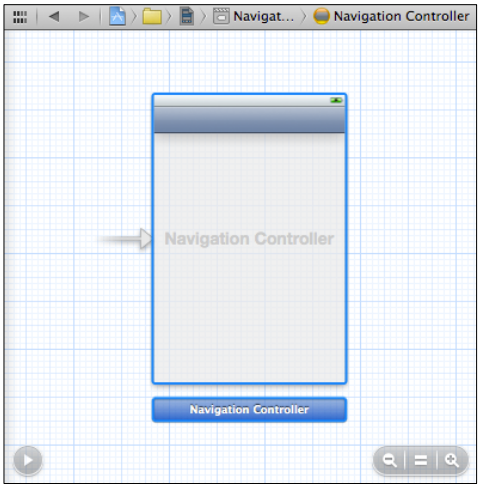


Fig. 2.213: Navigation Controller sin Table View Controller (no necesario)

El siguiente paso consiste en agregar la vistas que formarán parte de la navegación de la aplicación. En la Fig. 2.77 se puede visualizar un *Navigation Controller* que conecta a tres *Table View Controller*. Para este ejemplo se agregará un *View Controller* y un *Table View Controller* que implementarán el *Navigation Controller*. Para esto, se debe arrastrar al *Storyboard* un *View Controller* y un *Table View Controller* así:

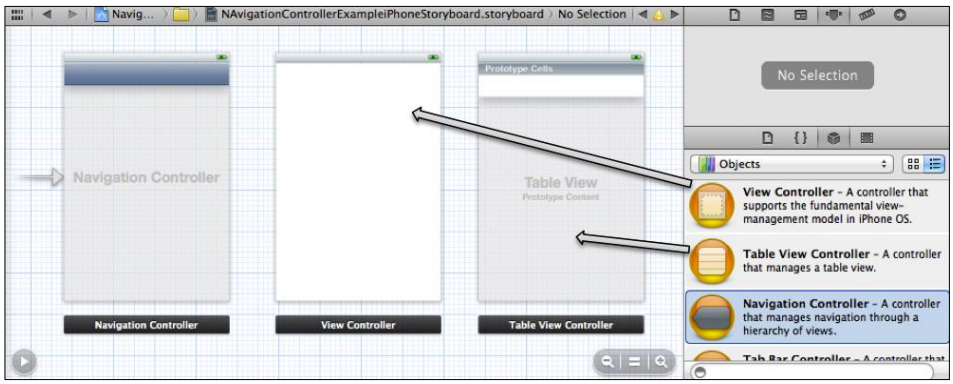


Fig. 2.214: Agregar un View Controller y un Table View Controller

A continuación se establecerá las relaciones entre las vistas y el *Navigation Controller*, para ello, se debe mantener pulsada la tecla "*ctrl*", presionar el *Navigation Controller* y conectarlo con el *View Controller* como se indica a continuación:

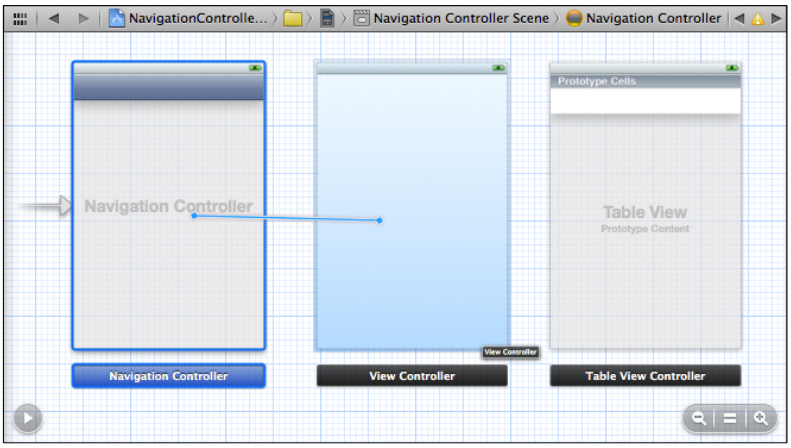


Fig. 2.215: Conexión del Navigation Controller con el View Controller

Al finalizar la conexión se presenta un menú desplegable del tipo de conexión que se desea establecer con cuatro opciones:

***Relationship – RootViewController*:** Establecer una relación y seleccionar como vista principal o raíz.

***Push*:** Empujar una vista para presentarla

***Modal*:** Colocar una vista sobre otra hasta que el usuario realice una acción concreta.

Custom: Presentar una vista de manera personalizada mediante código fuente.

Se debe seleccionar el tipo de conexión "*Relationship – Root View Controller"* de la Fig. 2.216. Al hacerlo se podrá visualizar la conexión y el *Navigation Controller* en la parte superior del *View Controller* asociado, como se puede observar en la Fig. 2.217.

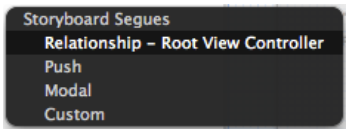


Fig. 2.216: Tipos de conexión

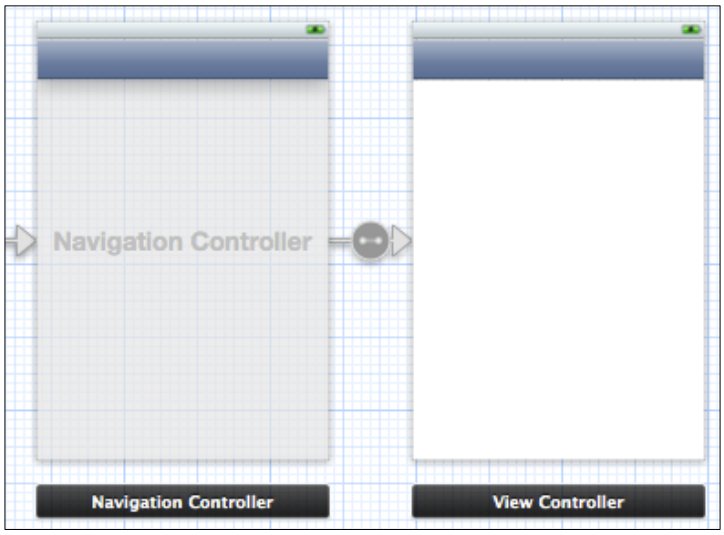


Fig. 2.217: Navigation Controller conectado al View Controller

En este momento se puede personalizar el título del *Navigation Controller* dentro del *View Controller*, para lo cual se debe dar un doble *click* sobre el mismo y llamarlo "Vista 1". Se ha establecido el fondo "*Group Table View Background Color*" desde los atributos de la vista y se ha agregado un *Round Rect Button* para presentar la siguiente vista como se muestra a continuación:

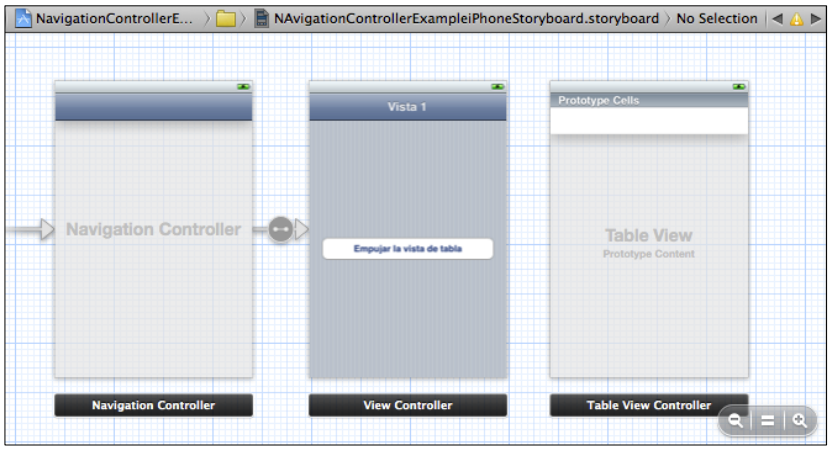


Fig. 2.218: Vista 1 personalizada y con *Navigation Controller*

Finalmente para conectar la Vista 1 con el *Table View Controller*, se realiza un proceso similar: mantener presionada la tecla "*ctrl*" y realizar una conexión desde el *Round Rect Button* de la Vista 1 hasta el *Table View Controller*, así:

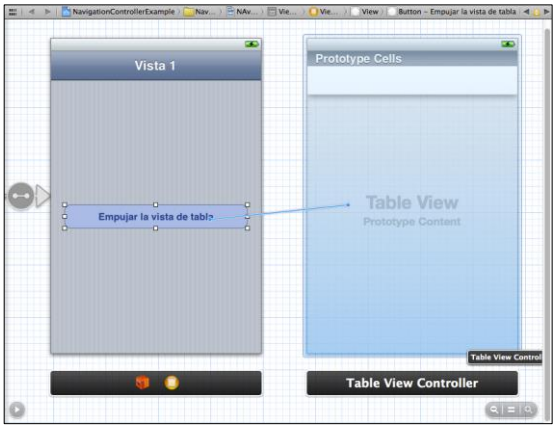


Fig. 2.219: Conexión del botón con el *Table View Controller*

A partir de la segunda vista conectada al *Navigation Controller*, se debe seleccionar *Push* en el tipo de conexión (Fig. 2.216). Ahora se puede escribir el título del *Navigation Controller* en el *Table View Controller* asociado; se lo nombrará "Vista de Tabla 1". Se ha personalizado también el estilo del *Table View*, seleccionando *Grouped* para el atributo *Style* en sus propiedades. Al finalizar, el *Storyboard* se debería ver así:

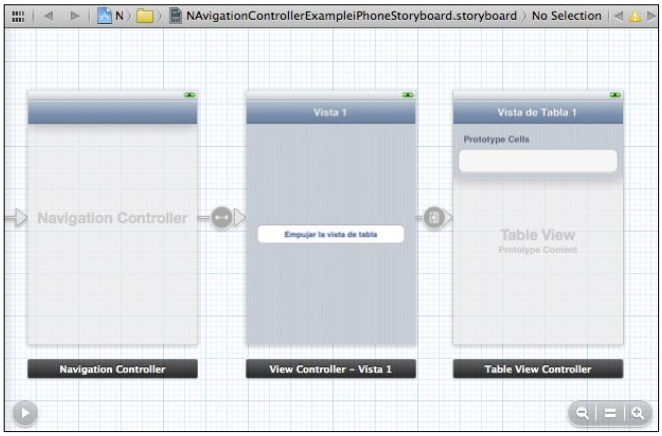


Fig. 2.220: *Storyboard* de NavigationControllerExample

Finalmente se agregará una clase *UITableViewController* para indicar el numero de secciones (1) y filas (2) del *Table View* (lo cual sirve únicamente para este ejemplo, debido a que ya se ha implementado el *Navigation Controller* exitosamente). Si se requiere ayuda, se recomienda revisar los apartados *"2.3.1 Cómo agregar un fichero a*

un proyecto" y *"2.5.4 Eventos comunes de Table Views"* y esta aplicación "*NavigationControllerExample*".

Antes de ejecutar la aplicación en el simulador, se deben borrar las líneas de código innecesarias del fichero *AppDelegate* (ver "*Ajuste 2*" dentro del apartado "*2.4.2 Construcción de una interface gráfica de usuario*") y asociar el *Storyboard* agregado, a la aplicación (ver "*2.3.1 Cómo agregar un fichero a un proyecto*", sub apartado "*A) Agregar un Storyboard*", *Fig. 2.48*).

Al ejecutar la aplicación se observa el siguiente flujo gracias al *Navigation Controller* presente en sus vistas:

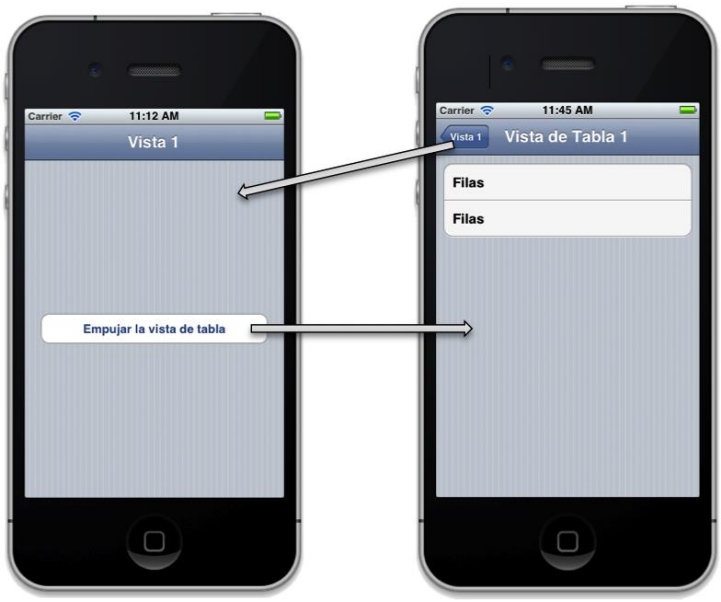


Fig. 2.221: Flujo de una aplicación que implementa un *Navigation Controller*

Una aplicación que implementa un *Navigation Controller*, lo mantiene siempre visible en todas o la mayoría de sus vistas. Cuando se llama a otra vista, el *Navigation Controller* crea un botón que permite regresar a la vista anterior, como se puede observar en "Vista de Tabla 1", la cual implementa el botón "Vista 1" dentro de su *Navigation Controller*.

Importante: Si no es posible realizar la conexión gráficamente dentro del *Storyboard*, (por ejemplo conectar la quinta fila de un *Table View* con otro *View Controller*), se puede instanciar la vista mediante código fuente escribiendo estas líneas dentro del evento deseado.

```
- (void)tableView:(UITableView *)tableView accessoryButtonTappedForRowWithIndexPath:
    (NSIndexPath *)indexPath{

    AdditionalNotesView *additionalNotesView = [self.storyboard
        instantiateViewControllerWithIdentifier:@"AdditionalNotesViewIdentifier"];
    [self.navigationController pushViewController:additionalNotesView animated:YES];

}
```

Fig. 2.222: Instanciar una vista mediante código fuente.

Gracias a éste código, en la Fig. 2.222, se instancia a una vista "*AdditionalNotesView*", dentro del evento del *Table View Controller* que consiste en presionar un botón dentro de la fila de una tabla. El identificador de la vista lo decide el desarrollador, en este caso es *AdditionalNotesViewIdentifier*, el cual, debe ser asignado también en las propiedades de la vista como se indica a continuación:

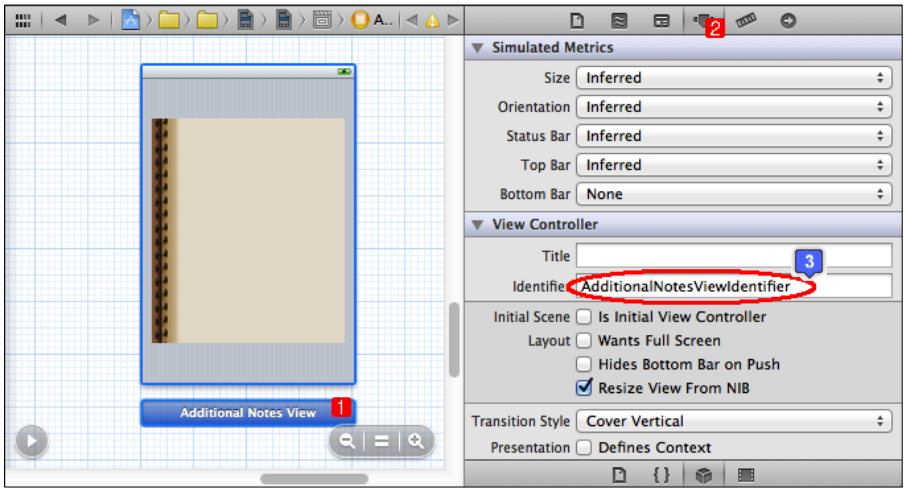


Fig. 2.223: Asignación de un identificador para una vista.

2.6.2 Implementación de un Tab Bar Controller (Control de Pestañas)

Como se ha indicado en el apartado *"2.4.1 Análisis de componentes comunes"*, un *Tab Bar Controller* puede manejar un conjunto de *View Controllers* que representan elementos del *Tab Bar* o barra de pestañas. Cada *View Controller* muestra información sobre la pestaña a la que corresponde y presenta la vista correspondiente cuando es seleccionado en el *Tab Bar*.

Para desarrollar una aplicación que implemente un *Tab Bar Controller*, se podría partir de la plantilla *Tabbed Application*, pero en lugar de ello, se generará un nuevo proyecto vacío y se lo nombrará *TabBarControllerExample* de forma de comprender mejor el funcionamiento del componente. Si se requiere ayuda, se debe referir al apartado *"2.2 Cómo crear un nuevo proyecto"*.

Acto seguido se debe agregar un *Storyboard* en donde se construirá la interface utilizando el *Tab Bar Controller.*

Para empezar se debe arrastrar un *Tab Bar Controller* sobre el *Storyboard* como se indica a continuación:

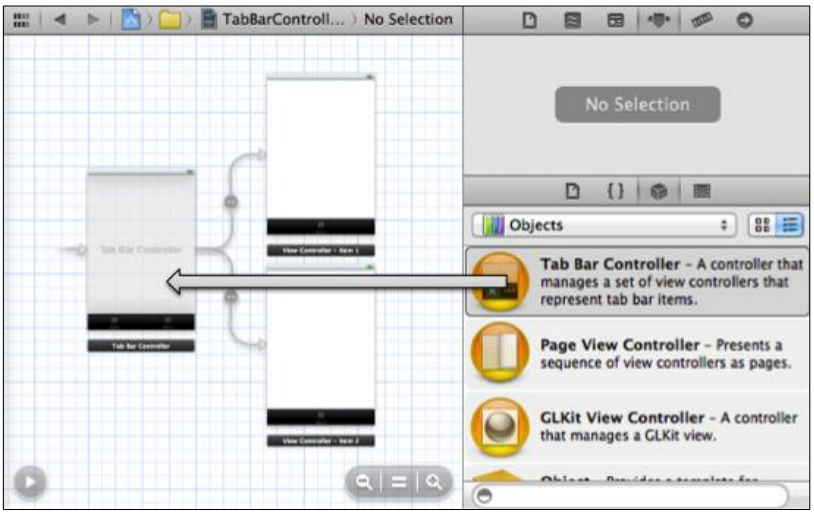


Fig. 2.224: Arrastrar un *Tab Bar Controller* sobre el *Storyboard*

Al agregar un *Tab Bar Controller*, viene por defecto asociado a dos *View Controllers.* Como se puede observar, cada *View Controller* tiene una pestaña o *Tab* en su parte

inferior; mientras el *Tab Bar Controller* esta compuesto de dos *Tabs* en su barra de pestañas debido a que se encuentra asociado a dos *View Controllers*.

El siguiente paso consiste en agregar dos *View Controllers* adicionales sobre el *Storyboard* y conectarlos con el *Tab Bar Controller* (mantener presionada la tecla "*ctrl*" y trazar la conexión desde el *Tab Bar Controller* hasta cada uno de los *View Controllers*) como se indica a continuación:

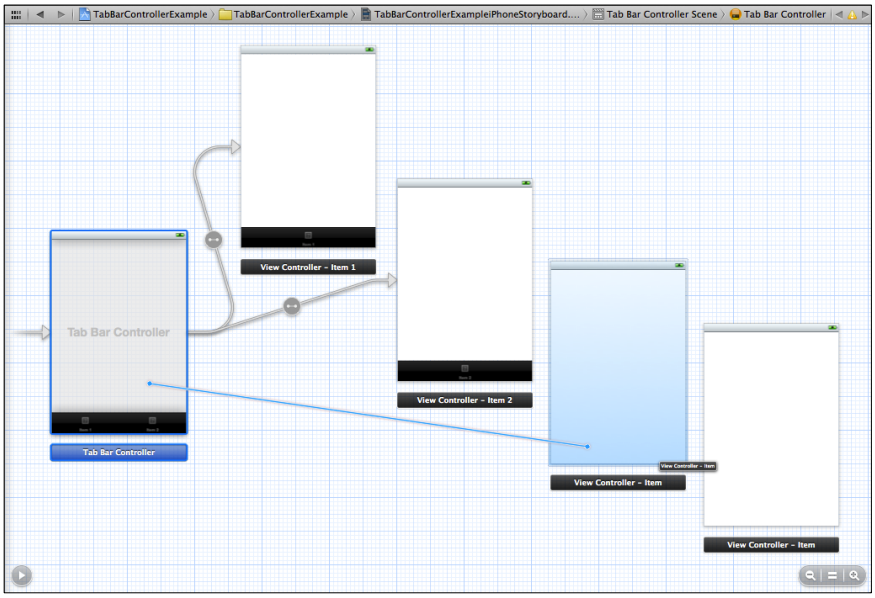


Fig. 2.225: Asociar mas View Controllers al Table View Controller

Al realizar la conexión, se presenta una ventana para indicar su tipo, la cual contiene las mismas opciones de la presentada en la Fig. 2.216, excepto por la primera opción la cual en este caso es *"Relationship – View Controllers"*. Se debe seleccionar este tipo de conexión para cada una de las vistas que se asocien al *Tab Bar Controller*.

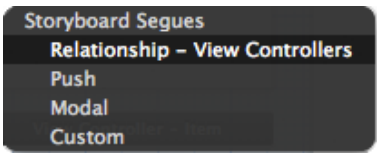


Fig. 2.226: Tipos de conexión con un *Tab Bar Controller*

Al finalizar las conexiones, el *Storyboard* se verá así:

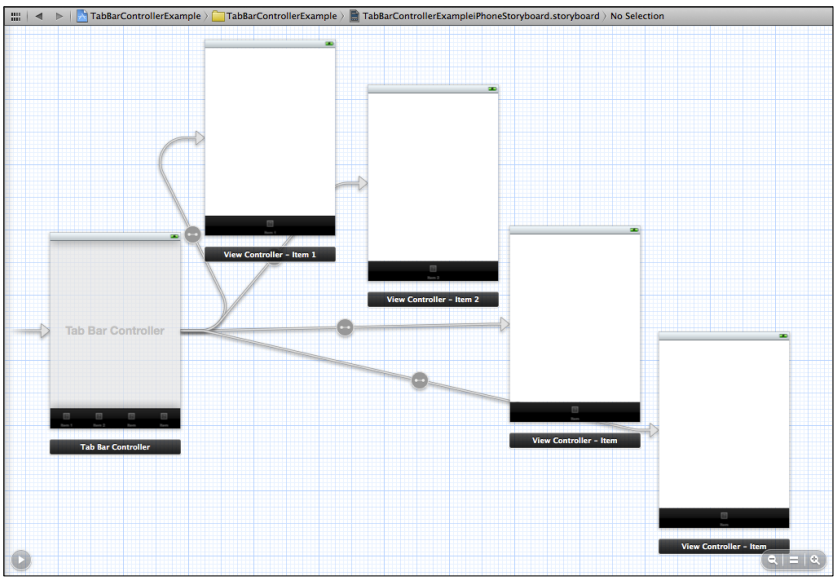


Fig. 2.227: Tab Bar Controller asociado a cuatro View Controllers

Como se puede observar, el *Tab Bar Controller* consta ahora de 4 pestañas o *Tabs* dentro de su *Tab Bar*, cada uno de los cuales se encuentra asociado a un *View Controller* específico.

Para diferenciar los *View Controllers*, se asignará un distinto color de fondo para cada uno. Los íconos de cada pestaña o *Tab*, deben ser imagenes en formato "*.png"*, y de preferencia con un tamaño máximo de 35x35 pixeles que deben ser previamente agregadas al proyecto. Al asignar una imagen como ícono de un *Tab*, la imagen toma una tonalidad gris en caso de haber sido colorida, es por esto que se utilizarán imágenes grises. Los iconos utilizados en este apartado se encuentran en el comprimido *IconosTabBarControllerExample.zip*. Luego de agregar las imágenes al proyecto, se debe seleccionar la barra inferior de uno de los *View Controllers* para poder editar su título e ícono como se indica a continuación:

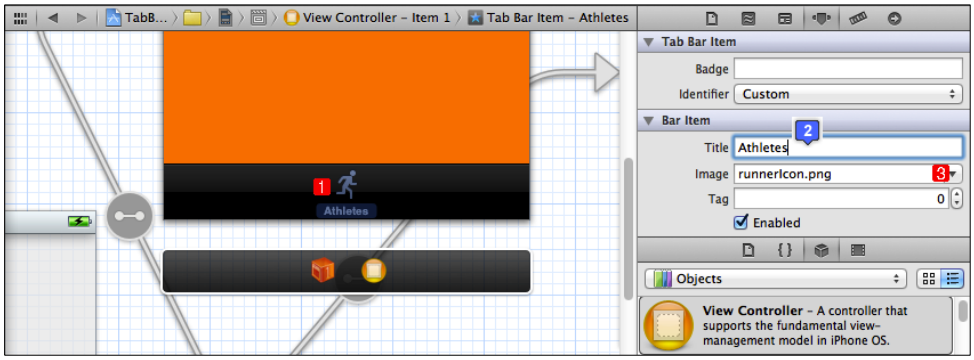


Fig. 2.228: Personalización del ícono y título de pestaña de un *View Controller*

Repetir el proceso anterior para personalizar las pestañas de los *View Controllers* restantes para que queden así:

Fig. 2.229: Personalización del resto de pestañas de los *View Controllers* del *Storyboard*

Una vez finalizado este proceso, se puede observar que la barra de pestañas del *Tab Bar Controller* consta de cada pestaña personalizada anteriormente.

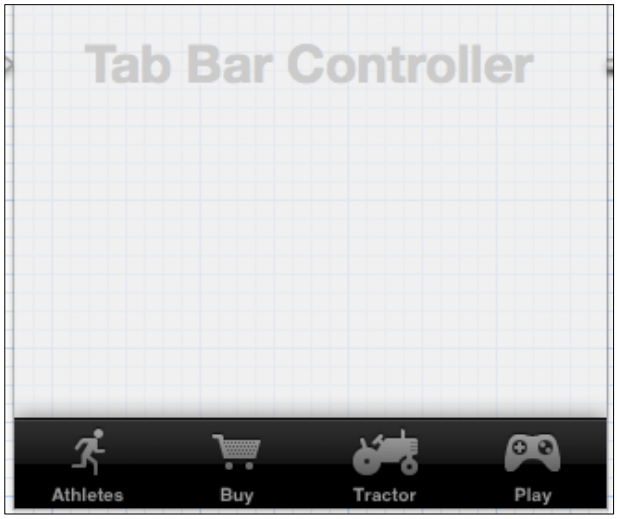


Fig. 2.230: *Tab Bar Controller* con *Tab Bar* personalizado

Para administrar y manejar las vistas, se pueden agregar clases *UIViewController* y asignarlas a cada una de las mismas.

Antes de ejecutar la aplicación en el simulador, se deben borrar las líneas de código innecesarias del fichero *AppDelegate* (ver "*Ajuste 2*" dentro del apartado "*2.4.2 Construcción de una interface gráfica de usuario*") y asociar el *Storyboard* agregado, a la aplicación (ver "*2.3.1 Cómo agregar un fichero a un proyecto*", sub apartado "*A) Agregar un Storyboard*", *Fig. 2.48*).

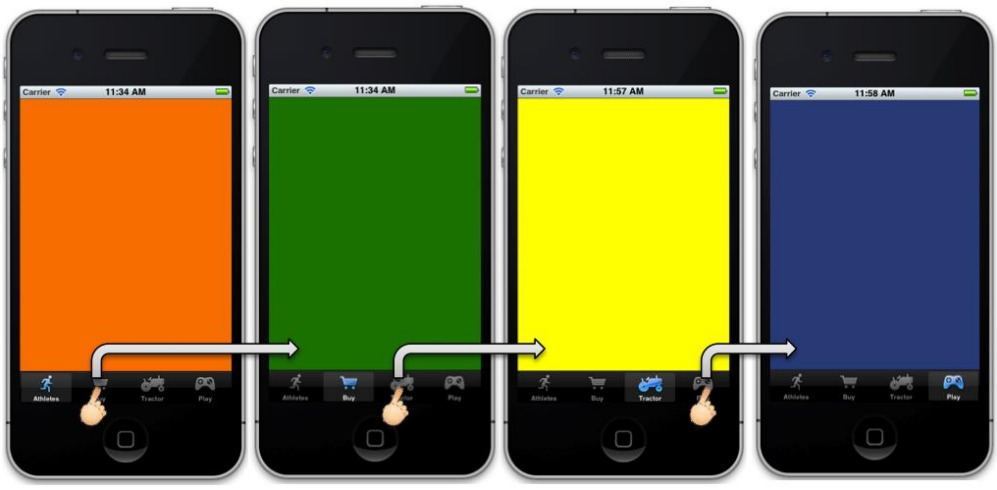


Fig. 2.231: *TabBarControllerExample* en ejecución

Al ejecutar la aplicación se observa al *Tab Bar Controller* en la parte inferior de cada vista; y, al presionar cada pestaña, se presenta la vista correspondiente a la misma. Una aplicación de *iOS* muy conocida e incluida por defecto en los dispositivos *iPhone*, que implementa un *Tab Bar Controller* es la aplicación "*Teléfono",* como se puede observar en la Fig. 2.79.

2.6.3 Análisis de tipos de transiciones

Al conectar dos *View Controllers (o Table View Controllers)*, la conexión es representada en el *Storyboard* mediante una flecha denominada *Segue*.

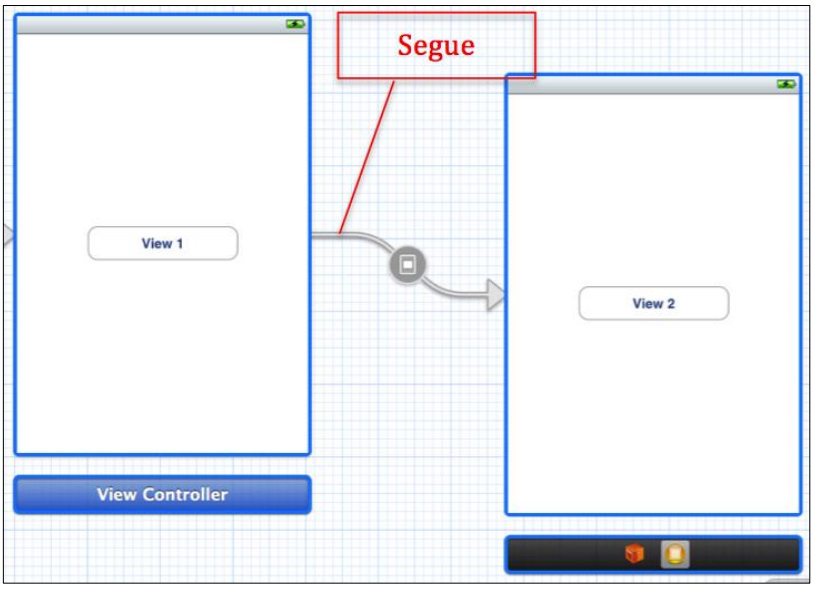


Fig. 2.232: Flecha que representa una conexión entre dos vistas (*Segue*)

Al realizar una conexión entre dos vistas, se presenta una ventana para indicar el tipo de conexión que se realizará. Como se indicó en el apartado "*2.6.1 Implementación de un Navigation Controller (Control de Navegación)*", el tipo de conexión "*Push*" es necesario cuando se conecten vistas que implementan un *Navigation Controller*. Ahora se explicarán los otros dos tipos de conexión: *"Modal"* y *"Custom"*.

El tipo de conexión *Modal*, presenta una vista sobrepuesta a la vista que la llama; y es el único que permite seleccionar el tipo de transición. Para poder observar cada tipo de transición existente se deben agregar dos *View Controllers* a un *Storyboard* y conectarlos con un tipo de conexión *Modal.* Para este apartado, se ha creado el siguiente *Storyboard*:

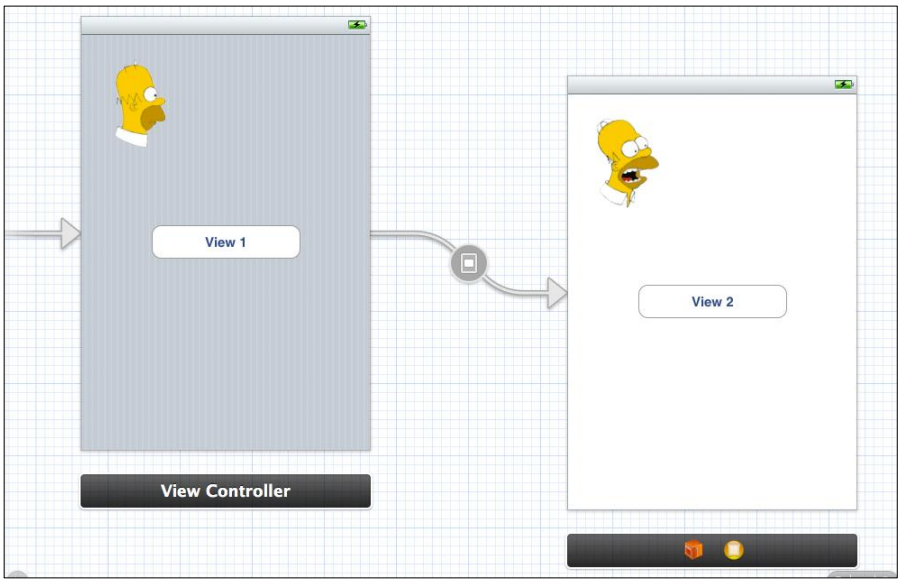


Fig. 2.233: *Storyboard* de la aplicación demostrativa *TransitionsExample*

En este *Storyboard* de la aplicación *TransitionsExample*, la segunda vista es presentada al pulsar el botón de la primera vista. Se ha establecido un tipo de conexión modal entre el botón de la vista 1 y la vista 2. La vista 2 tiene un botón sin ninguna funcionalidad; el cual ha sido colocado únicamente para indicar que esta es la vista 2.

Si se selecciona al *Segue*, se le puede asignar un identificador (*Identifier*), modificar el tipo de conexión (*Style*) y modificar el tipo de transición (*Transition*) como se puede observar a continuación:

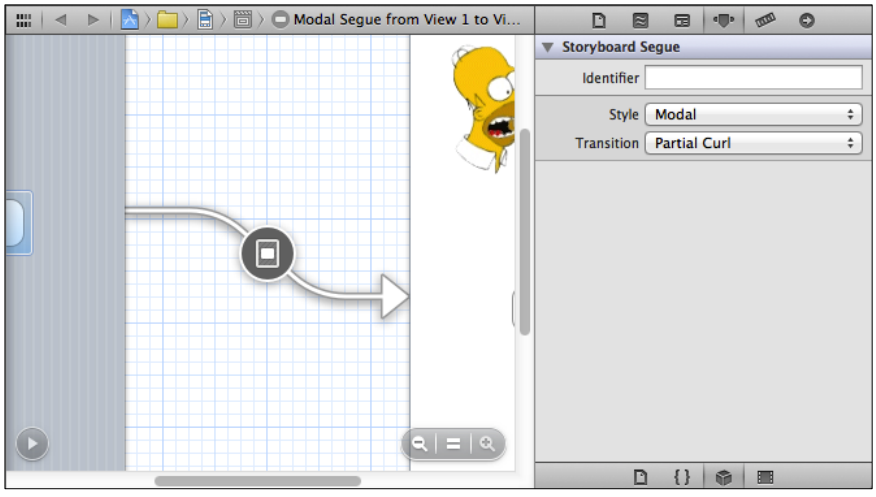


Fig. 2.234: Propiedades de un *Segue*

Al conectar dos vistas con un *Segue* de tipo *Modal*, cuando la primera vista llama a la segunda, ésta se presenta con una transición o animación específica (apartado *Transition* en la Fig. 2.234).

Las transiciones o animaciones tienen una duración de aproximadamente un segundo. A continuación se presentará para cada tipo de animación su descripción y una secuencia de imágenes que la representan:

Cover Vertical: Es un tipo de transición en la cual la segunda vista cubre a la primera realizando una animación desde abajo hacia arriba, así:

Fig. 2.235: Tipo de transición *Cover Vertical*

Flip Horizontal: Es un tipo de transición en la cual se presenta la segunda vista realizando una animación que simula el volteado de la primera; es decir, como si la segunda vista estuviera en el lado posterior de la primera como se puede observar a continuación:

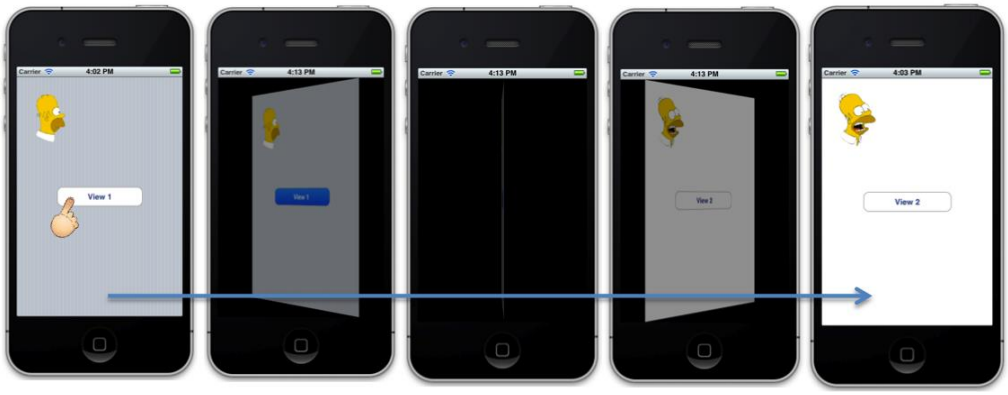


Fig. 2.236: Tipo de transición *Flip Horizontal*

Cross Dissolve: Es un tipo de transición en la cual se presenta la segunda vista realizando una animación que simula la desaparición de la primera; es decir, como si la segunda vista estuviera detrás de la primera y ésta se hubiese desvanecido, así:

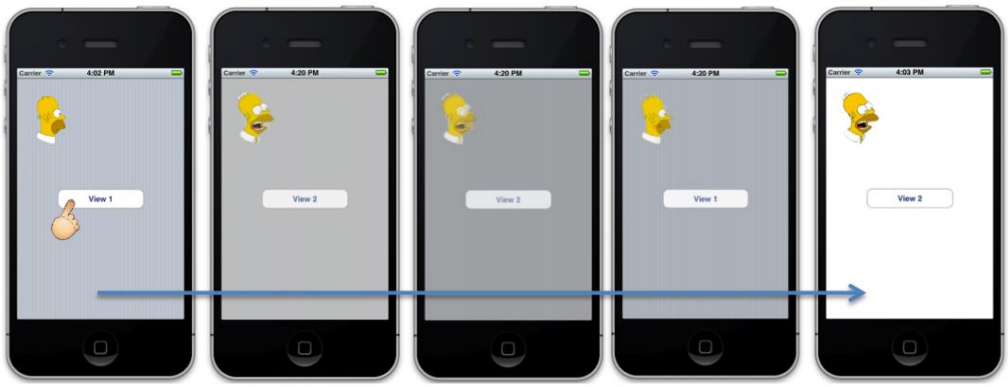


Fig. 2.237: Tipo de transición *Cross Dissolve*

Partial Curl: Es un tipo de transición en la cual se presenta la segunda vista realizando una animación que simula una página; es decir, la primera vista se enrolla hacia un costado para permitir visualizar a la segunda vista. Un aspecto interesante de esta transición es el hecho de que una vez presentada la segunda vista, si se quisiera regresar a la primera, se debe tocar el borde superior en donde se encuentra ésta recogida hacia un costado como se indica a continuación:

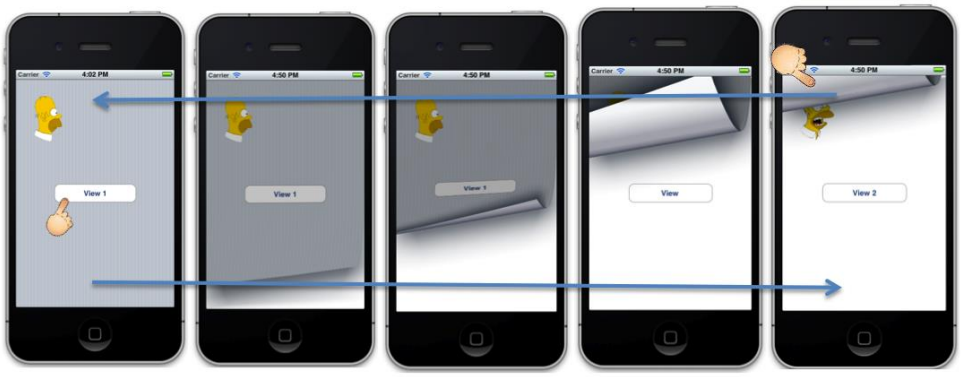


Fig. 2.238: Tipo de transición *Partial Curl*

Se puede ejecutar la aplicación *TransitionsExample* y probar todos los distintos tipos de transiciones. Para cambiar el tipo de transición se debe seleccionar el *Segue* y a continuación seleccionar el tipo de transición en el apartado *Transition* que se puede observar en la Fig. 2.234.

Por otra parte el tipo de conexión "*Custom*" requiere necesariamente de un identificador de *Segue* para poder manejar la transición. Generalmente, se utiliza este tipo de conexión en casos de que se deba cumplir algún requerimiento específico en una vista antes de realizar la transición hacia otra.

2.7 Cómo realizar persistencia de datos

En algún momento, se puede requerir que una aplicación sea capaz de almacenar información y que ésta sea conservada y presentada cada vez que se ejecute esta aplicación. Para la persistencia de datos en aplicaciones de dispositivos *iOS* existen diferentes alternativas; entre estas se encuentra el *framework Core Data,* que es el método recomendado por *Apple Inc*[55]. para la persistencia de datos en sus aplicaciones. *Core Data,* internamente utiliza *SQL Lite,* pero adicionalmente brinda una serie de herramientas que facilitan la creación del modelo de datos y el proceso de gestión de información desde código fuente[56].

Para poder utilizar este *framework* dentro de una aplicación y almacenar datos en la misma, es esencial que primero se comprenda la estructura del *framework.* Su funcionamiento se explicará conforme se implemente *Core Data* en una aplicación.

En los siguientes sub apartados, se explicará inicialmente la estructura de *Core Data*, una vez explicada su estructura y funcionamiento, se creará una aplicación que implemente este *framework* y permita grabar y leer datos almacenados en un modelo de datos.

[55] InfoJobs. (2012). *Persistencia de datos en aplicaciones iOS*. Recuperado el 25 de Enero de 2013, de Persistencia de datos en aplicaciones iOS: http://infojobs.hackathome.com/materiales/recursos-tecnologia/persistencia-de-datos-en-aplicaciones-ios/

[56] Joshua, N. (2011). *Mastering XCode 4: Develop & Design.* Berkeley, California, Estados Unidos: Peachpit Press. p112

2.7.1 Estructura de Core Data

Core Data presenta una arquitectura similar a la siguiente:

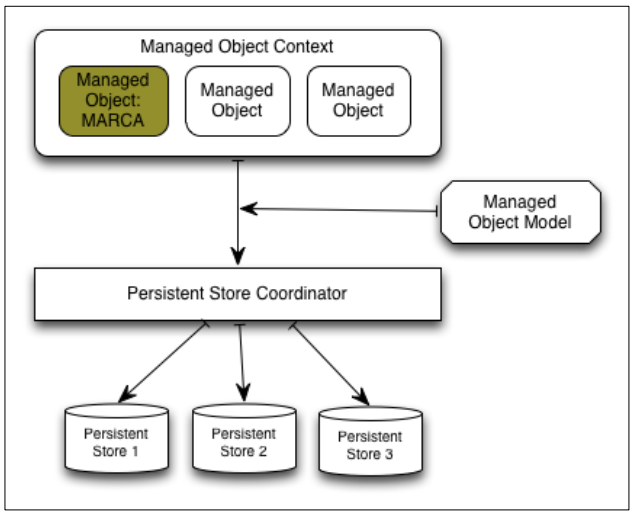


Fig. 2.239: Arquitectura de *CoreData*

Managed Object: Es un objeto gestionado, perteneciente a una clase implementada por el desarrollador. Cada una de las clases implementadas, se situarán como se puede observar en el nivel superior del diagrama. Se implementará una clase "MARCA" con un único atributo: "descripción" para almacenar marcas de vehículos como: *Ford*, *Chevrolet*, *Mazda*, *Hyundai*, entre otros.

Managed Object Context: Memoria temporal que aloja colecciones de *Managed Objects*, los dota de mecanismos de persistencia que ofrece *Core Data*. De esta forma, las operaciones de creación, inserción, modificado y borrado de objetos o *Managed Objects* se realizarán a través de este *Managed Object Context*.

Managed Object Model: Es el encargado de describir la estructura de los objetos; representa el modelo de datos o entidades que definen a los *Managed Objects*. Es representado por el fichero *xcdatamodeld* que puede ser editado mediante *XCode* y será analizado posteriormente en el siguiente sub apartado.[57]

[57] Joshua, N. (2011). *Mastering XCode 4: Develop & Design.* Berkeley, California, Estados Unidos: Peachpit Press. p113, p114

Persistent Store Coordinator: Es el encargado de persistir la información, el desarrollador lo utiliza generalmente solo para indicarle en donde se encuentra el almacenamiento. Maneja una colección de *Persistent Stores.*

Persistent Store: Es un repositorio (*SQL Lite*) en donde se almacena la información de los objetos en sí.[58]

Para una mejor comprensión de lo que representa cada componente de *Core Data*, se presenta el siguiente ejemplo:

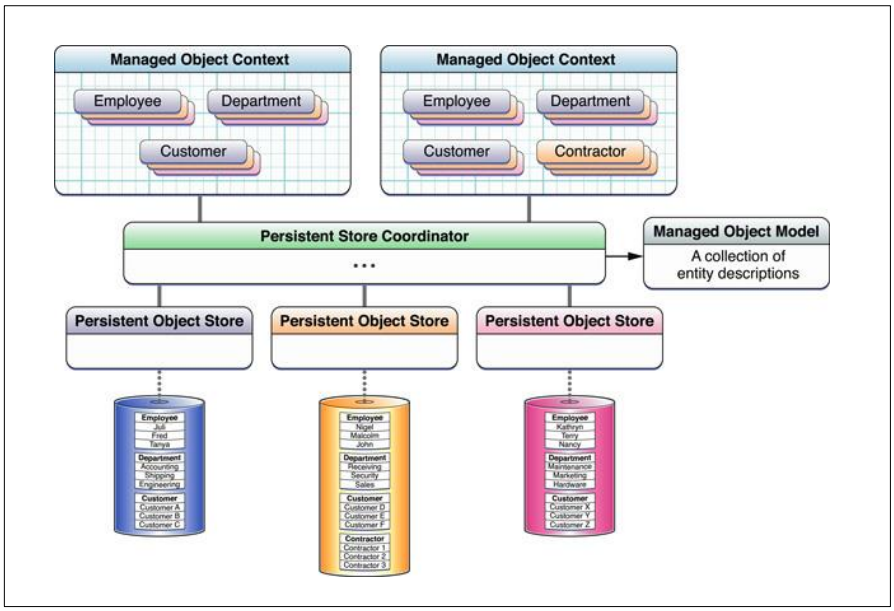


Fig. 2.240: Diagrama ejemplificado de *Core Data*[59]

[58] Joshua, N. (2011). *Mastering XCode 4: Develop & Design.* Berkeley, California, Estados Unidos: Peachpit Press. p113, p114

[59] Programador PHP. (2012). *Persistencia de datos en iOS y Google Web Toolkit*. Recuperado el 25 de Enero de 2013, de Persistencia de datos en iOS y Google Web Toolkit: http://www.programadorphp.org/blog/cursos/persistencia-de-datos-en-ios-y-google-web-toolkit/

2.7.2 Creación de una aplicación que implementa *Core Data*

Cuando se conoce la estructura de *Core Data*, resulta más sencillo comprender el funcionamiento e implementación del mismo. Para este apartado se creará la aplicación *CoreDataExample* en la que se implementará una clase MARCA que almacenará marcas de vehículos como *Chevrolet, Ford,* entre otras.

Funcionamiento: La aplicación permitirá almacenar información persistente mediante *Core Data* de marcas de vehículos. En la interface gráfica existirá botones para grabar, borrar y modificar una marca de vehículo ingresado en un campo de texto. Existirá también un botón para mostrar las marcas que han sido ingresadas, las cuales se presentarán en un *Label* ubicado en la parte inferior (El cual debe ser seleccionado y en su atributo *"Lines"*, asignarle al menos 4 debido a la cantidad de marcas que se pudieran ingresar). La interface gráfica que se acaba de describir es presentada en la Fig. 2.242.

Para empezar se debe crear un nuevo proyecto vacío, si se requiere ayuda se debe referir al apartado *"2.2 Cómo crear un nuevo proyecto"*, nombrarlo *CoreDataExample* y marcar la opción *"Use Core Data"* como se indica a continuación:

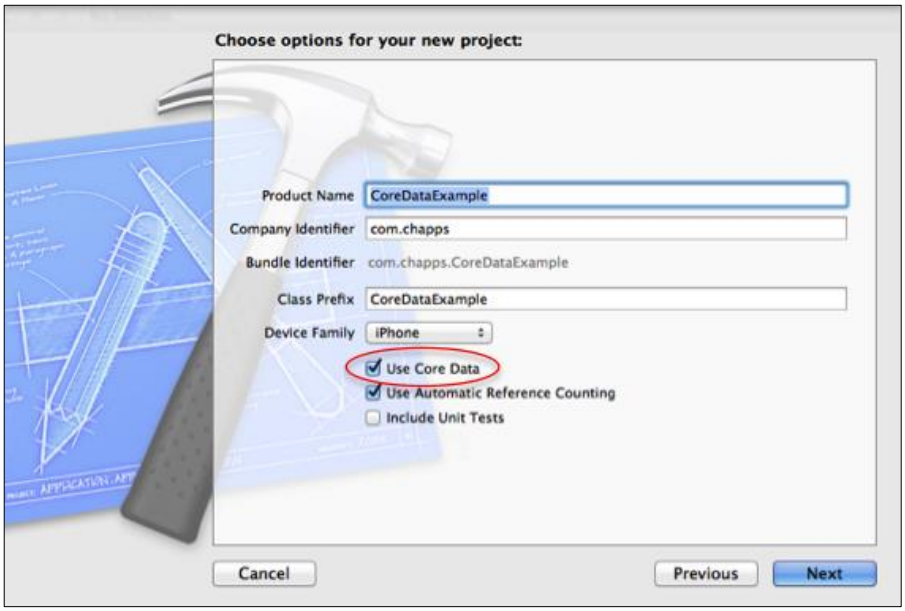


Fig. 2.241: Marcar opción *Use Core Data* al crear el proyecto

Una vez creado el proyecto, se debe agregar un *Storyboard* con un *View Controller*, su respectiva clase controladora y construir la siguiente interface gráfica:

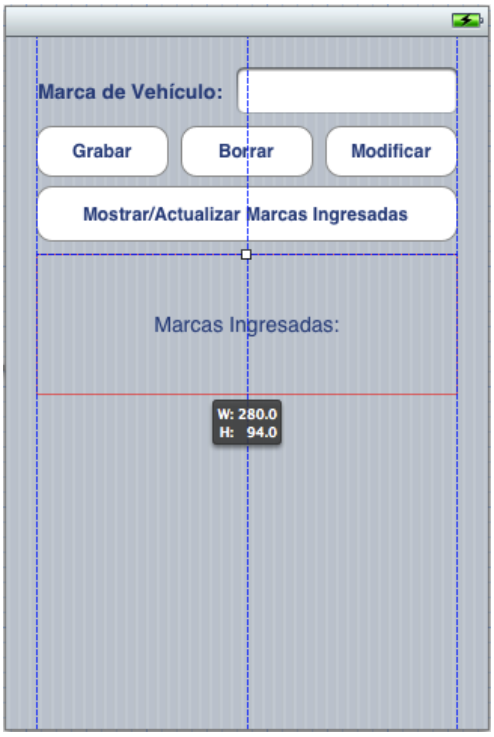


Fig. 2.242: Interface gráfica de *CoreDataExample*

No se debe olvidar de asociar el *View Controller* con su respectiva clase controladora (ver *Fig. 2.114* del *Paso 6* del apartado *"2.4.2 Construcción de una interface gráfica de usuario"*). Tampoco se debe olvidar borrar el evento *loadView* de la clase controladora de la vista (ver *Ajuste 1* del apartado *"2.4.2 Construcción de una interface gráfica de usuario"*).

A continuación se asociarán los objetos con sus respectivas variables y eventos, si se requiere ayuda, se puede referir al apartado *"2.5.1 Asignación de variables a componentes de GUI"* y *"2.5.2 Asignación de métodos a componentes de GUI"*. Esta asignación se realizará de acuerdo a la siguiente tabla:

Componente	Variable (outlet)	Accion (Action)	Nombre
	X	-	txtMarca
Grabar	-	X	grabarMarca
Borrar	-	X	borrarMarca
Modificar	-	X	modificarMarca
Mostrar/Actualizar Marcas Ingresadas	-	X	mostrarMarcas
Marcas Ingresadas:	X	-	lblMarcas

Fig. 2.243: Tabla de variables y acciones a asignar a componentes

Una vez finalizado este proceso de asignación de variables y métodos a los componentes, corresponde analizar el fichero *AppDelegate*, el cual en esta ocasión al haber seleccionado la opción *"Use Core Data"* al momento de crear el proyecto, presenta nuevos métodos y variables que son los siguientes:

Fig. 2.244: Fichero *AppDelegate.h* al implementar *CoreData*

Como se puede observar, el fichero *AppDelegate* al utilizar *Core Data* crea automáticamente tres variables: *managedObjectContext*, *managedObjectModel* y *persistentStoreCoordinator;* los cuales han sido descritos anteriormente. De igual manera, implementa los métodos: *saveContext* para las operaciones de mantenimiento de los registros y *applicationDocumentsDirectory* para indicarle a la aplicación la ruta de almacenamiento de los registros.

Ahora, se analizará el fichero de implementación del *AppDelegate*, el cual además de los métodos presentados en el apartado *"2.3 Análisis de tipos de posibles ficheros de un proyecto"*, ahora implementa algunos otros métodos que son necesarios para el uso de *Core Data*, y son presentados a continuación:

```
- (NSManagedObjectContext *)managedObjectContext
{
    if (__managedObjectContext != nil) {
        return __managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];
    if (coordinator != nil) {
        __managedObjectContext = [[NSManagedObjectContext alloc] init];
        [__managedObjectContext setPersistentStoreCoordinator:coordinator];
    }
    return __managedObjectContext;
}
```

Fig. 2.245: managedObjectContext en fichero AppDelegate.m

Retorna el *Managed Object Context* de la aplicación. Si no existe, es creado y asociado al *Persistent Store Coordinator*.

```objc
- (NSManagedObjectModel *)managedObjectModel
{
    if (__managedObjectModel != nil) {
        return __managedObjectModel;
    }
    NSURL *modelURL = [[NSBundle mainBundle] URLForResource:@"CoreDataExample"
        withExtension:@"momd"];
    __managedObjectModel = [[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL];
    return __managedObjectModel;
}
```

Fig. 2.246: managedObjectModel en fichero AppDelegate.m

Retorna el *Managed Object Model* para la aplicación. Si aun no existe, es creado desde el modelo de la aplicación.

```objc
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator
{
    if (__persistentStoreCoordinator != nil) {
        return __persistentStoreCoordinator;
    }

    NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppendingPathComponent:
        @"CoreDataExample.sqlite"];

    NSError *error = nil;
    __persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:
        [self managedObjectModel]];
    if (![__persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType configuration:nil
        URL:storeURL options:nil error:&error]) {
        /*
         Replace this implementation with code to handle the error appropriately.

         abort() causes the application to generate a crash log and terminate. You should not use
             this function in a shipping application, although it may be useful during development.

         Typical reasons for an error here include:
         * The persistent store is not accessible;
         * The schema for the persistent store is incompatible with current managed object model.
         Check the error message to determine what the actual problem was.


         If the persistent store is not accessible, there is typically something wrong with the file
             path. Often, a file URL is pointing into the application's resources directory instead
             of a writeable directory.

         If you encounter schema incompatibility errors during development, you can reduce their
             frequency by:
         * Simply deleting the existing store:
         [[NSFileManager defaultManager] removeItemAtURL:storeURL error:nil]

         * Performing automatic lightweight migration by passing the following dictionary as the
             options parameter:
         [NSDictionary dictionaryWithObjectsAndKeys:[NSNumber numberWithBool:YES],
             NSMigratePersistentStoresAutomaticallyOption, [NSNumber numberWithBool:YES],
             NSInferMappingModelAutomaticallyOption, nil];

         Lightweight migration will only work for a limited set of schema changes; consult "Core Data
             Model Versioning and Data Migration Programming Guide" for details.

         */
        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }

    return __persistentStoreCoordinator;
}
```

Fig. 2.247: persistentStoreCoordinator en fichero AppDelegate.m

Retorna el *Persistent Store Coordinator*. Si no existe es creado, el almacenamiento de la aplicación es agregado a éste.

```objc
- (NSURL *)applicationDocumentsDirectory
{
    return [[[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
        inDomains:NSUserDomainMask] lastObject];
}
```

Fig. 2.248: applicationDocumentsDirectory en fichero AppDelegate.m

Retorna la ruta o *URL* hasta el directorio de documentos de la aplicación.

```
- (void)saveContext
{
    NSError *error = nil;
    NSManagedObjectContext *managedObjectContext = self.managedObjectContext;
    if (managedObjectContext != nil) {
        if ([managedObjectContext hasChanges] && ![managedObjectContext save:&error]) {
             // Replace this implementation with code to handle the error appropriately
                .
             // abort() causes the application to generate a crash log and terminate.
                 You should not use this function in a shipping application, although
                 it may be useful during development.
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
            abort();
        }
    }
}
```

Fig. 2.249: saveContext en fichero AppDelegate.m

Método que sirve para realizar cualquier tipo de mantenimiento de los registros, ya que como se ha explicado anteriormente, las operaciones de mantenimiento se realizan sobre el *Managed Object Context* y no sobre el registro directamente.

Cabe recalcar que es recomendable no alterar estos métodos autogenerados, a no ser de que se requiera manejar los errores que pudieran ocurrir de alguna manera en especial.

Una vez analizados los métodos automáticamente generados en el fichero *AppDelegate*, corresponde explicar cómo generar el modelo (*CoreDataExample.xcdatamodeld*) o base de datos con el cual trabajará la aplicación.

Para empezar se debe seleccionar el fichero antes mencionado *CoreDataExample.xcdatamodeld* el cual se encuentra en el Área de Navegación del proyecto y también es generado automáticamente al marcar la opción *Use Core Data* al momento de crear la aplicación.

En este archivo, las entidades (*Entity*) son las clases, a las cuales se les puede agregar cada uno de sus atributos. Se agregará la clase “MARCA” con un único atributo “descripción” de tipo *String*.

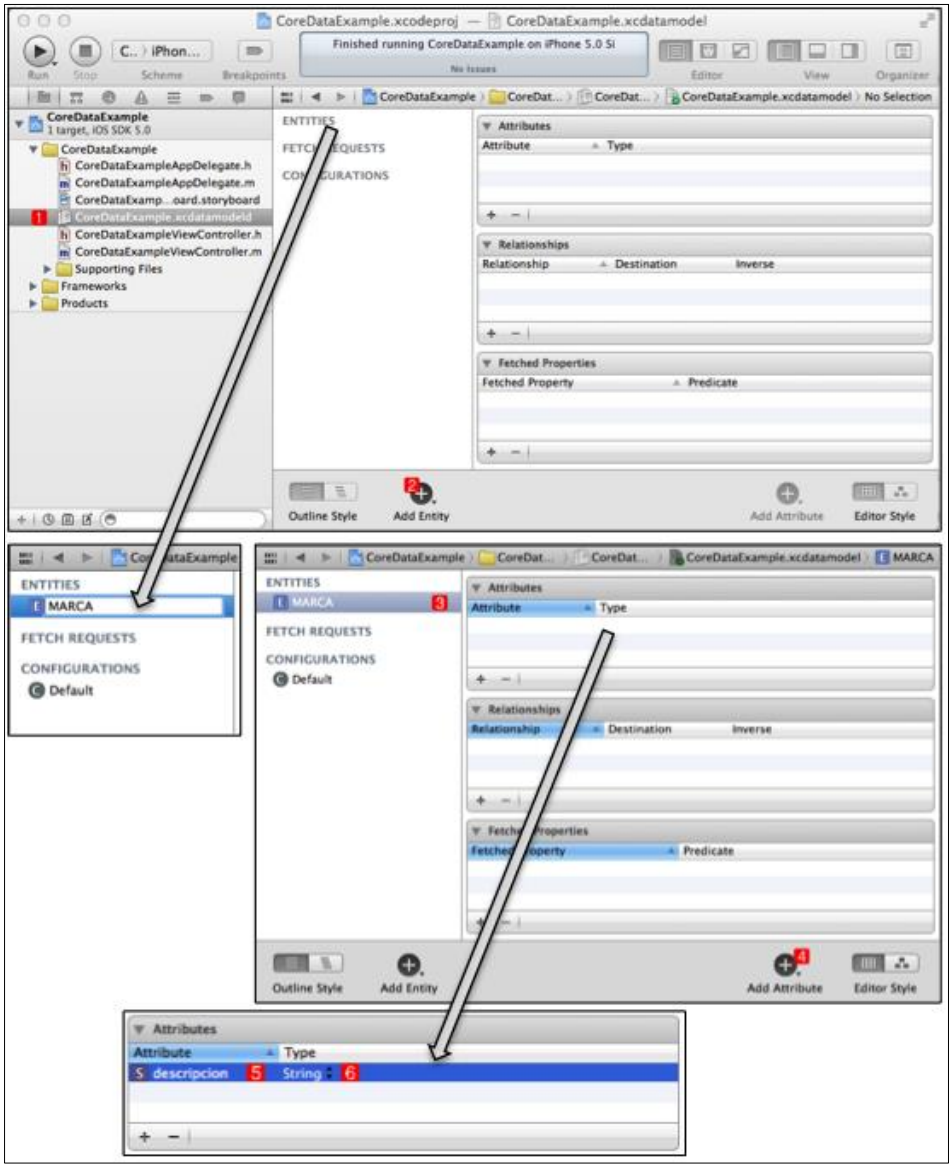


Fig. 2.250: Agregar una clase con sus atributos al modelo de datos

Proceso realizado: Seleccionar el archivo del modelo de datos, agregar una entidad y nombrarla "MARCA". Seleccionar la entidad MARCA y agregar un atributo, nombrarlo "descripción". Seleccionar *String* para su tipo de dato.

Una vez establecido el modelo, se puede presionar el botón del estilo de edición para poder visualizar y editar el modelo de datos de la siguiente forma:

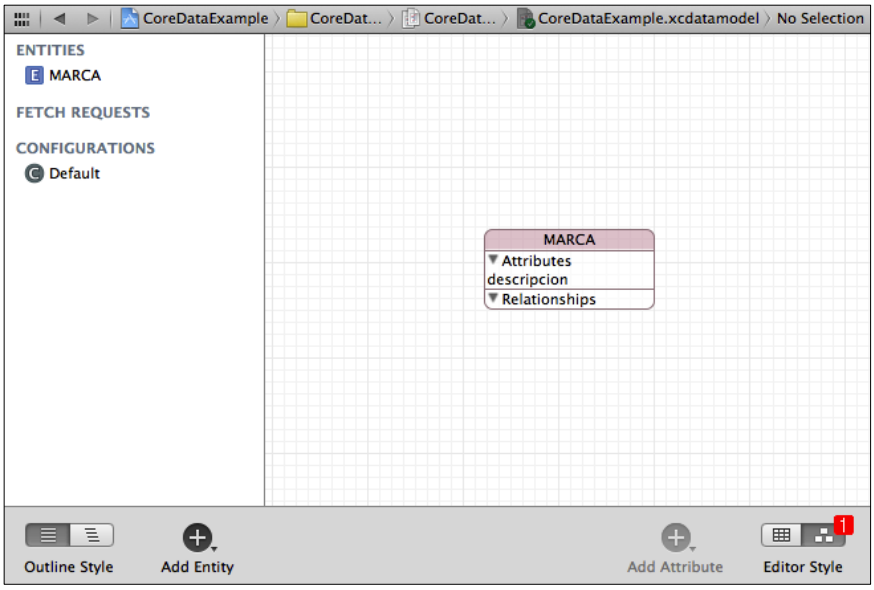


Fig. 2.251: Visualización del modelo de datos establecido.

El siguiente paso consiste en generar los ficheros cabecera e implementación de cada una de las clases especificadas en el modelo, este proceso es relativamente sencillo gracias a la implementación del modelo. Para esto, se debe agregar un nuevo archivo al proyecto: File → New → File como se indica en la Fig. 2.43. En la ventana que se presenta a continuación se debe agregar un *NSManagedObjectSubclass* dentro de la sección *Core Data* así:

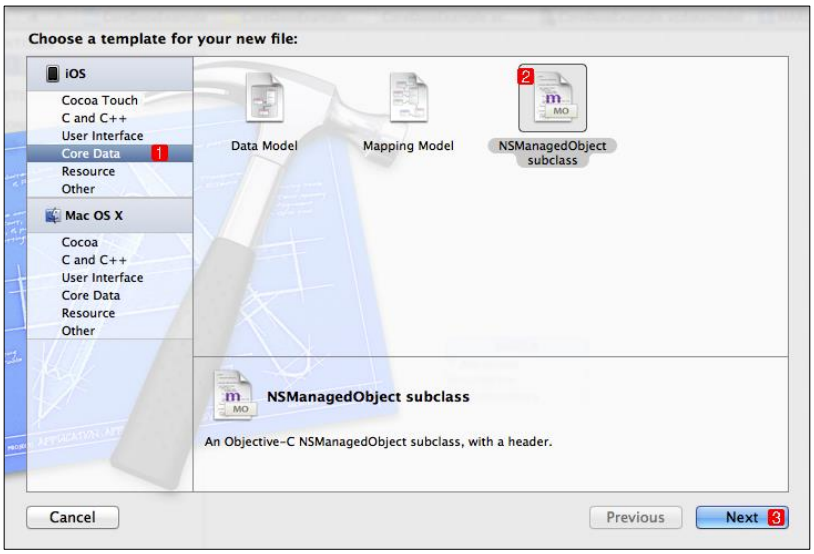


Fig. 2.252: Agregar un fichero *NSManagedObjectSubclass*

En la siguiente ventana se puede seleccionar una ubicación específica para los archivos que se generarán pero no es necesario, por lo que se debe finalizar presionando el botón *"Create"*. Los archivos de la clase MARCA se han agregado al proyecto como se puede observar a continuación:

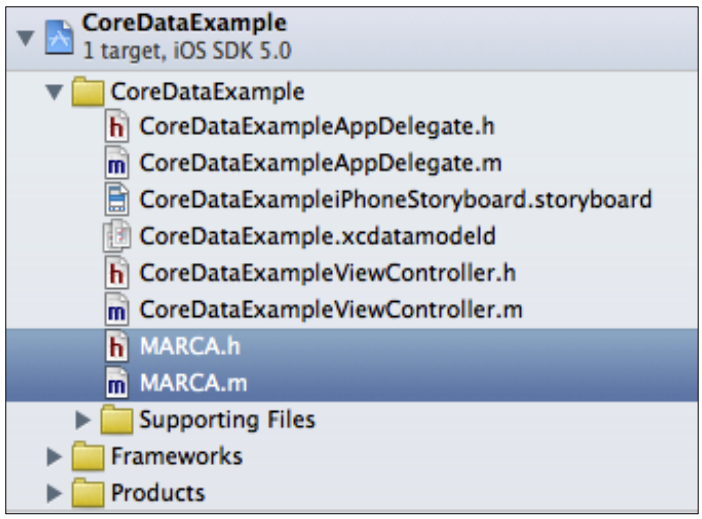


Fig. 2.253: Archivos del proyecto CoreDataExample

El último paso consiste en grabar los datos de forma persistente mediante código fuente. Para lo cual, se debe seleccionar el archivo de implementación del *View Controller* e incluir en la parte superior los archivos cabecera de los ficheros *AppDelegate* y la clase MARCA como se indica a continuación:

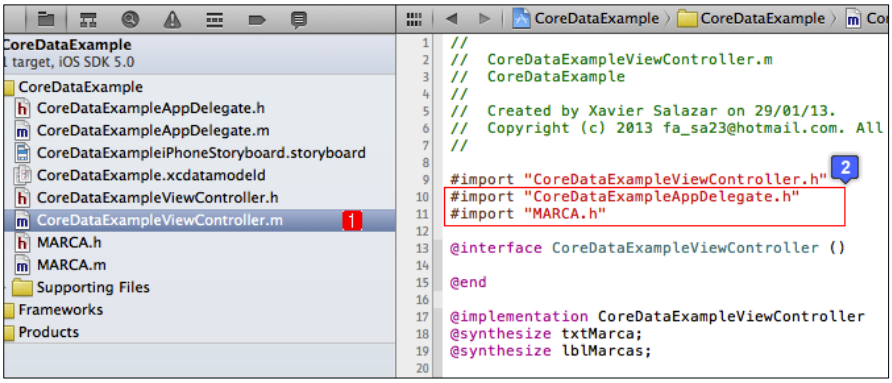


Fig. 2.254: Importar los ficheros cabecera de las clases necesarias

A continuación se implementarán cada uno de los métodos correspondientes a los botones de la interface gráfica de usuario de la Fig. 2.242, para una mejor comprensión de los mismos, se ha comentado cada línea, excepto las que ya han sido comentadas previamente.

Grabar marca de vehículo: El método grabar marca almacena en la base de datos una marca de vehículo escrita en el campo de texto.

```
- (IBAction)grabarMarca:(id)sender {

    // Crear una variable appDelegateOutlet para instanciar el AppDelegate de la aplicación
    CoreDataExampleAppDelegate *appDelegateOutlet = (CoreDataExampleAppDelegate*)
        [[UIApplication sharedApplication] delegate];

    // Crear un objeto de la clase MARCA indicando que será insertado mediante el Managed
        Object Context a la entidad MARCA
    MARCA *marca = [NSEntityDescription insertNewObjectForEntityForName:@"MARCA"
        inManagedObjectContext: [appDelegateOutlet managedObjectContext]];

    // Establecer los atributos del objeto recién creado
    marca.descripcion = txtMarca.text;

    // Realizar la operación de grabado
    NSError *saveError;
    if ([[appDelegateOutlet managedObjectContext] save:&saveError]){
        // Gestion de error de grabado de datos
    }

}
```

Fig. 2.255: Método grabar marca llamado al presionar el botón “Grabar”

Tanto para borrar, modificar y mostrar las marcas ingresadas, se debe realizar una consulta en la base de datos; la estructura de una consulta será explicada más adelante; después de haber implementado cada método para complementar su comprensión.

Borrar marca de vehículo: El método borrar marca realiza una consulta en la base de datos de la marca escrita en el campo de texto; en caso de encontrarla, la elimina.

```
- (IBAction)borrarMarca:(id)sender {

    CoreDataExampleAppDelegate *appDelegateOutlet = (CoreDataExampleAppDelegate*)
        [[UIApplication sharedApplication] delegate];

    //Se crea un objeto NSFetchRequest para realizar la consulta
    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

    //Se definine la entidad o tabla donde se realizará la consulta
    NSEntityDescription *entity = [NSEntityDescription entityForName:@"MARCA"
        inManagedObjectContext:appDelegateOutlet.managedObjectContext];

    // Se crea un predicado que se debe cumplir
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"descripcion=%@",txtMarca.text];

    //Se asigna la entidad y el predicado al NSFetchRequest

    [fetchRequest setEntity:entity];
    [fetchRequest setPredicate:predicate];

    // Se realiza la consulta de los registros y se los almacena en un NSArray
    NSError *errorRequest;
    NSArray *items = [appDelegateOutlet.managedObjectContext
                      executeFetchRequest:fetchRequest error:&errorRequest];

    // Se elimina el registro en caso de que la consulta haya tenido al menos un resultado
    if ([items count]>0)
        [appDelegateOutlet.managedObjectContext deleteObject:[items objectAtIndex:0]];

    NSError *saveError;
    if (![appDelegateOutlet.managedObjectContext save:&saveError]) {
        NSLog(@"Update or delete failed: %@", saveError);
    }

}
```

Fig. 2.256: Método borrar marca llamado al presionar el botón “Borrar”

Modificar marca de vehículo: El método modificar marca realiza una consulta en la base de datos de la marca escrita en el campo de texto; en el caso de encontrarla, la reemplaza por la marca “*Ford*”, se debe tomar en cuenta que podría ser reemplazada por cualquier variable de tipo *NSString*.

```
- (IBAction)modificarMarca:(id)sender {

    CoreDataExampleAppDelegate *appDelegateOutlet = (CoreDataExampleAppDelegate*)
        [[UIApplication sharedApplication] delegate];

    //Se crea un objeto NSFetchRequest para realizar la consulta
    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

    //Se definine la entidad o tabla donde se realizará la consulta
    NSEntityDescription *entity = [NSEntityDescription entityForName:@"MARCA"
        inManagedObjectContext:appDelegateOutlet.managedObjectContext];

    // Se crea un predicado que se debe cumplir
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"descripcion=%@",txtMarca.text];

    //Se asigna la entidad y el predicado al NSFetchRequest

    [fetchRequest setEntity:entity];
    [fetchRequest setPredicate:predicate];

    // Se realiza la consulta de los registros y se los almacena en un NSArray
    NSError *errorRequest;
    NSArray *items = [appDelegateOutlet.managedObjectContext
                      executeFetchRequest:fetchRequest error:&errorRequest];

    //Se modifica el registro en caso de que la consulta haya tenido al menos un resultado, lo
        que hace es reemplazar la marca encontrada por la marca ''Ford''
    if ([items count]>0)
        [[items objectAtIndex:0] setDescripcion:@"Ford"];

    NSError *saveError;
    if (![appDelegateOutlet.managedObjectContext save:&saveError]) {
        NSLog(@"Update or delete failed: %@", saveError);
    }

}
```

Fig. 2.257: Método modificar marca llamado al presionar el botón “Modificar”

Mostrar/Actualizar marcas de vehículos ingresadas: El método mostrar marcas realiza una consulta en la base de datos de todas la marcas ingresadas, las ordena y las concatena separadas por el carácter "/". El resultado es asignado a la etiqueta *lblMarcas.*

```
- (IBAction)mostrarMarcas:(id)sender {

    CoreDataExampleAppDelegate *appDelegateOutlet = (CoreDataExampleAppDelegate*)
        [[UIApplication sharedApplication] delegate];

    // Reiniciar el texto del Label lblMarcas
    lblMarcas.text = @"Marcas Ingresadas: ";

    // Se crea un objeto NSFetchRequest para realizar la consulta
    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];

    // Se definine la entidad o tabla donde se realizará la consulta
    NSEntityDescription *entity = [NSEntityDescription entityForName:@"MARCA"
        inManagedObjectContext:appDelegateOutlet.managedObjectContext];

    // Se asigna la entidad al NSFetchRequest
    [fetchRequest setEntity:entity];

    // Se define el ordenado de los resultados
    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"descripcion"
        ascending:YES];
    NSArray *sortDescriptorsArray = [NSArray arrayWithObject:sortDescriptor];
    [fetchRequest setSortDescriptors:sortDescriptorsArray];

    // Ejecución de la consulta
    NSError *error;
    NSArray *result = [appDelegateOutlet.managedObjectContext executeFetchRequest:fetchRequest
        error:&error];

    // Bucle para concatenar todos los resultados de la consulta en una sola cadena y asignarla
        al texto de lblMarcas, los resultados son concatenados separados por "/"
    NSArray *marcasIngresadas = result;
    for (int i=0; i<[marcasIngresadas count]; i++) {
        lblMarcas.text = [lblMarcas.text stringByAppendingString:@"/"];
        lblMarcas.text = [lblMarcas.text stringByAppendingString:[[marcasIngresadas
            objectAtIndex:i] descripcion]];
    }

}
```

Fig. 2.258: Método mostrar marcas llamado al presionar el botón "Mostrar/Actualizar marcas ingresadas"

Consultas: Para poder realizar una consulta en la base de datos, se requiere comprender algunos conceptos básicos, así como la estructura de una consulta.

Fetch Request (NSFetchRequest): Es una búsqueda de *Managed Objects* sobre un *Managed Object Context*. Se pueden extraer todos los elementos con únicamente especificar el nombre de la entidad; se pueden especificar filtros (*NSPredicate*)[60], el orden de los datos (*NSSortDescriptor*)[61], entre otros. El resultado de la ejecución de esta consulta retorna un arreglo de *Managed Objects.*

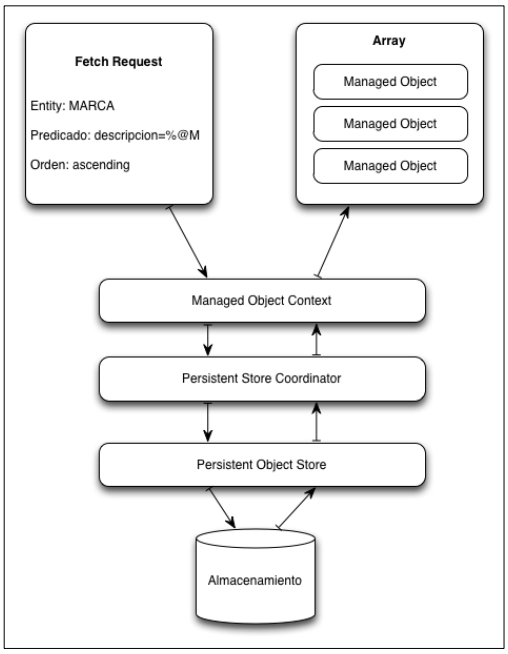


Fig. 2.259: Estructura de una consulta

Una vez implementados los métodos para cada botón de la interface gráfica de usuario, se puede proceder a ejecutar la aplicación. Antes de hacerlo, se deben borrar las líneas de código innecesarias del fichero *AppDelegate* (ver "*Ajuste 2*" dentro del apartado

[60] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p478, p479

[61] Microedition Biz. (2012). *Core Data (1) - Conceptos iniciales*. Recuperado el 30 de Enero de 2013, de Core Data (1) - Conceptos iniciales: http://www.microedition.biz/blog/?p=549

"*2.4.2 Construcción de una interface gráfica de usuario*") y asociar el *Storyboard* agregado, a la aplicación (ver "*2.3.1 Cómo agregar un fichero a un proyecto*", sub apartado "*A) Agregar un Storyboard*", *Fig. 2.48*).

Al ejecutar la aplicación se pueden agregar, modificar y borrar registros de marcas de vehículos como se indica en la Fig. 2.260; en la cual se ha seguido la siguiente secuencia:

- Grabar un registro *"Hyundai"*.
- Mostrar marcas ingresadas.
- Grabar un registro *"Nissan"*.
- Mostrar marcas ingresadas.
- Modificar el registro *"Nissan"* (por *"Ford"* de acuerdo al código fuente).
- Mostrar marcas ingresadas.
- Borrar el registro *"Hyundai"*
- Mostrar marcas ingresadas.

En cualquier momento, se pueden visualizar los registros contenidos en la base de datos, mediante cualquier gestor de base de datos *SQLite,* se usará *MesaSQLite* para demostrarlo.

La ubicación de la base de datos de la aplicación es la siguiente: Usuario/Librería/ApplicationSupport/iPhone Simulator/iOSFirmware/Applications/**6E829A85-12C0-491B-9630-5585CB89DCA8**/Documents

Fig. 2.260: Funcionamiento de la aplicación *CoreDateExample*

La porción **6E829A85-12C0-491B-9630-5585CB89DCA8** de la ruta anterior es exclusiva para cada aplicación que se desarrolle por lo que se debe buscar la correspondiente a la aplicación desarrollada en el computador en donde se la ha implementado.

Esta porción de la ruta anterior cambiará cada vez que se cree una base de datos.

Al abrir la aplicación *MesaSQLite* y seleccionar la base de datos que se desea visualizar, se presentará la siguiente ventana:

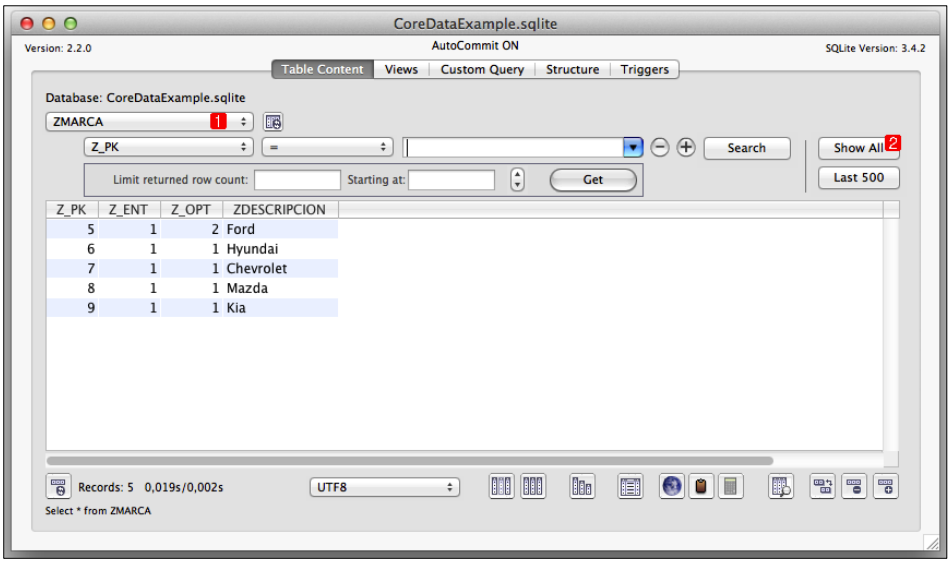


Fig. 2.261: Visualización de la base de datos creada desde la *iOS App* en *MesaSQLite*

Importante: Si en algún momento, se altera el modelo de datos, es decir se crea una nueva tabla, o se añade un nuevo atributo a una tabla; antes de ejecutar la aplicación, se deben volver a generar los ficheros de cada clase (ver Fig. 2.252 y Fig. 2.253). Al ejecutar la aplicación, cuando esta haga referencia a la base de datos, se presentará un error en la consola similar al siguiente:

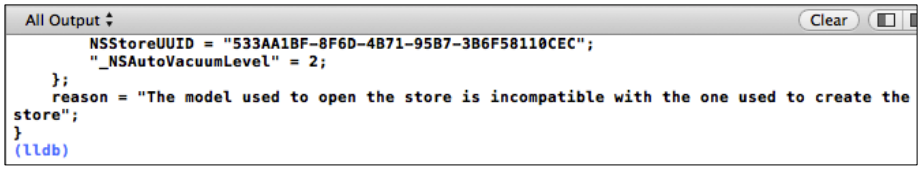


Fig. 2.262: Error en consola al ejecutar aplicación con modelo alterado

Para evitar este error, se debe eliminar la aplicación en el simulador de la misma forma como se lo hace en un dispositivo *iOS* físico: Mantener presionado el ícono de la aplicación hasta que aparezca el botón "x" en la esquina superior del mismo, finalizar presionándolo, como se indica a continuación:

Fig. 2.263: Eliminar una aplicación

La siguiente vez que se ejecute la aplicación ya no se presentará el error en consola.

2.8 Cómo conectar una aplicación con redes sociales

Si se desea que la aplicación tenga una buena acogida y logre una cantidad considerable de descargas o compras, la interacción de ésta con las redes sociales populares de la actualidad es fundamental. De esta forma se conseguirá su popularización.

Es común en la actualidad observar en redes sociales como muchos de los usuarios utilizan algunas aplicaciones como: "*Shazam*", "*SongPop*", "*Instagram*" entre otras. Esto despierta curiosidad en otros usuarios que también desean probarlas.

Incluso, algunas aplicaciones permiten importar las fotos de perfil de redes sociales de los usuarios, previa autorización de los mismos. Esto es común en los juegos, los cuales han adoptado un entorno mas agradable al permitir a los usuarios competir contra amigos de sus redes sociales.

Para conseguir la popularización de la aplicación, ésta debe estar en la capacidad de publicar en el perfil de al menos una de las redes sociales del usuario. Un factor muy importante es el de consultar al usuario antes de publicar algo en su perfil, ya que el publicar sin la autorización del mismo podría conllevar a la eliminación de la aplicación. Se deben evitar las publicaciones frecuentes debido a que podrían llegar a ser molestosas tanto para el usuario como para sus conexiones de la red social (amigos, seguidores, etc.)

Para conectar una aplicación con redes sociales se deben utilizar los *frameworks* respectivos, los cuales deben ser previamente importados al proyecto. Las redes sociales más populares, proveen estos frameworks, algunos de éstos pueden ser encontrados dentro del mismo *IDE XCode*.

2.8.1 Cómo añadir frameworks

El proceso de agregado de un framework al proyecto es relativamente sencillo. Para esto, dentro de cualquier proyecto abierto se debe seguir la siguiente secuencia:

Seleccionar el proyecto, después seleccionar el target u objetivo, acto seguido seleccionar *Summary* y desplazarse hacia abajo hasta encontrar la sección de *Frameworks* y librerías asociadas al proyecto. Presionar el botón para agregar un nuevo *framework* o librería como se indica a continuación:

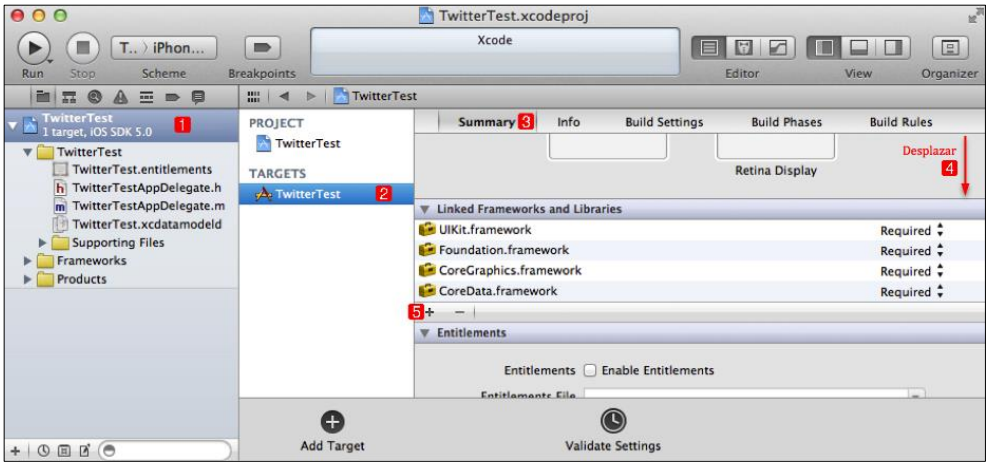


Fig. 2.264: Forma de agregar un nuevo framework al proyecto

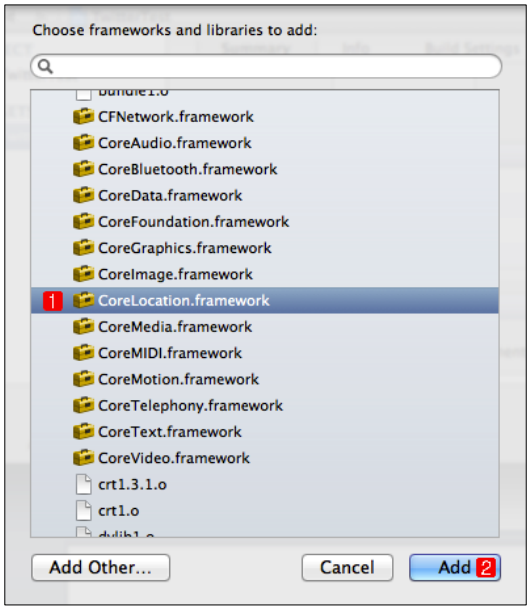


Fig. 2.265: Listado de frameworks y librerías disponibles

Al hacerlo, se presenta la lista de frameworks y librerías disponibles para agregar al proyecto, de las cuales se debe seleccionar el *framework* deseado (por ejemplo *CoreLocation*) y agregarlo; como se indica en la Fig. 2.265.

Finalmente el *framework* es agregado al proyecto, se lo puede ver tanto en sus propiedades (1) como en el Área de Navegación (2), de la siguiente figura:

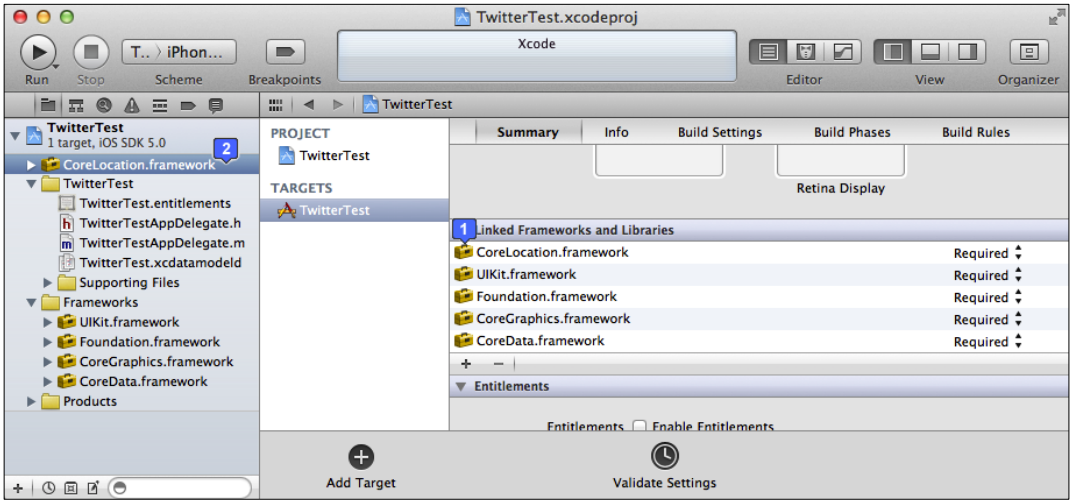


Fig. 2.266: *Framework CoreLocation* agregado con éxito

Para mantener el orden de los componentes del proyecto, se puede colocar el *framework* dentro de la carpeta *frameworks*, así:

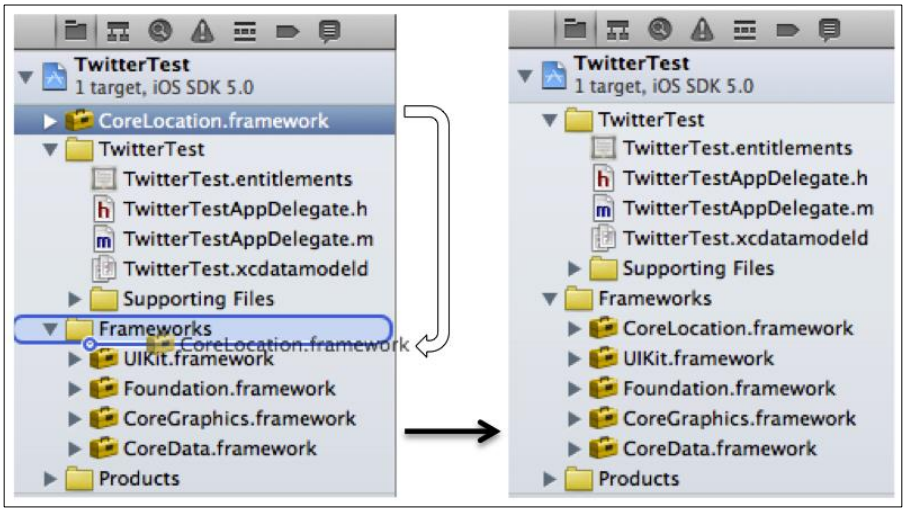


Fig. 2.267: Organización de frameworks del proyecto

En caso de no encontrar el *framework* necesario dentro del listado de librerías y *frameworks* de la Fig. 2.265, puede ser descargado de internet.

Si un *framework* es descargado de internet, para agregarlo al proyecto, basta con arrastrarlo desde su ubicación hasta los ficheros del proyecto o Área de Navegación, como se indica a continuación:

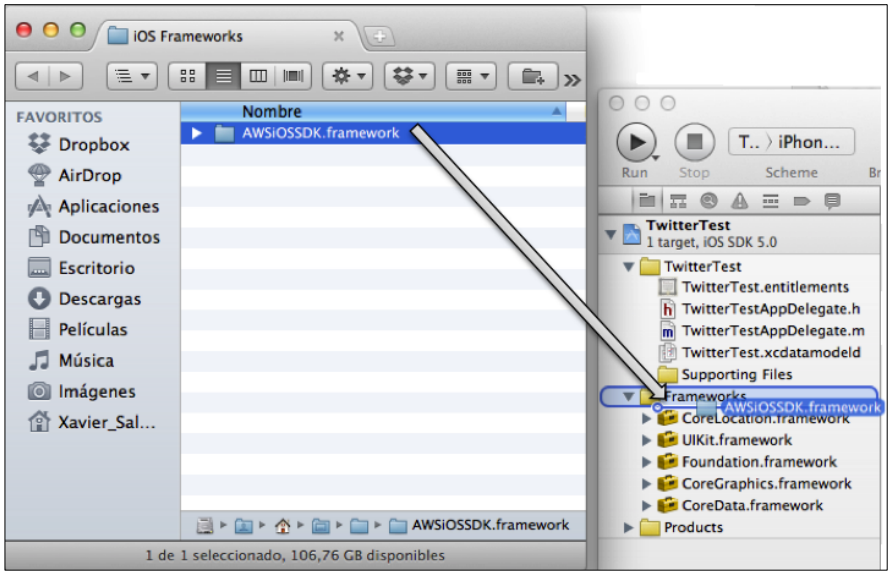


Fig. 2.268: Arrastrar al proyecto un *framework* descargado de internet

En la ventana que se presenta a continuación, se debe marcar la opción para copiar elementos al proyecto y presionar el botón "Finish".

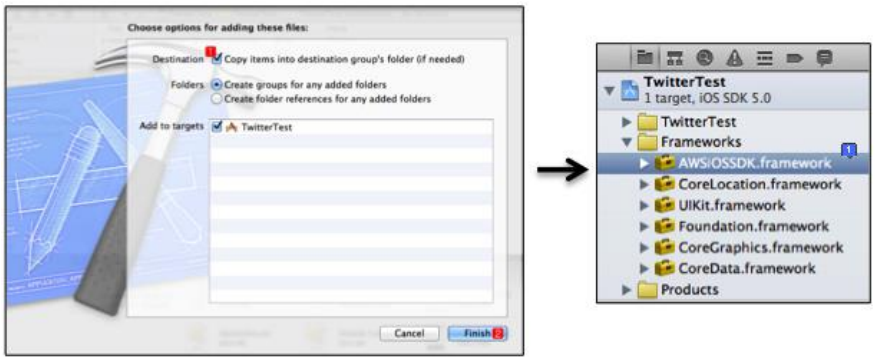


Fig. 2.269: Framework agregado exitosamente

2.8.1.1 Análisis de frameworks comunes

Se pueden encontrar varios *frameworks* desarrollados para el *IDE XCode* en lenguaje *Objective-C* con distintos fines, los mas importantes e incluidos por defecto en cualquier aplicación creada son: *UIKit*, *Foundation* y *CoreGraphics* los cuales fueron analizados en el apartado "*2.3 Análisis de tipos de posibles ficheros de un proyecto*".

Existen muchos otros *frameworks* que facilitan el desarrollo de aplicaciones, debido a que permiten realizar funciones determinadas dentro de una aplicación sin necesidad de escribir código sino únicamente importar y emplear los métodos del *framework* correspondiente. Entre estos *frameworks,* se pueden destacar: *AppiRater, UniqueIdentifier*, *Twitter* y *Facebook*; los cuales serán mencionados en capítulos posteriores del tutorial. Entre otros de los *frameworks* más importantes y populares, se pueden destacar los siguientes:

Framework: *CoreLocation*:

Fig. 2.270: *CoreLocation* solicitud para usar la localización del usuario

Es el encargado de proveer la localización e información de rumbo a las aplicaciones. Para la información de localización actual de longitud y latitud del usuario, el *framework* hace uso del *GPS* interno, la información de la operadora, o información del *router WiFi* al cual se encuentra conectado el dispositivo *iOS*[62]. Se puede incorporar

[62] Apple. (2012). *Core Services Layes*. Recuperado el 25 de Febrero de 2013, de Core Services Layes: http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/CoreServicesLayer/CoreServicesLayer.html#//apple_ref/doc/uid/TP40007898-CH10-SW3

esta tecnología a las aplicaciones para proveer información basada en localización a los usuarios; por ejemplo, presentar los restaurantes más cercanos de acuerdo a la localización actual del usuario.

Mediante este *framework*, existe también la posibilidad de definir regiones geográficas y monitorizar cuando el usuario cruce los límites de estas regiones predefinidas[63]. Las referencias u objetos más relevantes de esta clase son:

CLLocationManager: Se utiliza una instancia de esta clase para establecer los parámetros que determinarán cuando se deben enviar los eventos de localización y enrumbado del dispositivo, y; cuando se debe iniciar y detener el envío de estos eventos.[64]

CLLocation: Es generado por un *CLLocationManager* cuyo objetivo es el de representar los datos de localización. Este objeto incorpora las coordenadas geográficas y la altitud de la localización del dispositivo junto con valores que indican la precisión y tiempo o momento de realizado de éstas las mediciones. Reporta también información de velocidad y rumbo en el cual se está moviendo el dispositivo.[65]

CLRegion: Esta clase define un área geográfica que puede ser rastreada; al momento que el usuario cruce los bordes de esta región, el *CLLocationManager* es notificado.[66]

---

[63] Apple. (2012). *Core Location Framework Reference*. Recuperado el 25 de Febrero de 2013, de Core Location Framework Reference: http://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CoreLocation_Framework/_index.html#//apple_ref/doc/uid/TP40007123

[64] Apple. (2012). *CLLocationManager Class Reference*. Recuperado el 25 de Febrero de 2013, de CLLocationManager Class Reference: http://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CLLocationManager_Class/CLLocationManager/CLLocationManager.html#//apple_ref/doc/uid/TP40007125

[65] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p635

[66] Apple. (2012). *CLRegion Class Reference*. Recuperado el 25 de Febrero de 2013, de CLRegion Class Reference: http://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CLRegion_class/Reference/Reference.html#//apple_ref/doc/uid/TP40009575

Framework CoreData:

Fig. 2.271: Framework CoreData

Como se ha indicado anteriormente, es un *framework* que permite administrar un modelo de datos dentro de una aplicación. Para un análisis profundo de este framework, se puede referir al apartado *"2.7 Cómo realizar persistencia de datos"*

Framework MapKit:

Fig. 2.272: *MapKit* con puntos de interés y localización del usuario

Provee una interface desplazable de un mapa que puede ser integrado dentro de una jerarquía de vistas. Se puede utilizar este mapa para proveer direcciones o resaltar puntos de interés. Las aplicaciones pueden manejar los atributos del mapa de manera

programada o dejar que el usuario interactúe libremente sobre este. Las referencias u objetos más relevantes de esta clase son:

MKMapView: Objeto que provee una interface de mapa integrable, similar a la de la aplicación incluida por defecto en el sistema operativo de *iOS* "Mapas". Se puede utilizar esta clase para presentar información dentro del mapa y manipular sus contenidos desde la aplicación. De igual manera, se puede centrar el mapa en una coordenada dada, especificar el área que se desea mostrar y anotar el mapa con información personalizada.

MKPointAnnotation: Clase que define un objeto anotación localizado en un punto especificado; se puede utilizar esta clase cuando se desee asociar un punto en el mapa con un título.

MKUserLocation: Clase que define un tipo de anotación específica que identifica la localización actual del usuario.

MKPlaceMark: Objeto (o anotación) que almacena información de un lugar con una latitud y longitud dada. Esta información incluye datos como país, estado, ciudad y dirección asociado a las coordenadas.[67]

[67] Apple. (2012). *MKMapView Class Reference*. Recuperado el 25 de Febrero de 2013, de MKMapView Class Reference: http://developer.apple.com/library/ios/#documentation/MapKit/Reference/MKMapView_Class/MKMapView/MKMapView.html#//apple_ref/doc/uid/TP40008205

2.8.2 Implementación del framework de Twitter

Establecer una conexión entre una aplicación *iOS* y la red social *Twitter* es relativamente sencillo, debido a que el *framework* de esta red social, se encuentra incluido por defecto dentro del listado de *frameworks* y librerías disponibles en el *IDE XCode.*

Para desarrollar una aplicación que sea capaz de enviar un *tweet* desde el perfil del usuario que la utilice, lo que se debe hacer es añadir el *framework* de *Twitter* a un proyecto (se recomienda generar un proyecto con la plantilla *Single View Application* denominado *TwitterTest*), si se requiere ayuda se puede referir a los apartados *"2.2 Cómo crear un nuevo proyecto"* y *"2.8.1 Cómo añadir frameworks"*.

Hasta este momento el proyecto consta de los siguientes ficheros:

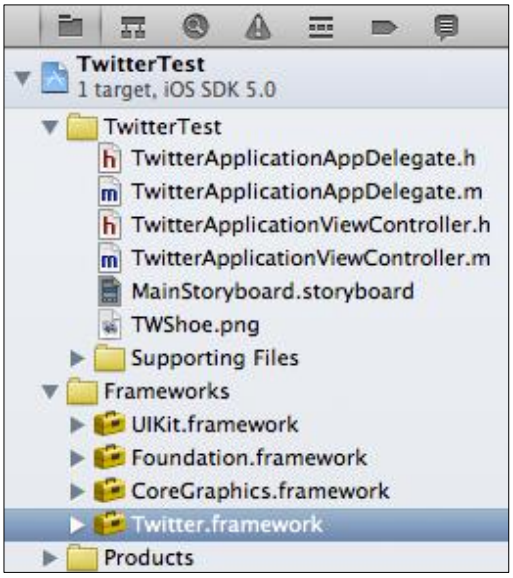


Fig. 2.273: Ficheros de la aplicación *TwitterTest*

Como se puede observar, también se ha agregado un archivo imagen a la aplicación (*TWShoe.png*) cuyas dimensiones son irrelevantes; para enviarla a través de un *tweet.*

El siguiente paso consiste en agregar un botón al *View Controller* del *Storyboard*, el cual, al ser presionado presentará una ventana modal con el contenido del *tweet* que se desea enviar. Asignar el método *enviarTweet* al botón. Si se requiere ayuda se puede

referir al apartado *"2.5.2 Asignación de métodos a componentes de GUI"*. El *View Controller* debe ser similar al siguiente:

Fig. 2.274: View Controller de TwitterTest

A continuación se escribirá código para finalizar la implementación de la aplicación conectada con *Twitter*. El primer paso consiste en abrir el fichero cabecera del *View Controller* para importar el *framework* al mismo, y generar una variable que maneje la vista previa del *tweet*, así como sus contenidos.

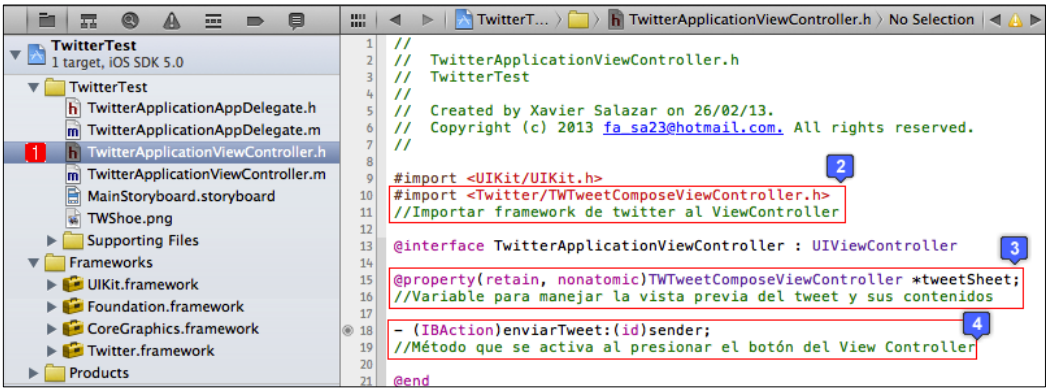


Fig. 2.275: Fichero cabecera del *View Controller* de *TwitterTest*

El siguiente paso consiste en implementar el método *enviarTweet* asignado previamente al botón, dentro del archivo de implementación. Se implementará también un método que permita construir la vista previa del *tweet* (llamado *buildTweetSheet*) así:

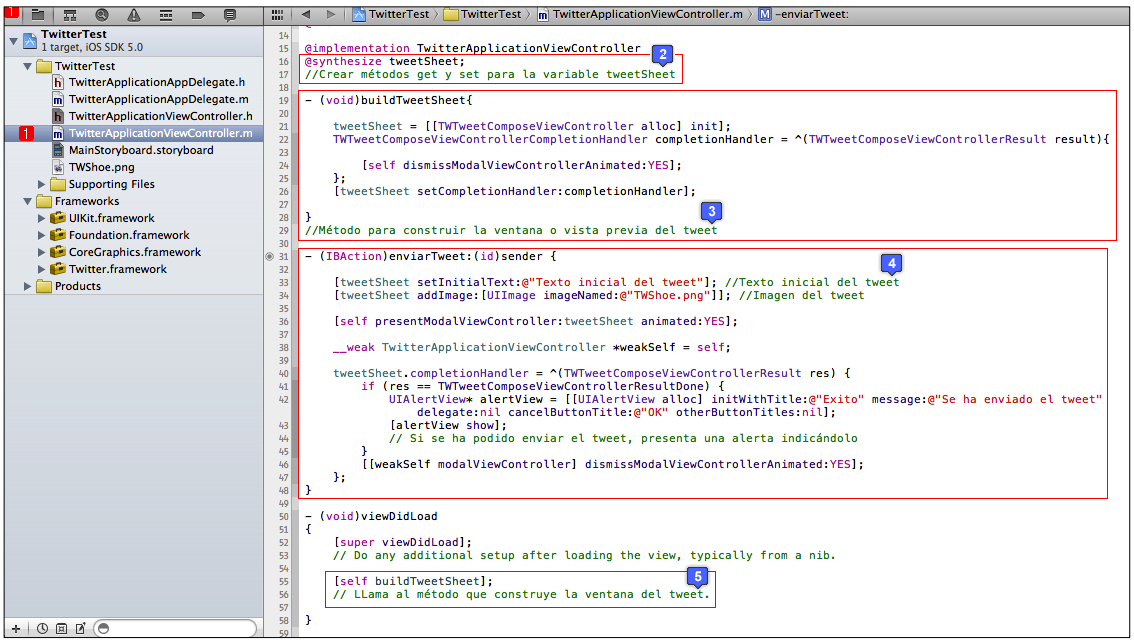


Fig. 2.276: Fichero de implementación del *View Controller* de *TwitterTest*

Como se puede observar, primero se hizo el respectivo *@synthesize* de la variable *tweetSheet* declarada en el archivo cabecera, se implementó el método para construir la vista previa del *tweet buildTweetSheet*, el cual será llamado en el evento *viewDidLoad*. También se implementó el método *enviarTweet* que es llamado cuando se presiona el único botón de la vista.

Ahora se puede proceder a ejecutar la aplicación en el simulador *iOS*, al presionar el único botón de la vista, se presenta la vista previa del *tweet* con su texto inicial y la imagen adjunta. En caso de no tener ninguna cuenta de *Twitter* asociada al dispositivo *iOS* del usuario, se presentará la ventana de ajustes para que asocie una cuenta y al volver a ejecutar la aplicación, se podrá enviar el *tweet* sin problemas, como se indica a continuación:

Fig. 2.277: Aplicación *TwitterTest* en ejecución

2.8.3 Implementación del framework de Facebook

A partir del lanzamiento del sistema operativo *iOS* 6, la integración de una aplicación con la red social *Facebook* es relativamente sencillo, debido a que este sistema operativo permite la comunicación del dispositivo *iOS* directamente con esta red social. La diferencia con la integración con la red social *Twitter* del apartado anterior es que no existe un *framework* de *Facebook* integrado directamente dentro del *IDE XCode*.

La aplicación que se desarrollará en el presente apartado, requiere de *XCode* 4.6, el cual incluye un simulador *iOS* que permite asociar una cuenta de usuario de *Facebook* dentro de sus ajustes, como se ha realizado con la red social *Twitter* en el apartado anterior.

Para desarrollar una aplicación que sea capaz de publicar en *Facebook* desde el perfil del usuario que la utilice, lo que se debe hacer primero es descargar el *framework* de *Facebook* para *iOS*, el cual se puede encontrar en https://developers.facebook.com/ios/

Luego de abrir el enlace se debe presionar el botón verde para descargar el *SDK* para *iOS* de *Facebook*. Se descargará un paquete (*.pkg)*, el cual debe ser instalado.

Fig. 2.278: Descargar e instalar el paquete *Facebook iOS SDK*[68]

Al instalar el paquete descargado, se crea una nueva carpeta dentro de "Documentos" denominada *FacebookSDK* cuyos contenidos son los siguientes:

[68] Facebook. (2012). *Developers Facebook*. Recuperado el 1 de Marzo de 2012, de Developers Facebook: https://developers.facebook.com/ios/ (Facebook, 2012)

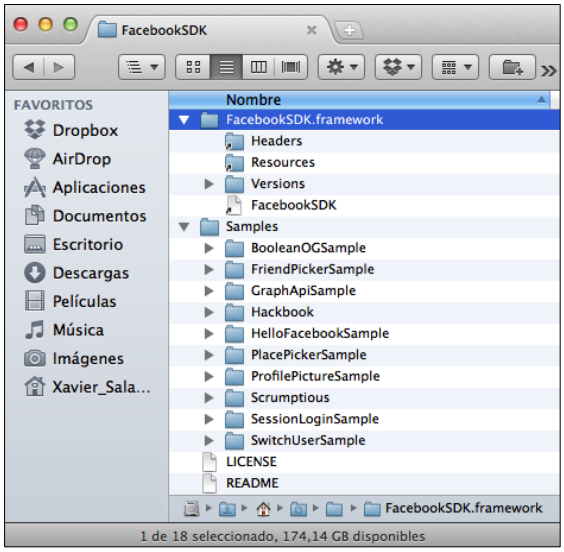


Fig. 2.279: Contenidos de la carpeta *FacebookSDK*

La carpeta *Samples* contiene ejemplos de aplicaciones *iOS* o proyectos de *XCode* que implementan el *framework* de *Facebook* y pueden ser revisados para comprender la implementación del mismo.

El siguiente paso consiste en crear un nuevo proyecto con la plantilla *Single View Application* llamado *FacebookTest* y agregar los *frameworks* necesarios, los cuales son *Accounts*, *AdSupport*, *Social* y *FacebookSDK*. Si se requiere ayuda se puede referir a los apartados *"2.2 Cómo crear un nuevo proyecto"* y *"2.8.1 Cómo añadir frameworks"*.

Los tres primeros *frameworks* anteriores, se encuentran en el listado de librerías y *frameworks* incluidos por defecto en el *IDE XCode*, mientras que el *framework FacebookSDK.framework*, debe ser arrastrado desde la carpeta *FacebookSDK* dentro de Documentos hacia el proyecto, al hacerlo, no se debe olvidar marcar la opción de copiar los archivos dentro del proyecto. Al finalizar este paso, el proyecto consta de los siguientes ficheros y frameworks:

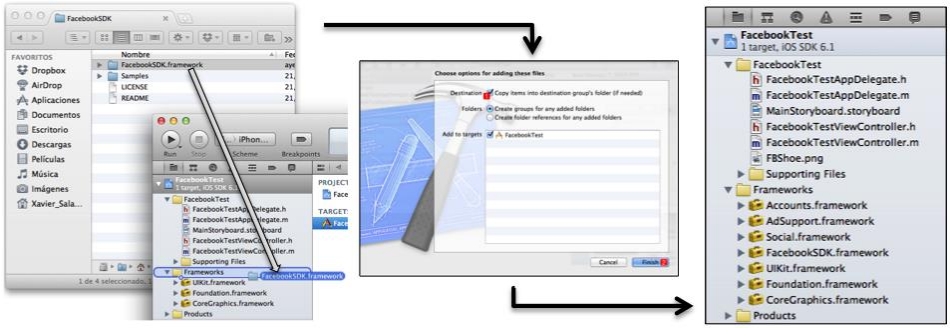


Fig. 2.280: *Frameworks* y archivos del proyecto *FacebookTest*

Como se puede observar, también se ha agregado un archivo imagen a la aplicación (*FBShoe.png*) cuyas dimensiones son irrelevantes; para publicarla en *Facebook* a través de la aplicación.

Algunas partes del *SDK* de *Facebook* se basan en *SQLite* para emitir datos, por lo que se debe crear un enlace tanto a *SQLite* como a *Objective C.* Para esto, se debe seleccionar el proyecto, luego *Build Settings* y a continuación escribir *"other link"* para filtrar el listado inferior. Presionar dos veces sobre el apartado *Other Link Files* para agregar un nuevo enlace. Presionar el botón + y escribir *"–lsqlite3.0"*. Repetir el ultimo paso para agregar la sentencia *"–ObjC"*. Como se indica en la siguiente figura:

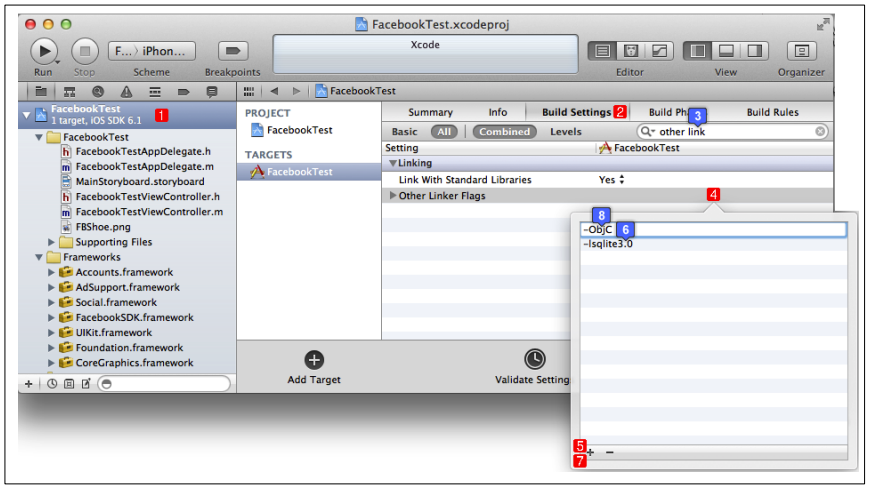


Fig. 2.281: Crear enlaces entre la aplicación y *SQLite / Objective C*

A continuación se agregará un nuevo atributo al archivo de información y lista de propiedades *Info.plist* con el identificador de aplicación de *Facebook*, en caso de disponer una. Este paso es requerido incluso si no se dispone de una aplicación de *Facebook* existente. Lo que se debe hacer es seleccionar el archivo *FacebookTest-Info.plist* y agregar un atributo (Ver Fig. 2.58) llamado *FacebookAppID*, en donde se escribirá el identificador de aplicación de *Facebook* en caso de disponer de una. Caso contrario se debe agregar el atributo y dejar vacío el campo "*Value*", como se indica a continuación:

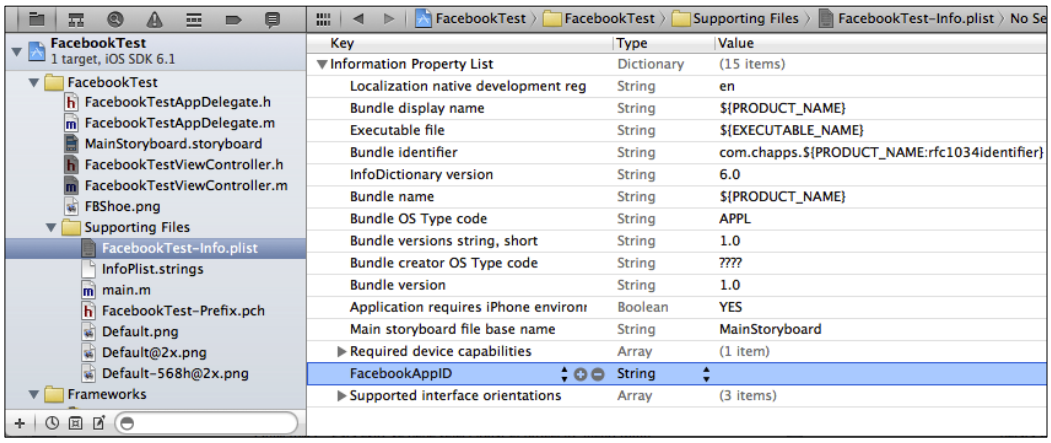


Fig. 2.282: Agregar atributo *FacebookAppID* al fichero *Info.plist*

El siguiente paso consiste en agregar un botón al *View Controller* del *Storyboard*, el cual, al ser presionado presentará una ventana modal con el contenido del *post* que se publicará en *Facebook*. Asignar el método *publicarEnFacebook* al botón. Si se requiere ayuda se puede referir al apartado *"2.5.2 Asignación de métodos a componentes de GUI"*. El *View Controller* debe ser similar al siguiente:

Fig. 2.283: View Controller de FacebookTest

A continuación se escribirá código para finalizar la implementación de la aplicación conectada con *Facebook*. El primer paso consiste en abrir el fichero cabecera del *View Controller* para importar el *framework* al mismo:

Fig. 2.284: Fichero cabecera del *View Controller* de *FacebookTest*

El siguiente paso consiste en implementar el método *publicarEnFacebook* asignado previamente al botón, dentro del archivo de implementación, así:

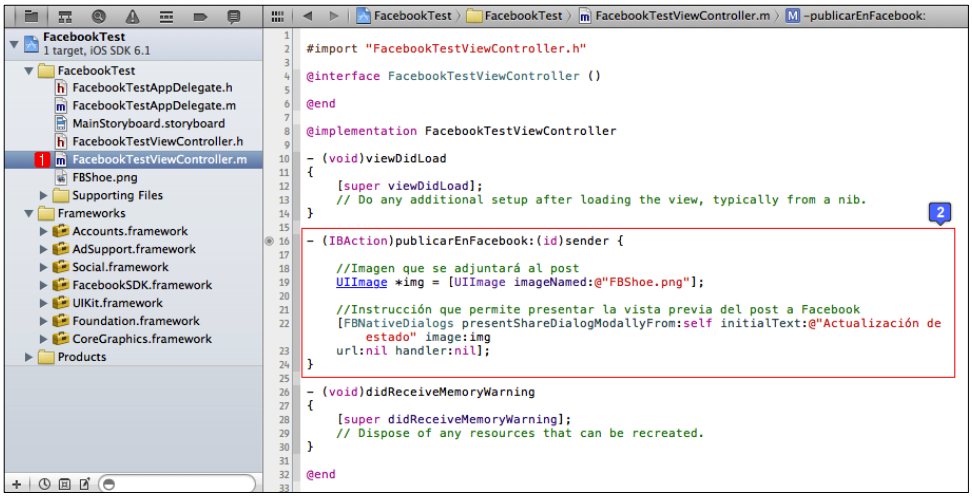


Fig. 2.285: Fichero de implementación del *View Controller* de *FacebookTest*

Esta aplicación funcionaría únicamente en dispositivos *con firmware iOS* 6 o superior. Se pueden revisar los ejemplos incluidos al descargar el *SDK* de *Facebook* para conocer como hacer que esta aplicación sea compatible con dispositivos iOS con *firmwares* anteriores.

Ahora se puede proceder a ejecutar la aplicación en el simulador *iOS*, al presionar el único botón de la vista, se presenta la vista previa del *post* con su texto inicial y la imagen adjunta. En caso de no tener ninguna cuenta de *Facebook* asociada al dispositivo *iOS* del usuario, se presentará la ventana de ajustes para que se asocie una cuenta y al volver a ejecutar la aplicación, se podrá *postear* en *Facebook* sin problemas, y con la facilidad de indicar la ubicación actual y restringir la visibilidad o alcance de la publicación como se indica a continuación:

Fig. 2.286: Aplicación *FacebookTest* en ejecución

2.9 Cómo probar y depurar una aplicación en dispositivos físicos

Para poder probar y depurar las aplicaciones desarrolladas en dispositivos *iOS* físicos, se requiere una suscripción como desarrollador de *Apple*. Para conocer los pasos de suscripción como desarrollador *Apple*, se puede referir al apartado *"2.1.1 Suscripción como desarrollador de Apple"*.

Los miembros desarrolladores de *Apple*, tienen acceso al portal de desarrolladores en línea en donde se pueden encontrar: herramientas y recursos técnicos, herramientas para la distribución de aplicaciones en *App Store* y, un centro de comunidad y soporte.

Dentro del grupo de las herramientas y recursos técnicos, se encuentra el portal de aprovisionamiento *iOS (iOS Provisioning Portal)*, mediante el cual se pueden probar las aplicaciones desarrolladas en los dispositivos *iOS* físicos.

El *iOS Provisioning Portal* permite administrar certificados, autorizar dispositivos *iOS*, y crear perfiles para desarrollo, pruebas y distribución de aplicaciones.

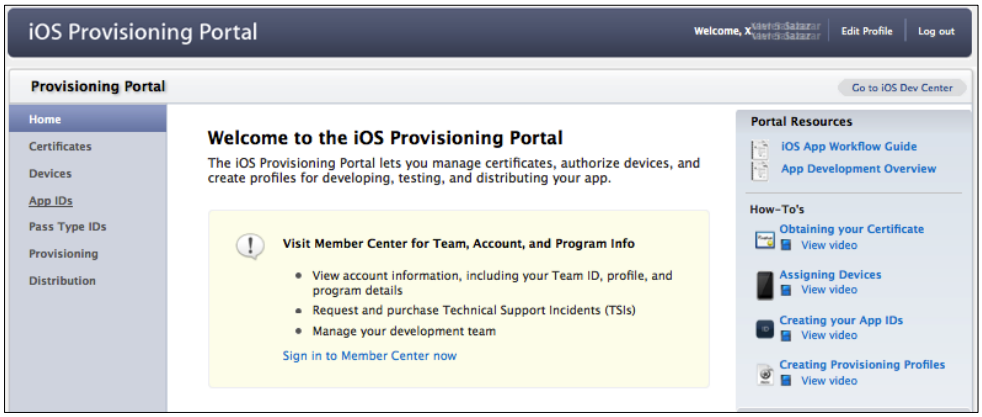


Fig. 2.287*: iOS Provisioning Portal* exclusivo para miembros desarrolladores de *Apple*[69]

Para explicar como probar una aplicación en un dispositivo *iOS* físico, es necesario comprender algunos conceptos, que se presentan como categorías dentro del *iOS Provisioning Portal*:

[69] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

Certificados: Todas las aplicaciones *iOS* deben ser firmadas por un certificado válido para poder ser ejecutadas en un dispositivo físico. Para poder firmar aplicaciones con propósito de pruebas, cada desarrollador individual necesita un certificado de desarrollador.

Los certificados digitales que se solicitan y descargan son documentos electrónicos que asocian la entidad digital con otra información que incluye: nombre, correo electrónico, o empresa. Una entidad digital es un medio electrónico de identificación que consiste de una "llave privada" secreta y una "llave pública" compartida. La llave privada permite a *XCode* firmar el binario de la aplicación *iOS.* Un certificado de desarrollo *iOS* esta restringido únicamente para desarrollo y es válido por un tiempo limitado.

App IDs: Permiten a una aplicación comunicarse con las notificaciones *Push* y/o accesorios hardware externos. Adicionalmente un *App ID* puede ser usado también para compartir documentos y datos de configuración entre las aplicaciones utilizando *iCloud*. Un *App ID* es una combinación de una cadena única de diez caracteres llamada *"Bundle Seed ID"* y un tradicional *"Bundle Identifier"*. Un *App ID* puede ser incorporado en cualquier accesorio externo de hardware que se desee aparear con la aplicación *iOS.* El registro del *App ID* es necesario para utilizar las notificaciones *Push* e incorporar compras dentro de la aplicación (*In-App Purchases*).

El *Bunde Identifier* de un *App ID* puede ser reemplazado por un carácter "*" (esto es conocido como *Wild-Card App ID*) de forma de que un simple *App ID* pueda ser utilizado para construir e instalar múltiples aplicaciones. Si no se usa un *Wild-Card App ID,* el *Bundle Identifier* del mismo debe ser ingresado en *XCode* para permitir que la aplicación se instale en un dispositivo físico. La porción *Bundle Seed ID* del *App ID* no necesita ser ingresada en *XCode.* Una *Wild-Card App ID* no puede ser usada con las notificaciones *Push* o para *In-App Purchases.*

Dispositivos: La sección de dispositivos en el i*OS Provisioning Portal* permite ingresar dispositivos *Apple* que se utilizarán para el desarrollo. Para poder depurar la aplicación *iOS* en un dispositivo físico, se debe primero ingresar el identificador único del dispositivo dentro de este portal. Se pueden agregar hasta cien dispositivos dentro del portal.

Provisioning Profile (Perfil de Aprovisionamiento): Es una colección de recursos digitales que son los únicos que pueden atar a desarrolladores y dispositivos a un equipo de desarrollo de *iOS* autorizado y permiten que un dispositivo pueda ser utilizado para pruebas. Un perfil de aprovisionamiento de desarrollo (*Development Provisioning Profile*) debe ser instalado en cada dispositivo en el que se desee que se ejecute la aplicación. Cada perfil de aprovisionamiento de desarrollo contendrá un conjunto de certificados de desarrollo, identificadores únicos de dispositivos y un *App ID*.[70]

[70] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

2.9.1 Pasos para probar una aplicación en un dispositivo iOS físico

En el presente apartado se probará en un dispositivo *iOS* físico la aplicación desarrollada en el apartado *"2.4.4 Rotación de interfaces y sus componentes"*, por lo que se debe abrir el proyecto *RotationExample* y conectar un dispositivo físico mediante cable *usb* al computador *Mac*. En este caso se conectará un *iPhone 5*.

Al hacerlo, se presentará una ventana u organizador. En caso de no visualizar el organizador, se lo puede abrir como se indica a continuación:

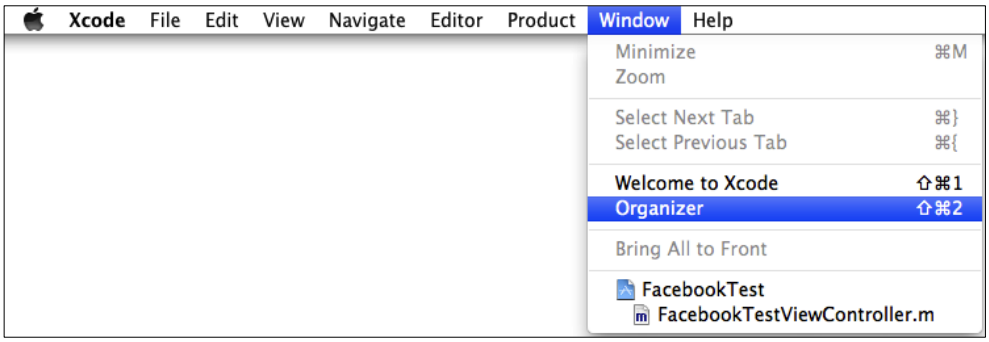


Fig. 2.288: Abrir el organizador de *XCode (Organizer)*

Al abrir el organizador, se debe seleccionar el dispositivo *iOS* conectado y presionar el botón "*Use for development*" así:

Fig. 2.289: Usar *iPhone* 5 para desarrollo

A continuación se instalarán algunos archivos necesarios en el dispositivo *iOS* conectado, así:

Fig. 2.290: Procesando archivos necesarios para *iPhone5*

Acto seguido se agregará automáticamente el identificador único del dispositivo al *iOS Provisioning Profile*, a la categoría *Devices*, es por esto que se presentará una ventana solicitando el ingreso de usuario y contraseña del desarrollador, ingresarlos y continuar así:

Fig. 2.291: Iniciar sesión con los datos de desarrollador

Fig. 2.292: No solicitar un certificado automáticamente

Si no se dispone de un certificado de desarrollador vigente, *XCode* preguntará si se desea solicitar uno. Presionar cancelar (ver Fig. 2.292) para solicitar uno de forma manual posteriormente.

Al finalizar la instalación de los archivos necesarios en el dispositivo *iOS* físico, se presentará la información del mismo en el organizador. Como se puede ver, no tiene ningún perfil de aprovisionamiento asignado.

Fig. 2.293: Dispositivo *iOS* conectado y configurado pero sin perfil de aprovisionamiento

El siguiente paso consiste en solicitar un certificado de desarrollador y verificar el agregado exitoso del identificador único del dispositivo *iOS* al *iOS Provisioning Profile*. Para esto se debe dirigir al portal de desarrolladores de *Apple* (www.developer.apple.com) e iniciar sesión en el mismo. Ingresar después al centro de miembros como se indica a continuación:

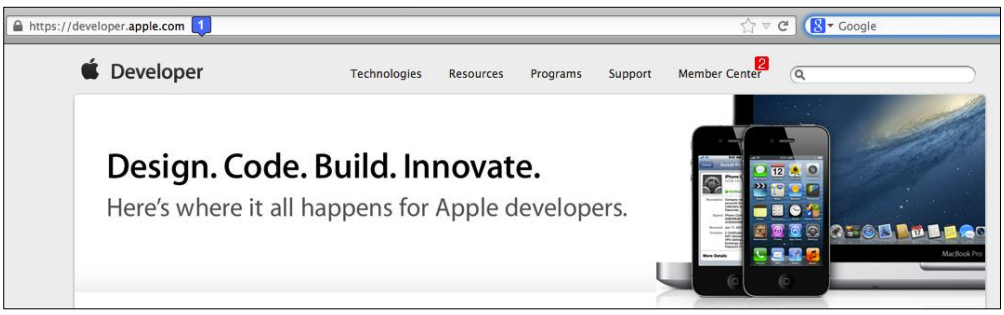


Fig. 2.294: Ingreso al centro de miembros desarrolladores de *Apple*[71]

[71] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

Dentro del área de miembros, se debe presionar el enlace que lleva al *iOS Provisioning Portal* como se indica a continuación:

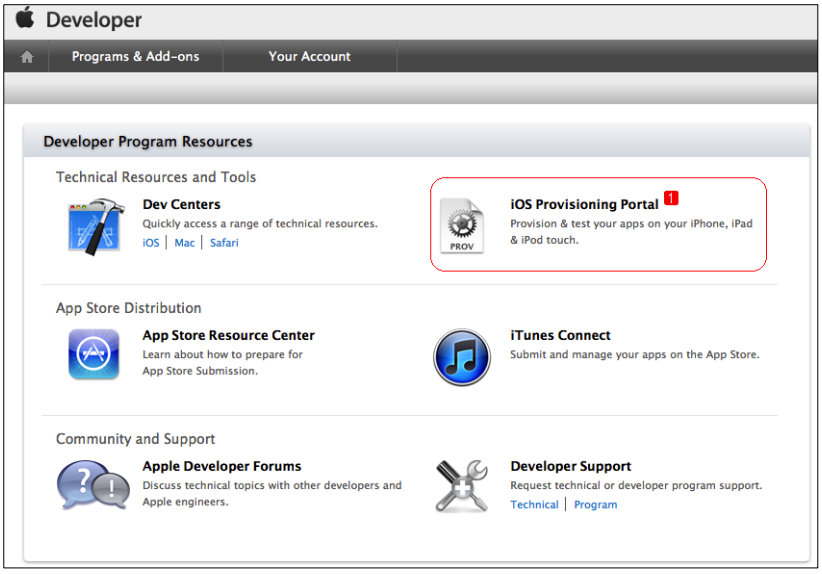


Fig. 2.295: Acceso al iOS Provisioning Portal[72]

El *iOS Provisioning Portal* es el presentado en la Fig. 2.287, donde se debe dirigir a la sección *Devices* para verificar que el dispositivo *iOS* ha sido agregado. Se debería encontrar un dispositivo con su nombre (*iPhone* 5) y su identificador único, el cual en este caso, debería coincidir con el de la Fig. 2.293. Así:

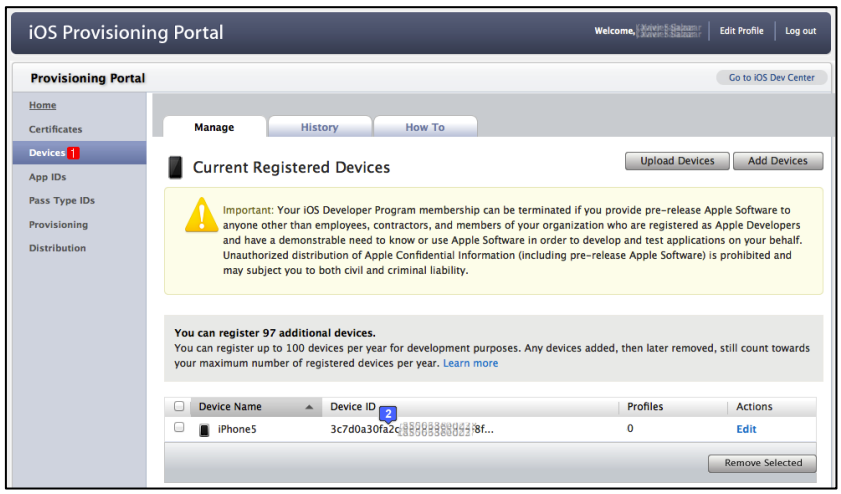


Fig. 2.296: Dispositivo agregado exitosamente al *iOS Provisioning Profile*

[72] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

Ahora se debe dirigir a la sección *Certificates* para solicitar un certificado de desarrollador. Como se puede observar no se dispone de ninguno actualmente, por lo que se debe presionar el botón *Request Certificate*.

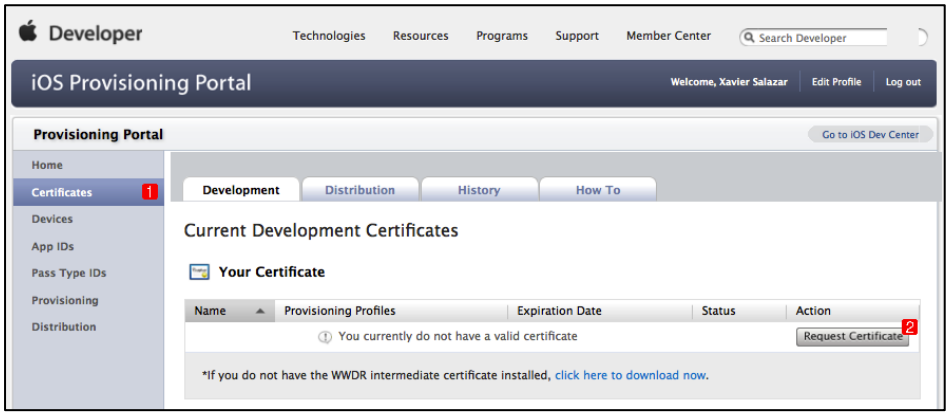


Fig. 2.297: Solicitar un certificado de desarrollador[73]

Al presionar este botón se explican los pasos que se deben seguir para solicitar un certificado de desarrollador, los cuales son los siguientes:

Abrir la aplicación "Acceso a llaveros" o *"Keychain Access"* del *Mac* y abrir el asistente de solicitud de certificados de una autoridad de certificación así:

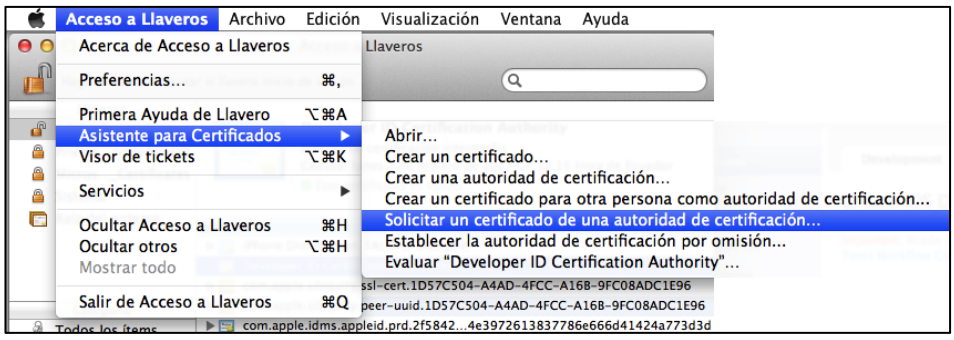


Fig. 2.298: Solicitar un certificado de una autoridad de certificación mediante la aplicación Acceso a Llaveros incluida por defecto en el *Mac*

Al continuación se presentará una ventana en la cual se deben ingresar los datos del desarrollador: Correo electrónico y nombre. Marcar la opción la palabra clave "se

[73] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

guarda en el disco" y presionar continuar. Dejar el campo Dirección de correo de la CA vacío.

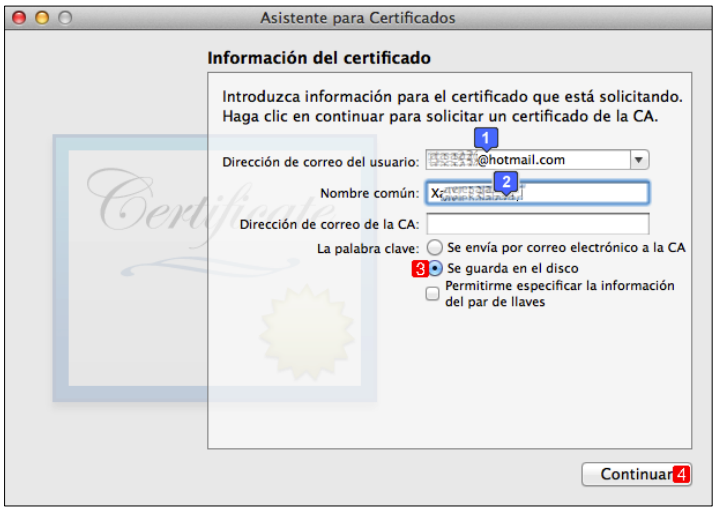


Fig. 2.299: Datos del desarrollador para el certificado

Finalizar indicando el directorio en donde se grabará la solicitud de certificado. Se mostrará una ventana de conclusión del procedimiento y se podrá localizar el fichero creado con el asistente dentro del directorio indicado.

Fig. 2.300: Directorio de grabado

Fig. 2.301: Archivo de solicitud de certificado

Fig. 2.302: Conclusión del procedimiento

Se debe enviar este archivo de solicitud de certificado recién creado a través del *iOS Provisioning Portal:*

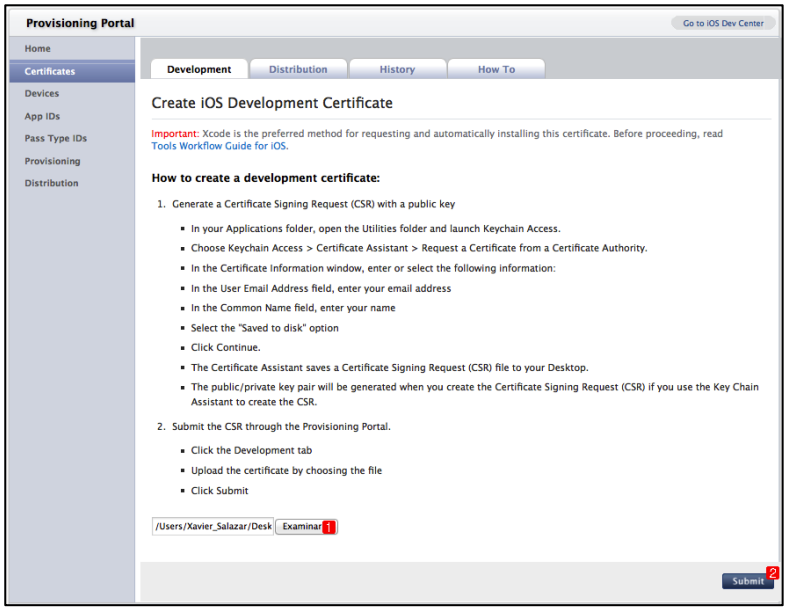


Fig. 2.303: Enviar la solicitud de certificado de desarrollador

Al enviar la solicitud, se encontrará enseguida el certificado en la sección *Certificates* del *iOS Provisioning Portal.*

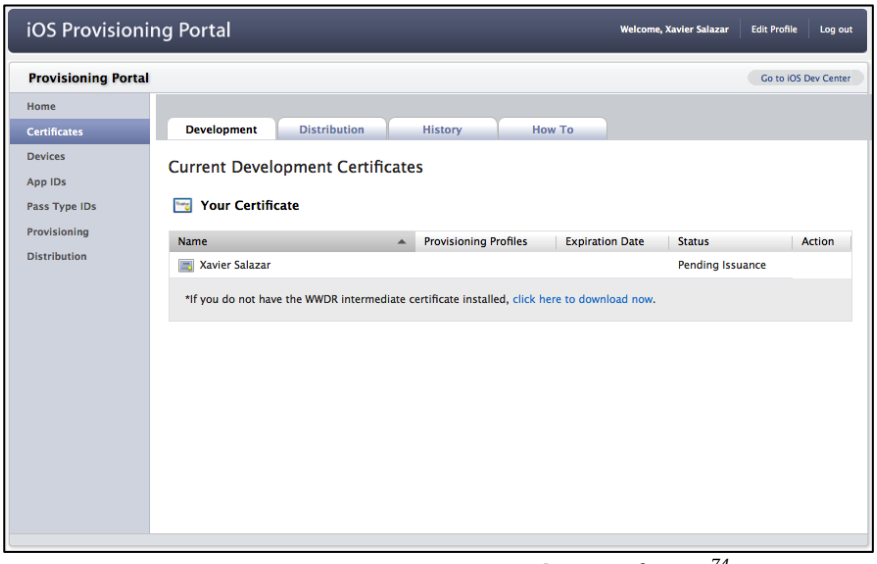


Fig. 2.304: *iOS Provisioning Portal - Certificates*[74]

[74] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

A continuación se creará un *App ID* para la aplicación *RotationExample*, para esto se debe dirigir a la sección *App IDs* y presionar el botón *New App ID.*

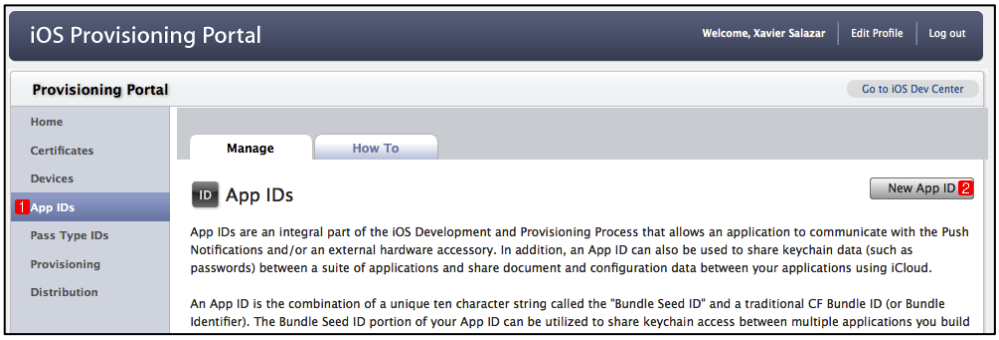


Fig. 2.305: Crear un nuevo *App ID* para la aplicación *RotationExample*

Para crear un nuevo *App ID* se debe ingresar un nombre que lo identifique en el *iOS Provisioning Profile* y escribir el *Bundle Identifier* de la aplicación que se desea probar en el dispositivo físico, para mayor información sobre el *Bundle Identifier* se puede referir al apartado *"2.2 Cómo crear un nuevo proyecto"*. El *Bunde Seed ID*, es un identificador único del desarrollador para cualquier *App ID* que se agregue. Finalizar presionando el botón *Submit.*

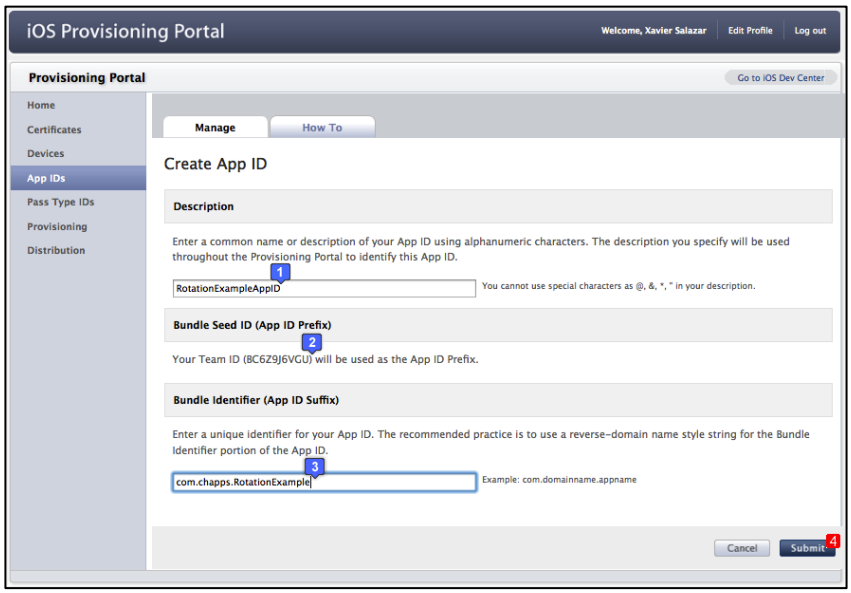


Fig. 2.306: Escribir los datos necesarios para crear el *App ID*[75]

[75] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

Enseguida se puede visualizar el *App ID* recién creado, el mismo que puede ser configurado para utilizar notificaciones *push*, *iCloud* e *In-App Purchases.*

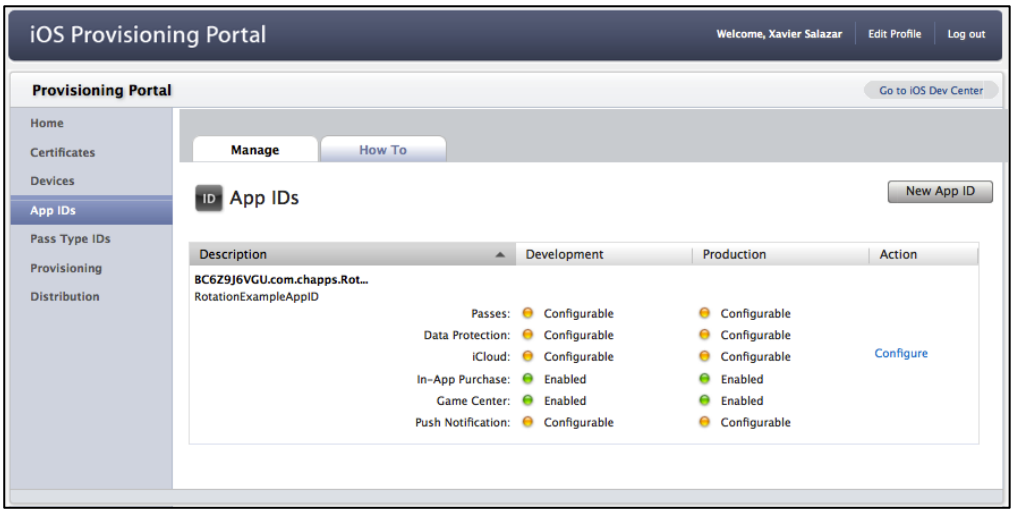


Fig. 2.307: Listado de *App IDs* con el estado de cada uno de sus atributos[76]

El paso final consiste en crear el *iOS Development Provisioning Profile*, al cual se le asignarán el *App ID* recién creado, el dispositivo físico agregado al portal y el certificado solicitado previamente. Para esto se debe dirigir a la sección *Provisioning* y presionar *New Profile.*

Fig. 2.308: Crear un nuevo Development Provisioning Profile

El nombre del perfil sirve para identificarlo, se debe seleccionar el certificado que se desea asociar junto al *App ID* y los dispositivos que se desea que puedan ejecutar la aplicación. Finalizar presionando el botón *Submit*.

[76] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

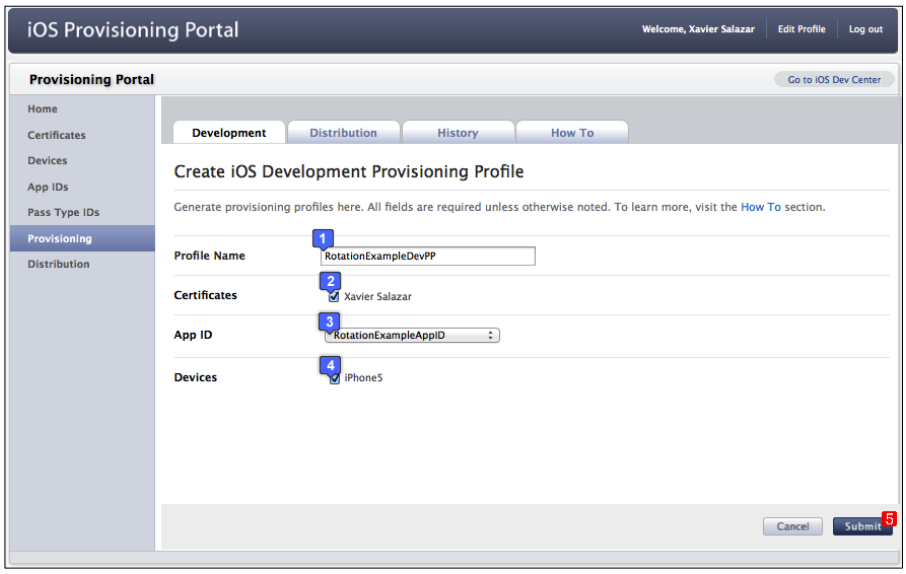


Fig. 2.309: Datos de un nuevo *Development Provisioning Profile*

De esta manera se crea el *Development Provisioning Profile*, y puede ser encontrado en la sección *Provisioning*. Se lo debe descargar presionando el botón *Download*.

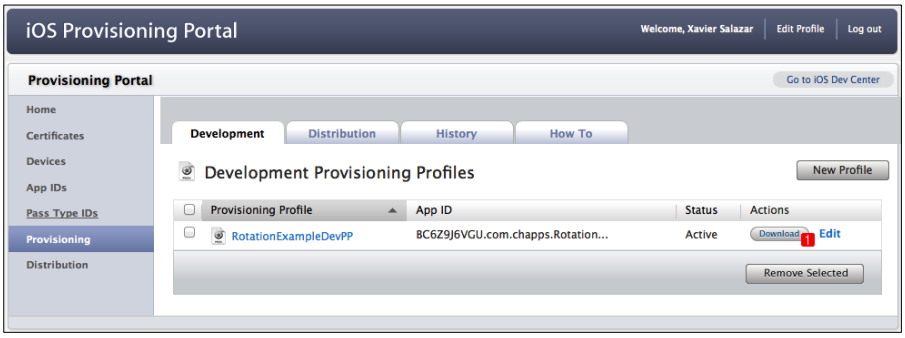


Fig. 2.310: Descargar el Development Provisioning Profile para RotationExample[77]

De igual manera, se debe descargar el certificado de desarrollador solicitado anteriormente de la sección *Certificates.* Como se puede observar, se le ha asignado el *Provisioning Profile* creado en el paso anterior. Se debe descargar también el certificado de *Apple WorldWide Developer Relations Certification Autorithy WWDR*, como se indica a continuación:

[77] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

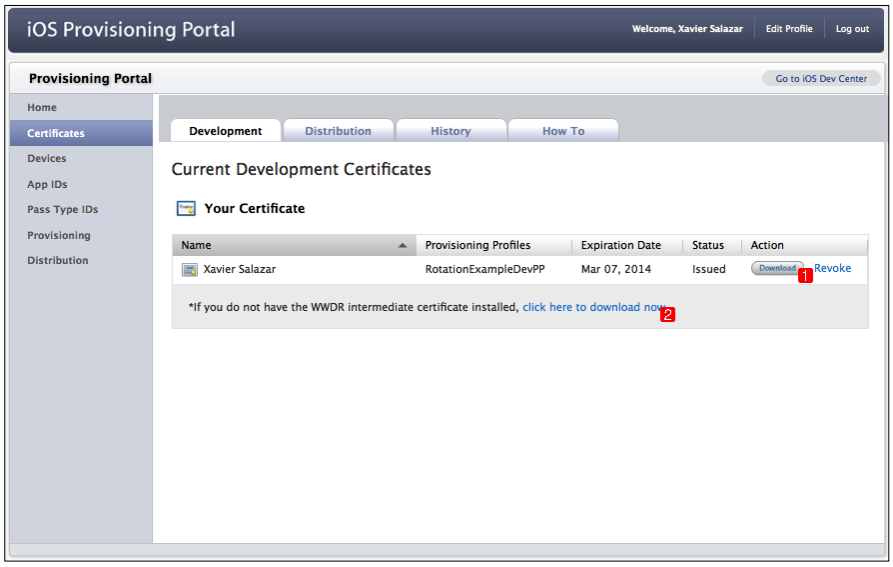


Fig. 2.311: Descarga de certificados de desarrollador y *WWDR*[78]

Los tres archivos descargados deben ser ejecutados. Los dos certificados se abren en la aplicación Acceso a Llaveros, mientras que el *Provisioning* Profile se puede observar dentro del *Organizer* de *XCode.*

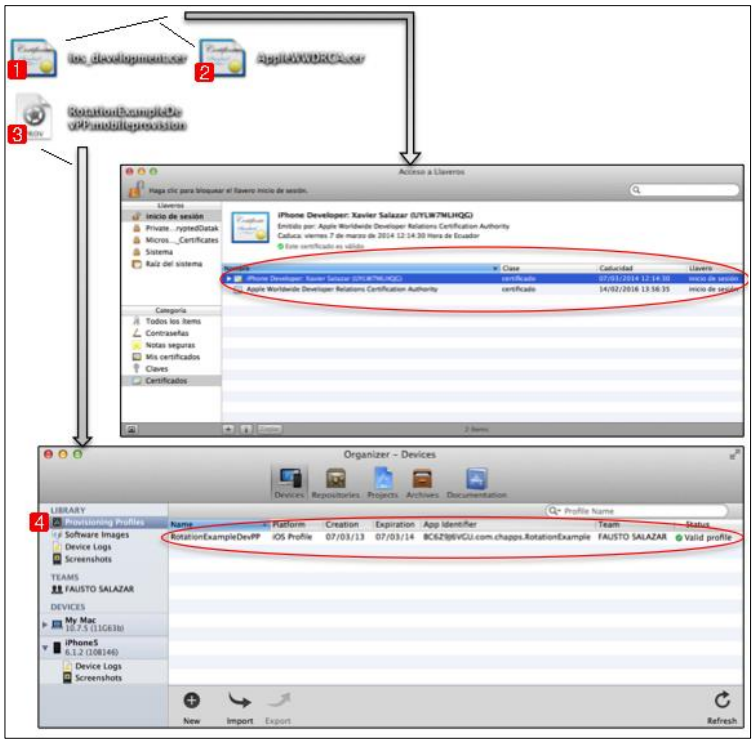


Fig. 2.312: Ejecución de los ficheros descargados

[78] Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

Los certificados permiten firmar el código fuente de las aplicaciones desarrolladas para ser ejecutarlas en un dispositivo físico. Por esto, el último paso para poder ejecutar la aplicación en un dispositivo físico, consiste en abrir el proyecto y configurar dentro de sus ajustes el firmado de código. Así:

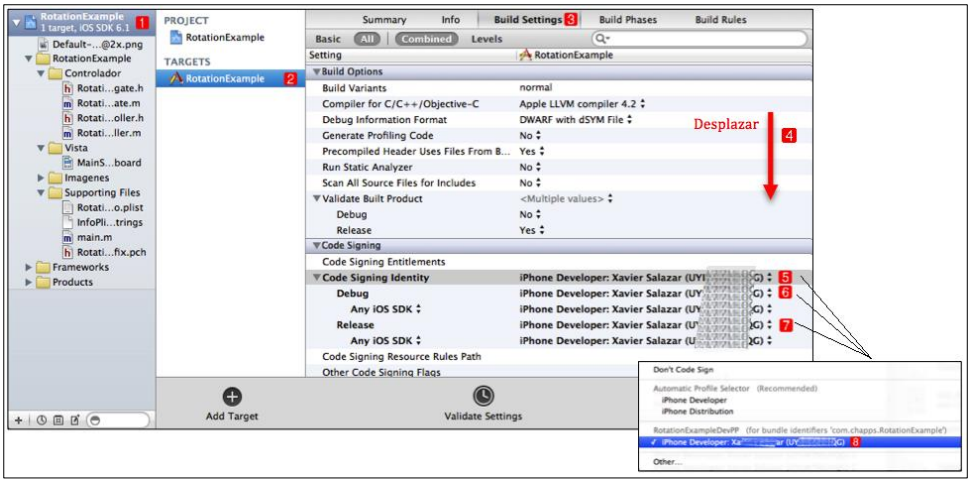


Fig. 2.313: Configurar el firmado de código de la aplicación

A partir de este momento se puede ejecutar la aplicación en el dispositivo físico configurado anteriormente. Antes de hacerlo se debe indicar que se desea ejecutar la aplicación en el dispositivo físico, como se indica a continuación:

Fig. 2.314: Ejecución de *RotationExample* en un dispositivo físico

2.9.2 Compilación y depuración de aplicaciones

La depuración de aplicaciones se puede realizar tanto en el simulador *iOS* como en dispositivos físicos. Es importante depurar aplicaciones en dispositivos físicos, debido a que el simulador *iOS* carece de algunas funciones importantes como cámara y acelerómetro.

Para poder depurar una aplicación, es necesario que ésta pueda ser compilada con éxito para su posterior ejecución. El proceso de compilación de una aplicación se realiza automáticamente previo a la ejecución de la misma. En caso de no poder compilarse una aplicación, el *IDE* presenta los errores de compilación. Al presionar cualquiera de estos errores, se puede ubicar directamente la línea de código que está causando el conflicto.

A continuación se presenta un ejemplo: al abrir el proyecto *AdvancedVariablesExample*, se ha borrado un carácter "**;**" al final de una sentencia. *XCode* al momento de querer ejecutar la aplicación no lo permite e indica la localización y descripción del error. Así:

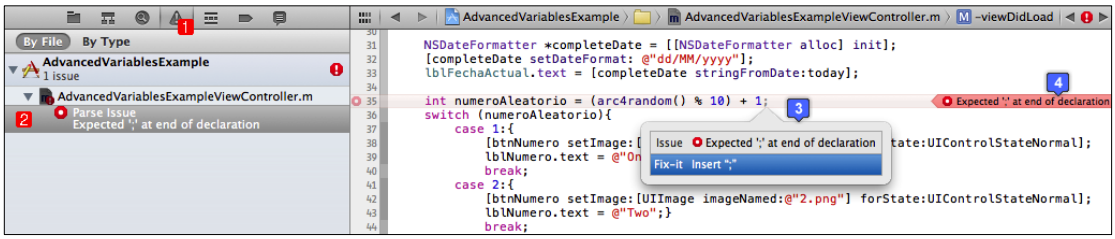


Fig. 2.315: Error ubicado en la compilación del proyecto *AdvancedVariablesExample*

El usuario puede entones corregir este error y proceder con la ejecución de la aplicación. Alguna pocas veces el *IDE XCode*, al igual que la mayoría de *IDEs*, tiene conflictos al identificar la posición exacta del error en el código y puede señalar otras líneas.

Una vez que se pueda compilar exitosamente la aplicación, se la puede depurar. Para depurarla, se puede hacer uso de *breakpoints.* Los *breakpoints* son puntos en donde se

detiene la ejecución de la aplicación y se puede visualizar los contenidos actuales de las variables que intervienen en la misma.

Se insertará un *breakpoint* en el proyecto *AdvacedVariablesExample*. Para insertar el *breakpoint* en una línea concreta de código fuente, se debe presionar el número de la misma, así:

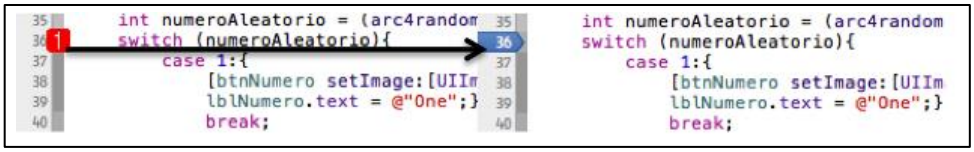


Fig. 2.316: Insertar un *breakpoint* en proyecto *AdvancedVariablesExample*

Ahora, la aplicación se ejecutará hasta que se detenga en este *breakpoint*. En esta línea de código, se ha almacenado en la variable *numeroAleatorio* un número entre uno y diez; y, de acuerdo a este se presenta el texto en inglés correspondiente al mismo. Al depurar la aplicación se puede conocer el contenido de esta variable (En el área de depuración situado en la parte inferior) y por lo tanto el número en letras que se presentará en la aplicación, como se indica a continuación:

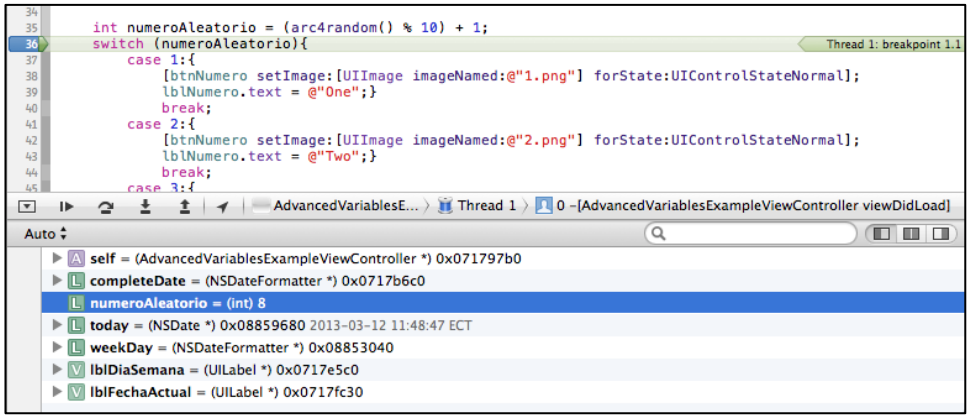


Fig. 2.317: Depuración de una aplicación

El área de depuración se compone de algunos botones los cuales tienen distintas funcionalidades:

Fig. 2.318: Barra de Depuración

El botón  permite mostrar o esconder el Área de Depuración.

El botón  permite continuar con la ejecución de una aplicación después de que ésta haya sido detenida por un *breakpoint.*

El botón  permite continuar la ejecución hacia la siguiente instrucción. Si esta instrucción es un método o función, lo omite y continúa con la siguiente instrucción del mismo fichero.

El botón  es igual al anterior pero en lugar de saltar las funciones o métodos, entra en los mismos y continúa la ejecución línea a línea.

El botón  es igual al anterior pero en lugar de avanzar con la ejecución, permite regresar una instrucción hacia atrás.

El botón  permite simular una ubicación *GPS* en el simulador *iOS*. Al presionarlo, presenta un listado de ubicaciones que pueden ser simuladas.

Mientras se depura una aplicación se puede presionar cualquiera de los botones anteriores de forma de proseguir con la depuración: avanzar, retroceder, regresar a la ejecución o simular cualquiera de las ubicaciones disponibles. A medida que se avance a través de cada línea de código el área de depuración presentará las variables y sus contenidos actuales.

2.10 Uso de archivos multimedia

A diferencia de desarrollar aplicaciones para computadores *Mac* y *PC*, al desarrollar aplicaciones para dispositivos *iOS*, se dispone de características únicas que se pueden utilizar; entre éstas: el micrófono, cámara, *GPS* integrado, acelerómetro, entre otras.

Estas características de los dispositivos *iOS*, permiten desarrollar aplicaciones útiles y entretenidas para los usuarios. Un ejemplo claro de esto son los juegos que utilizan el acelerómetro como "*Asphalt 7"* en el cual se debe girar el dispositivo para controlar el movimiento de un vehículo; o, aplicaciones como "*Shazam*" que utiliza el micrófono del dispositivo *iOS* para grabar un segmento de una canción e identificarla a través de una comparación de ésta con una base de datos en línea.

Otros tipos de aplicaciones como "*Foursquare"* utilizan el *GPS* integrado del dispositivo *iOS* para mostrar al usuario su ubicación en un mapa y los lugares cercanos a ésta. Esta red social permite también acceder a la cámara y capturar fotografías para publicarlas.

Las aplicaciones que se pueden desarrollar gracias a estas características de los dispositivos *iOS* son innumerables, existen muchas con distintos fines: aplicaciones que simulan un velocímetro de un vehículo y alertan los límites de velocidad; aplicaciones deportivas que permiten crear rutas y seguir el progreso de los usuarios sobre la misma; aplicaciones que permiten aplicar efectos especiales sobre fotografías capturadas mediante la cámara, entre otras.

Es por esto que se ha considerado destinar sub apartados para conocer como desarrollar aplicaciones que puedan acceder a la cámara y micrófono del dispositivo *iOS*, los mismos que serán presentados a continuación.

2.10.1 Implementación de fotos de galería y cámara

Otra de las ventajas de desarrollar aplicaciones para dispositivos *iOS* es que se puede acceder a la mayoría de funcionalidades del dispositivo mediante métodos y funciones implementados dentro de distintos *frameworks*, incluidos por defecto en el *IDE*.

Desarrollar una aplicación que permita gestionar imágenes almacenadas en el dispositivo o capturadas mediante la cámara integrada es relativamente sencillo gracias a un componente que permite esta funcionalidad y es incluido por defecto en el *IDE*. El único problema es que el simulador *iOS* no permite ejecutar instrucciones de aplicaciones que intentan acceder a la cámara; al intentar ejecutar una de estas instrucciones, *XCode* no lo permitirá y se presentará un error en la consola.

Es necesario que este tipo de aplicaciones sean probadas directamente en dispositivos *iOS* físicos, en los cuales se ejecutarán sin problemas, permitiendo capturar fotografías con la cámara integrada. Si se requiere ayuda se puede referir al apartado *"2.9 Cómo probar y depurar una aplicación en dispositivos físicos"*

En el siguiente sub apartado se desarrollará una aplicación capaz de acceder a la galería y cámara del dispositivo *iOS* para obtener una imagen a través de cualquiera de estos medios y colocarla dentro de una vista.

2.10.1.1 Uso del componente UIImagePickerController

El componente *UIImagePickerController* incluido en el *framework UIKit* permite acceder a las fotos almacenadas en el dispositivo *iOS* y a la cámara para capturar una nueva fotografía. Además tiene una característica que permite editar una foto antes de utilizarla con cualquier fin.[79]

A continuación se explicará como insertar este componente a una aplicación. Para empezar se creará un nuevo proyecto denominado *PhotosApp* utilizando la plantilla *Single View Application.* Si se requiere ayuda se puede referir al apartado *"2.2 Cómo crear un nuevo proyecto"*.

Construir una interface gráfica de usuario como la siguiente:

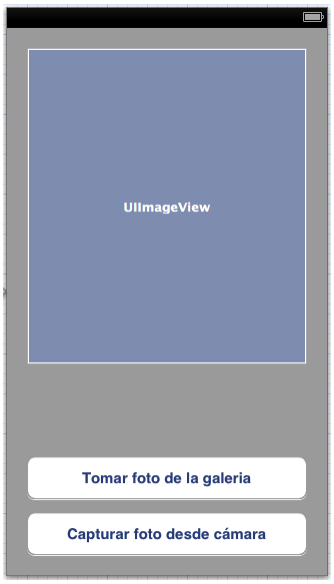


Fig. 2.319: View Controller de PhotosApp

Acto seguido asignar la variable *imgFoto* al *UIImageView* y los eventos *tomarFotoGaleria* y *capturarFotoCamara* a los botones correspondientes. Si se requiere ayuda se puede referir al apartado *"2.5.1 Asignación de variables a componentes de GUI"* y *"2.5.2 Asignación de métodos a componentes de GUI"*. En el fichero cabecera del *View Controller* se debe crear también una variable para manejar el

[79] Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress. p673, p674.

ImagePickerController y declarar adherencia a protocolos de delegado de *UIImagePickerController* y *UINavigationController*. Al declarar adherencia a protocolos de delegado de estos controles, se pueden manejar eventos exclusivos de los mismos dentro del archivo de implementación. El fichero cabecera lucirá de la siguiente manera:

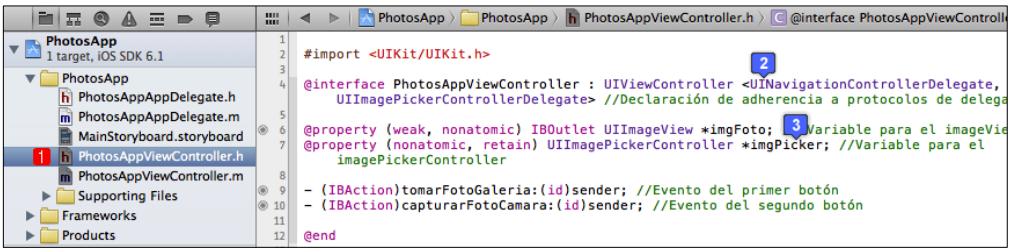


Fig. 2.320: Archivo cabecera del *View Controller* de *PhotosApp*

Dentro del fichero de implementación se debe realizar primero el *@synthesize* de las propiedades establecidas en el fichero cabecera. Acto seguido, inicializar el *imagePickerController* dentro del evento *viewDidLoad*. Después, implementar los métodos correspondientes a cada botón e implementar un evento de delegado del *imagePickerController* para realizar una acción después de haber obtenido una imagen desde la cámara o galería del dispositivo *iOS*. El archivo de implementación quedará como el presentado en la Fig. 2.321.

Una vez desarrollada la aplicación se recomienda probarla en un dispositivo físico para verificar que ésta permita capturar fotos mediante la cámara del dispositivo *iOS*. La funcionalidad de presentar la galería para seleccionar una foto, es posible verificarla en el simulador *iOS*, como se indica en la Fig. 2.322.

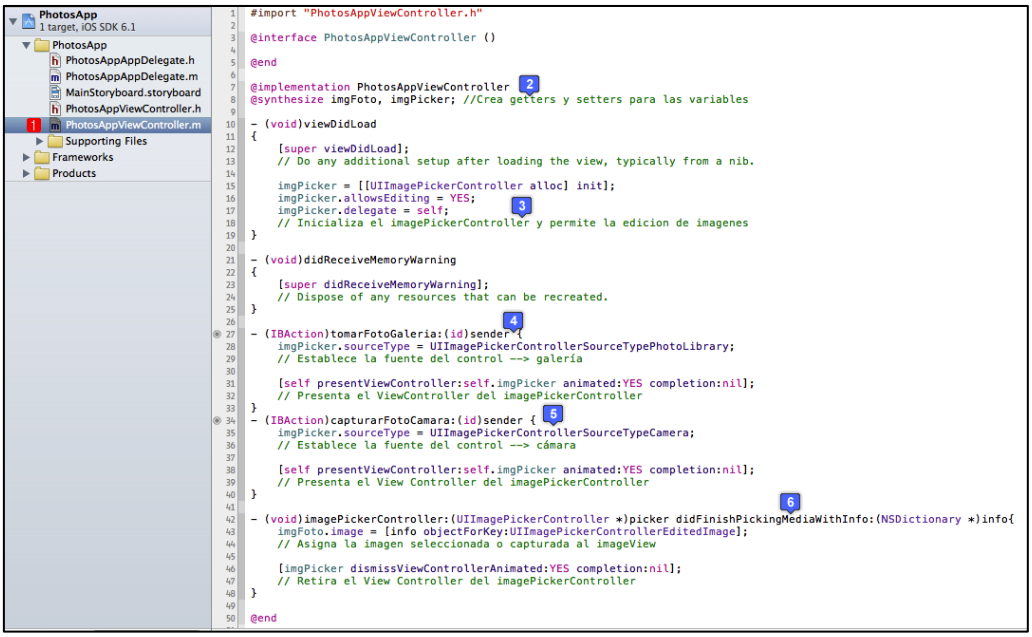


Fig. 2.321: Archivo de implementación del *View Controller* de *PhotosApp*

Fig. 2.322: Aplicación *PhotosApp* en ejecución en el simulador *iOS*

2.10.2 Acceso al micrófono del dispositivo

Otra de las características de los dispositivos *iOS* es el micrófono. Muchas aplicaciones lo usan con distintos fines, entre las mas conocidas se encuentran "*Shazam*" que graba un segmento de una canción y la identifica y "*Voxer*" que es una aplicación de mensajería instantánea de notas de voz.

En el presente apartado se creará una aplicación que permita utilizar el micrófono para grabar notas de voz y reproducirlas. Esta aplicación podrá ser probada en el simulador *iOS* sin problemas debido a que se utiliza el micrófono integrado del computador *Mac* para simular el micrófono del dispositivo *iOS.*

Para desarrollar una aplicación que permita utilizar el micrófono integrado del dispositivo *iOS* se requiere del framework *AVFoundation,* el cual permite utilizar dos controles fundamentales necesarios: un grabador de audio *AVAudioRecorder* y un reproductor de audio *AVAudioPlayer.*[80]

A continuación se explicará como insertar estos componentes a una aplicación. Para empezar se creará un nuevo proyecto denominado *MicrophoneApp* utilizando la plantilla *Single View Application*. Si se requiere ayuda se puede referir al apartado *"2.2 Cómo crear un nuevo proyecto"*.

Agregar el *framework AVFoundation* al proyecto. Si se requiere ayuda se puede referir al apartado *"2.8.1 Cómo añadir frameworks"*.

Construir una interface gráfica de usuario como la siguiente:

[80] iPhone Tutorials. (2012). *Recording Audio on an iPhone with AVAudioRecorder (iOS 6)*. Recuperado el 15 de Marzo de 2013, de Recording Audio on an iPhone with AVAudioRecorder (iOS 6): http://prassan-warrior.blogspot.com/2012/11/recording-audio-on-iphone-with.html

Fig. 2.323: *View Controller* de MicrophoneApp

Acto seguido asignar los eventos *grabarAudio* y *reproducirAudio* a los botones correspondientes. Si se requiere ayuda se puede referir al apartado *"2.5.2 Asignación de métodos a componentes de GUI"*. En el fichero cabecera del *View Controller* se deben crear también variables para manejar el *AVAudioRecorder* y *AVAudioPlayer*. Declarar también adherencia a los protocolos de delegado de los controles anteriores. Se debe también importar el fichero cabecera del *framework AVFoundation* al *View Controller*. El fichero cabecera lucirá de la siguiente manera:

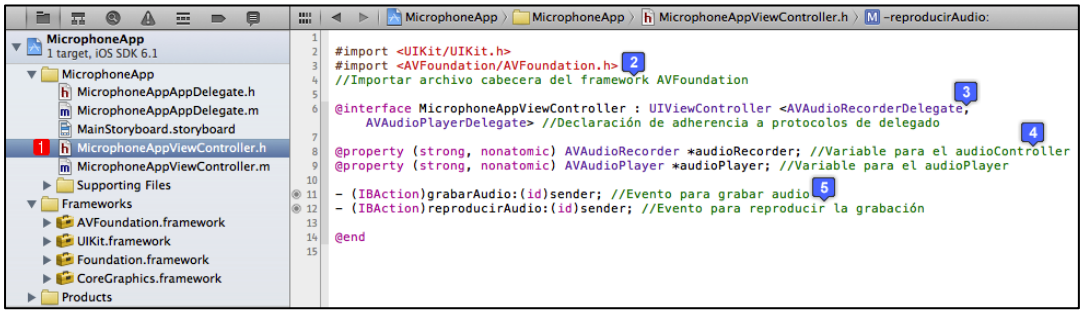


Fig. 2.324: Archivo cabecera del *View Controller* de *MicrophoneApp*

Dentro del fichero de implementación se debe realizar primero el *@synthesize* de las propiedades establecidas en el fichero cabecera. Acto seguido, inicializar el *audioRecorder* dentro del evento *viewDidLoad;* para inicializarlo es necesario indicar el archivo de grabación, la ruta del archivo y los ajustes de grabación como calidad, numero de canales, entre otros. Después, implementar los métodos correspondientes a cada botón, para esto es necesario implementar un método *detener,* el cual detendrá una grabación o reproducción en curso. Para el método reproducir, al inicializar el *audioPlayer* se debe indicar que debe reproducir los contenidos del *audioRecorder.*

Se implementarán también algunos eventos de delegado del *audioPlayer* y *audioRecorder,* los cuales no tienen ninguna instrucción en su interior, sino solo se presentan para indicar que se puede hacer uso de los mismos; por ejemplo, para indicar al usuario que hubo algún error en la codificación de la grabación en caso de que esto suceda.

El archivo de implementación quedará como el presentado en la Fig. 2.325.

Se puede ejecutar la aplicación tanto en el simulador *iOS* como en cualquier dispositivo físico y verificar que se pueden realizar grabaciones de audio y reproducirlas al presionar los botones correspondientes.

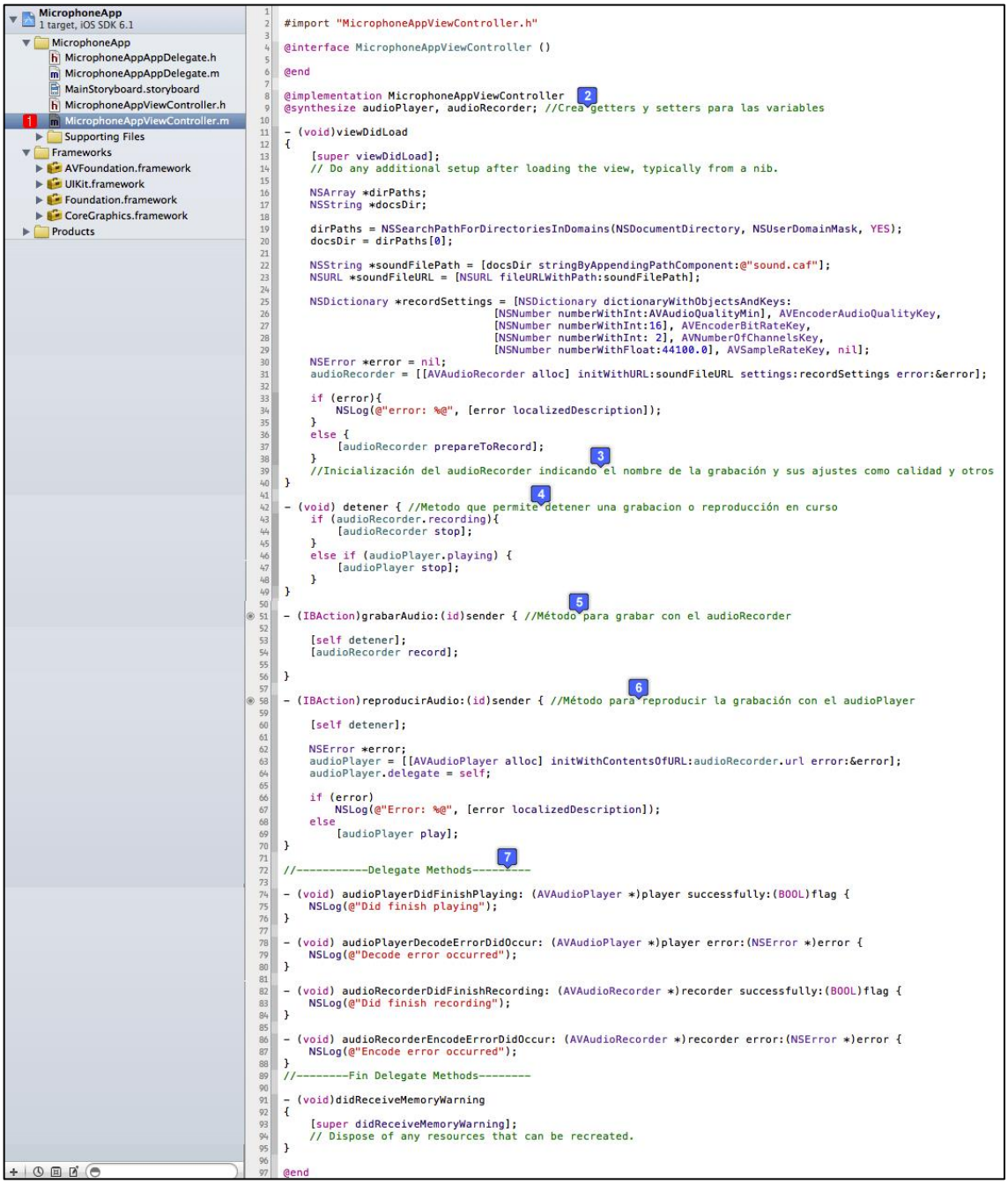


Fig. 2.325: Archivo de implementación del *View Controller* de *MicrophoneApp*

2.11 Acoplamiento de aplicaciones para que funcionen en iPads

En la actualidad muchas aplicaciones tienen su versión para dispositivos *iPhone* e *iPod* y su versión exclusiva para dispositivos *iPad.* Al buscar en *AppStore* estas aplicaciones, generalmente se presenta su versión estándar y su versión *High Definition (HD).*

Fig. 2.326: Aplicación *SD* para *iPhone e iPod* y *HD* para *iPad*

Otras aplicaciones, como *"Whatsapp",* han sido desarrolladas exclusivamente para *iPhone* e *iPod* y no tienen una versión para *iPad.* Éstas pueden ser ejecutadas en *iPads* pero se presentan dentro de una ventana muy pequeña en relación a la pantalla del mismo. Este tipo de aplicaciones presentan un botón *"2x",* que permite ampliar esta ventana para presentar la aplicación en pantalla completa en el *iPad*, pero lamentablemente los contenidos de la aplicación se distorsionan y no se presentan con una buena calidad.

Fig. 2.327: Aplicación exclusiva para *iPhone* e *iPod*

Otro tipo de aplicaciones, conocidas como "aplicaciones universales *iOS"*, generan un único producto final capaz de ejecutarse tanto en dispositivos *iPad* como en *iPhone* e *iPod.* Implican un mayor esfuerzo al momento de desarrollo y por lo tanto tienen un costo superior en *AppStore*, pero tienen una muy buena acogida por parte de los usuarios que saben que al comprar una sola vez el producto, podrán ejecutarlo en cualquiera de sus dispositivos *iOS.*

Fig. 2.328: Aplicación Universal para cualquier dispositivo *iOS*

Como se ha indicado en el apartado *"2.2 Cómo crear un nuevo proyecto"*, al generar un nuevo proyecto, se puede especificar si la aplicación será exclusiva para *iPhone/iPod*, exclusiva para *iPad* o universal. Al crear una aplicación universal, se crean dos *Storyboards*, uno para *iPhone/iPod* y uno para *iPad* y se puede proceder a desarrollar la aplicación universal.

El objetivo de este apartado consiste en acoplar una aplicación desarrollada exclusivamente para *iPhone/iPod* para que funcione como aplicación universal y pueda ser ejecutada también en un *iPad.*

Se acoplará la aplicación *AdvancedActionsExample* para que pueda ser ejecutada en dispositivos *iPad,* para esto se debe empezar por agregar un nuevo fichero *Storyboard* llamado *AdvancedActionsExampleiPadStoryboard*, si se requiere de ayuda se puede

referir al apartado *"2.3.1 Cómo agregar un fichero a un proyecto*" específicamente al subapartado *"a) Agregar un Storyboard"*.

Dentro de este *Storyboard* se agregará un elemento *View Controller* en donde se construirá una interface gráfica de usuario similar a la que se presenta cuando se ejecuta la aplicación en un dispositivo *iPhone/iPod*. Aprovechando el gran tamaño de pantalla del *iPad*, se pueden agrandar los botones para que sean de una dimensión superior y se los puede ubicar en la parte superior de la vista. Se puede hacer lo mismo con la etiqueta en la que se presentan los números, ubicándola en la parte inferior de la vista. Se establecerá un fondo de pantalla color lima, como se indica a continuación:

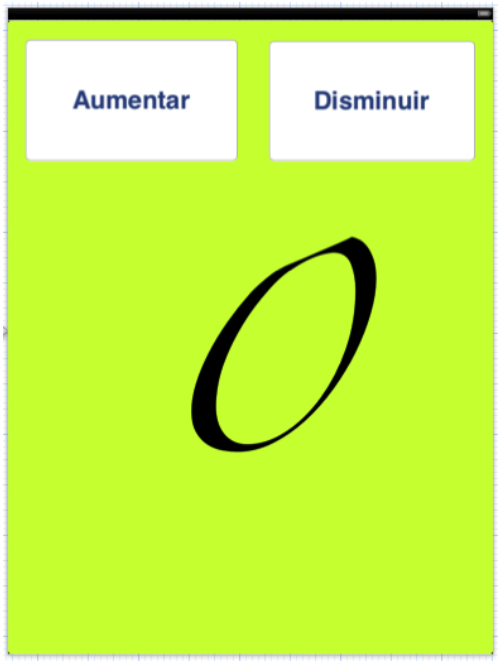


Fig. 2.329: AdvancedActionsExample iPad Storyboard

La clase controladora del *View Controller* dentro del *Storyboard* de *iPad* será la misma que se utilizó con el *View Controller* dentro del *Storyboard* de *iPhone/iPod*. Esto se asigna dentro de las propiedades del *View Controller* como se puede observar en la Fig. 2.114. La etiqueta en donde se muestran los números, requiere de una variable, la cual será la misma utilizada en el caso de *iPhone/iPod*, denominada *"lblNumero"*. Como se

hará referencia a una variable existente, lo que se hará es seleccionar la etiqueta y seleccionar el inspector de conexiones  de la barra de selección de inspector de la Fig. 2.113. A continuación se debe realizar una conexión desde el círculo del apartado *"New Referencing Outlet"* hasta la etiqueta. Al realizar la conexión se presenta la lista de variables existentes en la clase controladora, en este caso *lblNumero*, seleccionarlo para finalizar la asignación de la variable como se indica a continuación:

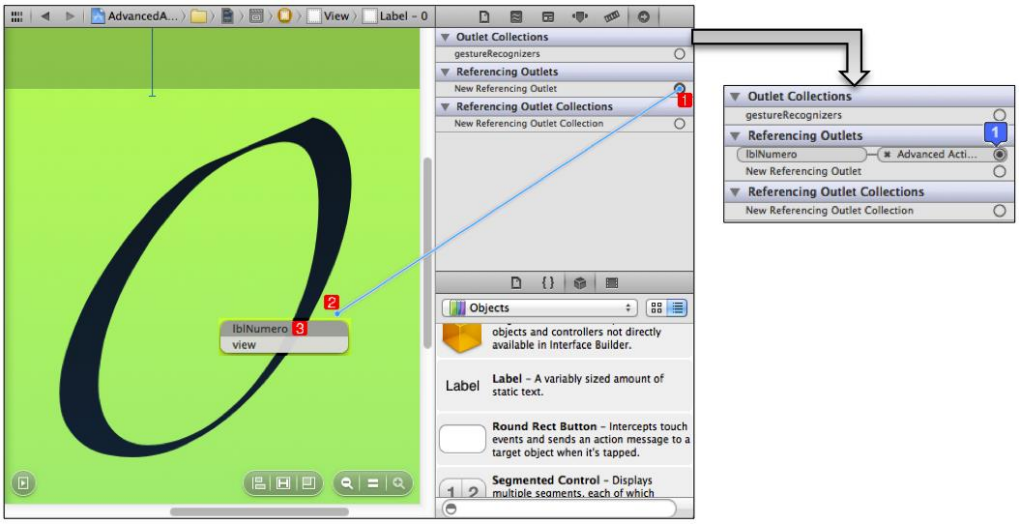


Fig. 2.330: Asignación de una variable existente a un componente en un *Storyboard*

De esta misma forma se deben asignar los métodos *aumentarNumero* y *reducirNumero* a los botones correspondientes. Al seleccionar un botón y después el inspector de conexiones, además de presentar el apartado para asignar una variable, se muestra un apartado con los eventos del botón. Al presionar cualquier botón en una aplicación, se activa el evento *"Touch Up Inside"* por lo que se debe realizar una conexión entre este evento y el botón. Al realizar la conexión se presenta la lista de métodos o acciones disponibles en la clase controladora, en este caso *aumentarNumero* y *reducirNumero*. Seleccionar el evento correspondiente de acuerdo al botón seleccionado, como se indica a continuación:

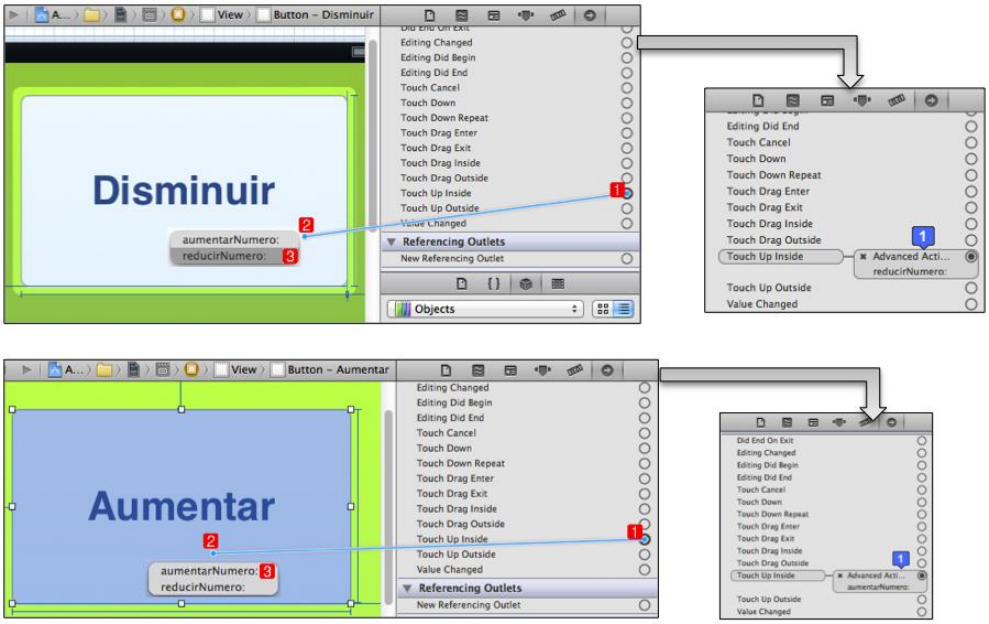


Fig. 2.331: Asignación de métodos existentes a componentes en un *Storyboard*

El último paso consiste en indicar en los ajustes del proyecto que la aplicación será universal y asignar los *Storyboards* correspondientes para cada apartado; se debe seleccionar primero el *Storyboard* de *iPhone/iPod* en el apartado *"iPhone/iPod Deployment info",* luego, desplazarse casi hasta el final en la misma ventana y seleccionar el *Storyboard* de *iPad* en el apartado *"iPad Deployment info"*, así:

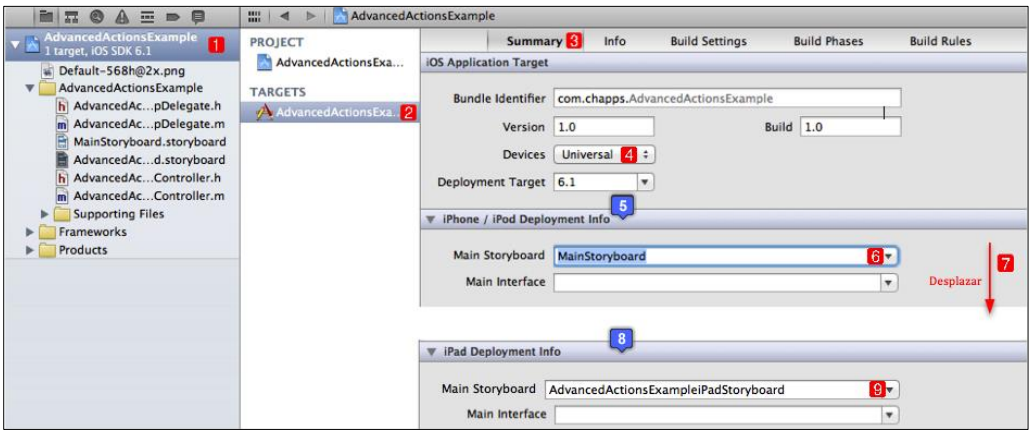


Fig. 2.332: Configuración de los ajustes del proyecto de la aplicación universal

Importante: Si se necesita saber mediante código fuente si la aplicación se encuentra ejecutando en un dispositivo *iPad* o *iPhone*/*iPod*, se puede usar una instrucción. Por

ejemplo, si se desea que cada vez que se presione el botón "Aumentar" en un *iPad* se presente una alerta, se debería escribir lo siguiente:

```
- (IBAction)aumentarNumero:(id)sender {

    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad){ //Aplicación se ejecuta en un iPad
        UIAlertView *alerta = [[UIAlertView alloc] initWithTitle:@"iPad" message:@"Boton iPad"
            delegate:nil cancelButtonTitle:@"Ok" otherButtonTitles:nil, nil];
        [alerta show]; //Muestra una ventana alerta
    }
    int numero = [[lblNumero text] intValue] + 1;
    lblNumero.text = [NSString stringWithFormat: @"%i", numero];
}
```

Fig. 2.333: Método *aumentarNumero*. Presenta una alerta si se ejecuta en un *iPad*

A continuación se puede ejecutar la aplicación en cualquier dispositivo físico o en el simulador *iOS*, el cual permite simular un *iPhone* o *iPad*. Al presionar el botón "Aumentar" en un *iPad*, se debería presentar el mensaje o alerta.

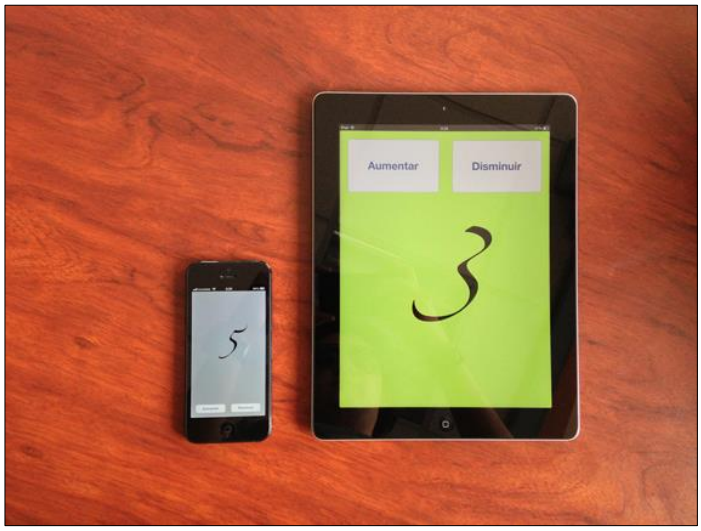


Fig. 2.334: Aplicación universal *AdvanceActionsExample* en ejecución en diferentes dispositivos *iOS*

Se puede encontrar este proyecto dentro del comprimido *AdvancedActionsExampleUniversal.zip* de la carpeta de aplicaciones.

2.12 Seguridad en aplicaciones iOS

Crear una aplicación implica un gran esfuerzo por parte del desarrollador o equipo de trabajo. La mayoría de desarrolladores buscan una remuneración por su trabajo ya sea por ventas de la aplicación desarrollada o por mostrar publicidad dentro de la misma. Es una verdadera lástima ver que una gran cantidad de aplicaciones de la tienda *AppStore* han logrado ser *hackeadas* y se pueden conseguir gratuitamente. Esto tiene algunas implicaciones negativas en los desarrolladores como las siguientes:

Fig. 2.335: Implicaciones negativas en los desarrolladores[81]

El *hacking* de aplicaciones se realiza generalmente así: los atacantes o *hackers* intentan encontrar vulnerabilidades de seguridad, al encontrarlas, las utilizan para robar información confidencial, corromper sistemas y redes, entre otros.

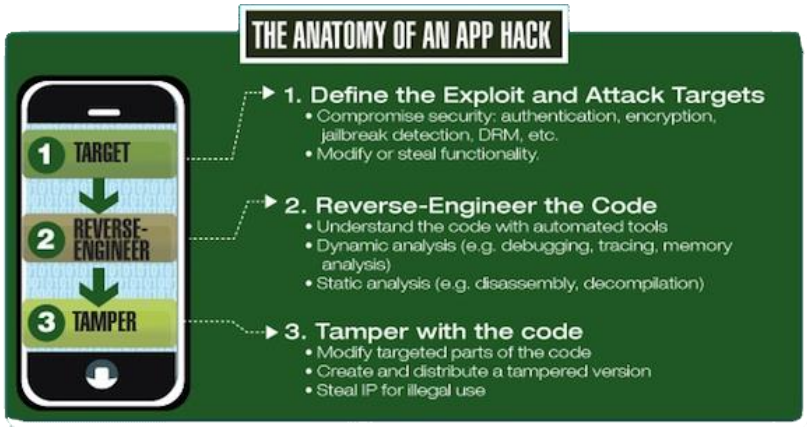


Fig. 2.336: Forma común de *hacking* de aplicaciones[82]

[81] IDownloadBlog. (2012). *State of app security on iOS and Android*. Recuperado el 19 de Marzo de 2013, de State of app security on iOS and Android: http://www.idownloadblog.com/2012/08/20/infographic-pirates-ahoy/

[82] IDownloadBlog. (2012). *State of app security on iOS and Android*. Recuperado el 19 de Marzo de 2013, de State of app security on iOS and Android.

Se debe robustecer la seguridad de las aplicaciones debido a que se encuentra en juego la reputación del desarrollador como la información de sus clientes o usuarios. Existen diferentes tipos de *hacking* de aplicaciones; los mas comunes son los siguientes:

Fig. 2.337: Diferentes tipos de *hacks* en aplicaciones móviles[83]

La seguridad en una aplicación debe ser considerada desde un principio, ya que si no es segura, se requeriría de un rediseño extensivo para robustecerla.

Al desarrollar una aplicación utilizando *Objective C* y sus *frameworks,* ésta, será considerablemente más segura comparada con programas implementados en *C* puro.

Es importante conocer las amenazas existentes en cuanto a codificación de aplicaciones e incorporar practicas de codificación segura mediante el planeamiento y desarrollo del producto.[84] La mayoría de estas amenazas o vulnerabilidades de *software* se asocian con las siguientes categorías:

[83] IDownloadBlog. (2012). *State of app security on iOS and Android*. Recuperado el 19 de Marzo de 2013, de State of app security on iOS and Android: http://www.idownloadblog.com/2012/08/20/infographic-pirates-ahoy/

[84] Apple. (2012). *Introduction to Secure Coding Guide*. Recuperado el 18 de Marzo de 2013, de Introduction to Secure Coding Guide: https://developer.apple.com/library/mac/#documentation/security/Conceptual/SecureCodingGuide/Introduction.html

a) Desbordamiento de búfer: Ocurre cuando una aplicación intenta escribir información "después del final" o "antes del inicio" de un búfer; esto generalmente sucede cuando una entrada de información es mas grande que el espacio reservado en memoria para esta. Si no se controla esta entrada, los datos sobrescriben a otros datos en memoria. Esto puede colgar la aplicación, comprometer información, o proveer un vector de ataque que comprometa el sistema en el que se ejecuta (*iOS).* Esta es la mayor fuente de vulnerabilidades de seguridad en *C, Objective C* y *C++*.

Solución: Detectar desbordamientos de búfer y corregirlos. Para esto se debe ingresar una cantidad superior de datos de los solicitados en cada una de las entradas de información de la aplicación; si existieran desbordamientos de búfer, ésta se colgaría (un poco después, al intentar utilizar los datos sobrescritos). El registro en la consola podría indicar o proveer pistas de que la causa del cuelgue fue por este motivo. Si se encuentran desbordamientos de búfer, se deben corregir, caso contrario podrían ser explotados.

b) Entrada sin validar: Se deben revisar todos los datos de entrada recibidas por la aplicación para asegurar que los datos son razonables. Los atacantes o *hackers* inyectan diferentes tipos de archivos a todas las posibles entradas de la aplicación esperando un cuelgue o comportamiento extraño de la misma; luego buscan una forma de aprovecharse del problema (*exploit*). Estos *exploits* han sido utilizados para tomar control de sistemas operativos, robar información y otros fines; incluso, un *exploit* es usado para realizar el *"jailbreak"* de dispositivos *iOS.*[85]

[85] Apple. (2012). *Types of Security Vulnerabilities*. Recuperado el 18 de Marzo de 2013, de Types of Security Vulnerabilities: https://developer.apple.com/library/mac/#documentation/security/Conceptual/SecureCodingGuide/Articles/TypesSecVuln.html#//apple_ref/doc/uid/TP40002529-SW2

c) Condiciones de competencia: Existen cuando cambios en el orden de dos o mas eventos pueden alterar el comportamiento de la aplicación. Si se requiere un orden concreto de ejecución para el funcionamiento apropiado de una aplicación, esto es un *bug.* Un atacante puede aprovecharse de la situación para insertar código malicioso, cambiar el nombre de un archivo o interferir con la operación normal de la aplicación, esto se convierte en una vulnerabilidad de seguridad.

Solución: Validar todas las formas posibles de ejecución de los eventos para que puedan ser realizados sin tener que seguir un orden específico.

d) Operaciones de archivos inseguros: La aplicación puede escribir o leer un archivo al que generalmente no tienen acceso los usuarios; pero un *hacker* puede cambiar los permisos del directorio y/o archivo y manipularlo.

Solución: Revisar el código resultante después de rutinas de lectura o escritura para detectar si ha habido alteraciones, en caso de que haya sucedido, se debe estar preparado para que la aplicación responda de manera adecuada.

e) Almacenamiento seguro: Consiste en proteger los datos e información de los usuarios cuando la información ha sido almacenada.

Solución: La solución de este problema es un diseño cuidadoso de arquitectura con una aproximación basada en riesgo para ayudar a decidir la postura de seguridad que la aplicación tendrá con respecto al almacenamiento de información. Una vez determinado el riesgo, es esencial proteger la información sensible en el dispositivo *iOS* usando una combinación de un fuerte encriptado y los servicios de llavero de *Apple* o *"Keychain"*[86]

[86] Apple. (2012). *Types of Security Vulnerabilities*. Recuperado el 18 de Marzo de 2013, de Types of Security Vulnerabilities: https://developer.apple.com/library/mac/#documentation/security/Conceptual/SecureCodingGuide/Articles/TypesSecVuln.html#//apple_ref/doc/uid/TP40002529-SW2

f) Comunicación segura con servidores: Casi todas las aplicaciones que manejan información sensible de usuario se conectan con algún servidor. Los desarrolladores son los que enfrentan el desafío de proteger información sensible en el transito mientras atraviesa el internet y algunas veces medios inalámbricos inseguros.

Solución: Encriptado de información que viajará por internet; el servicio web contratado debe brindar seguridad internamente, se puede encontrar mayor información refiriéndose al apartado *"2.16.2 Seguridad en el servicio web Amazon SimpleDB"*[87]

Según el portal web *ReadWrite.com,* existen cuatro pasos principales que los desarrolladores deben realizar para asegurar la protección de información en una aplicación *iOS*:

I. Realizar una evaluación de seguridad de la arquitectura de la aplicación: La evaluación debe ser realizada por el equipo de seguridad antes de que el equipo de desarrollo empiece a implementar el diseño arquitectónico. Las preguntas pueden incluir: ¿Es necesario almacenar las credenciales de acceso en el dispositivo *iOS*?, ¿Qué componente de autenticación se debe implementar?, ¿Qué información procesará la aplicación y de que manera? entre otras.

II. Declarar la clase de protección correcta para la información: Seleccionar *"ProtectionComplete"* si *iOS* debe cifrar siempre el archivo y descifrarlo cuando el usuario ingrese su contraseña. Seleccionar *"ProtectionNone"* si la protección del archivo no está atado a la contraseña y debe estar disponible tan pronto se encienda el dispositivo *iOS.*

[87] Apple. (2012). *Types of Security Vulnerabilities*. Recuperado el 18 de Marzo de 2013, de Types of Security Vulnerabilities: https://developer.apple.com/library/mac/#documentation/security/Conceptual/SecureCodingGuide/Articles/TypesSecVuln.html#//apple_ref/doc/uid/TP40002529-SW2

III. Colocar las credenciales en el llavero o *Keychain:* El llavero es un objeto de cifrado especial que tiene tres clases de protección disponibles: *"AvailableWhenUnlocked"*, *"AvailableAfterFirstUnlock"* y *"AvailableAlways"*. Si se necesitan las credenciales en tareas que se ejecutan en *background*, se debe utilizar *"AvailableAlways"*. *"AvailableWhenUnlocked"* es como *"ProtectionComplete"* en el paso número dos pero para los objetos de sistema. *"AvailableAfterFirstUnlock"* mantiene la información cifrada después de iniciar el dispositivo *iOS* hasta que el usuario ingrese su contraseña.

IV. Purgar los datos adecuadamente: La información debe ser removida adecuadamente de la memoria local. Además, se debe manejar de forma adecuada las excepciones que se podrían presentar cuando la información no se encuentre disponible debido a que el dispositivo esta bloqueado.[88]

El reporte del portal web lista algunas de las mejores practicas a las que se deben referir los programadores de aplicaciones *iOS*. Las mas importantes son: garantizar la entrada de usuarios basada en contraseñas; y, no acceder a la información privada como número telefónico o *IMEI* sin previa autorización del usuario.

[88] ReadWrite. (2012). *How To Build Secure iOS Apps*. Recuperado el 18 de Marzo de 2013, de How To Build Secure iOS Apps: http://readwrite.com/2011/06/09/How-to-build-secure-ios-apps#awesm=~o8zlHjhSzjKZZX

2.13 Cómo subir una aplicación a la tienda AppStore

El siguiente paso después del desarrollo de una aplicación *iOS* es la distribución de la misma, ya sea de forma gratuita o de pago, en la tienda de aplicaciones de *Apple: AppStore*.

El proceso de subida de una aplicación a la tienda *AppStore*, es relativamente sencillo, el problema, son las exigencias y restricciones que tiene la empresa *Apple* sobre las aplicaciones que ingresarán a su tienda. Es por esto que se recomienda la lectura del apartado *"2.13.1 Consideraciones para evitar el rechazo de aplicaciones"* antes de enviar una aplicación para revisión y posterior aprobación para su comercialización en la tienda *AppStore*.

La primera vez que se quiera subir una aplicación a la tienda *AppStore*, el desarrollador deberá leer y acordar los contratos para publicar aplicaciones de pago; publicar aplicaciones gratuitas; y, en caso de requerirlo, acordar el contrato de *iAds* para mostrar publicidad dentro de las aplicaciones.

Además, el desarrollador debe indicar su información personal y cuenta bancaria en la cual se realizarán los pagos por parte de *Apple* por ingresos de ventas y publicidad en las aplicaciones que desarrolle.

En algunas estadísticas que se pueden encontrar en internet, las aplicaciones *iOS* son muy seguras y fiables comparadas con el resto de aplicaciones móviles.[89] Se acredita esta seguridad y fiabilidad al riguroso proceso de revisión que realiza la empresa *Apple* para aprobar y distribuir las aplicaciones en su tienda.

[89] IDownloadBlog. (2012). *State of app security on iOS and Android*. Recuperado el 19 de Marzo de 2013, de State of app security on iOS and Android: http://www.idownloadblog.com/2012/08/20/infographic-pirates-ahoy/

Este proceso de revisión de aplicaciones tiene un tiempo promedio de siete días laborables debido al alto número de peticiones de revisión enviadas diariamente por parte de los desarrolladores.

Para poder publicar aplicaciones en la tienda *AppStore*, se requiere del portal de desarrolladores de *Apple* denominado *iTunes Connect* el cual será presentado en el apartado *"2.13.2 Portal iTunes Connect"*.

2.13.1 Consideraciones para evitar el rechazo de aplicaciones

Debido a la gran cantidad de solicitudes de revisión de aplicaciones por parte de los desarrolladores a nivel mundial, el proceso de aprobación de una aplicación para su comercialización en la tienda *AppStore* tarda alrededor de siete días. Cuando una aplicación es rechazada en este proceso revisión de *Apple*, el desarrollador debe realizar las respectivas correcciones en la aplicación y volverla a enviar para su revisión, lo que implica una espera de alrededor de siete días más, lo cual puede resultar muy desesperante. Es por esto que en este sub apartado se presentarán algunas consideraciones que se deben tomar en cuenta en el desarrollo de aplicaciones para evitar el rechazo de las mismas en el proceso de revisión. Si la aplicación no cumple con alguna de las consideraciones presentadas en este apartado, de seguro será rechazada.[90]

Funcionalidad:

- No debe cerrarse inesperadamente, colgarse o presentar bugs.
- Debe realizar las funciones descritas por el desarrollador al momento de subir la aplicación.
- No debe duplicar a otras aplicaciones ya existentes en *AppStore*.
- No debe ser relativamente sencilla
- No debe impulsar al excesivo consumo de alcohol o sustancias ilegales.
- No debe contener material pornográfico.[91]

[90] Apple. (2012). *Mac App Store Review Guidelines*. Recuperado el 10 de Abril de 2013, de Mac App Store Review Guidelines: https://developer.apple.com/appstore/mac/resources/approval/guidelines.html

[91] Apple. (2012). *Mac App Store Review Guidelines*. Recuperado el 10 de Abril de 2013, de Mac App Store Review Guidelines.

Metadatos:

- La descripción de la aplicación debe ser coherente con el contenido y funcionalidad de la misma.
- Los íconos y capturas de pantalla de las aplicaciones deben ser aptas para todo público.
- El desarrollador es responsable de las palabras o *Keywords* que se utilizarán para encontrar su aplicación. En caso de que estas sean inapropiadas, se la rechazará.
- Las direcciones *URL* de soporte ingresadas al momento de agregar la aplicación en el portal *iTunes Connect* deben ser funcionales y presentar la información adecuada.

Localización:

- Las aplicaciones que utilicen el *GPS* del dispositivo *iOS* deben notificar al usuario, el mismo que debe autorizarla para hacerlo.

Notificaciones *Push*:

- Si la aplicación utiliza notificaciones *Push*, lo debe hacer mediante el servicio *APN (Apple Push Notification)*.
- La aplicación debe solicitar autorización del usuario para enviar notificaciones *Push.*
- La notificación *Push* no debe contener información confidencial.

Publicidad:

- Los *iAd* banners se deben presentar adecuadamente, solo cuando se dispone una conexión a internet en el dispositivo *iOS.*[92]

[92] Apple. (2012). *Mac App Store Review Guidelines*. Recuperado el 10 de Abril de 2013, de Mac App Store Review Guidelines: https://developer.apple.com/appstore/mac/resources/approval/guidelines.html

Interface de usuario:

- Las aplicaciones que luzcan similares a las aplicaciones incluidas por defecto en los dispositivos *iOS* serán rechazadas.
- Si la interface de usuario es muy compleja o muy simple será rechazada.

Estropeado de dispositivos:

- La aplicación no debe incentivar a los usuarios a utilizar el dispositivo de manera en que pudieran estropearlo.
- No deben consumir excesivamente la batería de dispositivo o generar demasiado calor.

Violencia:

- No debe presentar imágenes de animales o personas siendo maltratadas, torturadas o asesinadas.
- Los enemigos en juegos no deben ser representados por una cultura, tribu o raza específica.

Privacidad:

- No debe solicitar información como correo electrónico y fecha de nacimiento para poder funcionar.
- No debe transferir información del usuario sin previa autorización del mismo, informando como se utilizará esta información.

Requerimientos legales:

- No debe utilizar nombres o iconos similares a aplicaciones existentes.
- No debe permitir compartir archivos ilegales.
- No debe permitir realizar llamadas o enviar mensajes anónimos.[93]

[93] Apple. (2012). *Mac App Store Review Guidelines*. Recuperado el 10 de Abril de 2013, de Mac App Store Review Guidelines: https://developer.apple.com/appstore/mac/resources/approval/guidelines.html

2.13.2 Portal iTunes Connect

Es el portal de desarrolladores de *Apple*. En *iTunes Connect* los desarrolladores pueden administrar cada una de las aplicaciones que hayan creado y deseen distribuirlas mediante la tienda *AppStore*. Además en este portal se puede: observar las ventas de las aplicaciones, ver los ingresos por publicidad y el total de ganancias del desarrollador. Por otra parte, en *iTunes Connect* también se configura la información para pagos al desarrollador por parte de *Apple*. A continuación se presenta cada sección de *iTunes Connect*:

a) Sales and Trends: Permite al desarrollador ver y descargar los reportes de ventas diarias, semanales o mensuales de sus aplicaciones. Presenta un desglose de ventas que incluye: numero de ventas, tipo de venta, producto vendido y país en donde se realizó la venta sin importar si la aplicación es gratuita o de pago.

b) Contracts, Tax, and Banking: Permite administrar la información correspondiente a la parte legal con *Apple*, la misma que incluye los contratos que se deben acordar, información de impuestos y cuenta bancaria (de manera de que *Apple* pueda pagar al desarrollador).

c) Payments and Financial Reports: Permite visualizar las ganancias después que *Apple* haya generado este reporte (a mediados de cada mes). Aquí se puede observar también los reportes de pagos pasados.

d) Manage Users: Módulo que se debería utilizar únicamente en caso de tener un equipo de trabajo, ya que permite al administrador crear cuentas para cada miembro del equipo y permite también controlar que secciones podría ver cada miembro.[94]

[94] iPhone Development Tutorial HQ. (2012). *What is iTunes Connect?* Recuperado el 28 de Marzo de 2013, de What is iTunes Connect?: http://www.iphonedevelopmenttutorialhq.com/?p=261

e) Manage Your Apps: Permite administrar las aplicaciones. Aquí se puede: crear una nueva aplicación, editar su descripción y metadatos, crear códigos promocionales, crear nuevas versiones de aplicaciones existentes, entre otras.

f) iAd Network: Permite visualizar un reporte en vivo de cuantos ingresos diarios, mensuales o anuales se generan por cada aplicación que implementa publicidad o *iAds* Esta información puede ser filtrada incluso por el país en el que la publicidad produce los ingresos.

g) Catalog Reports: Permite solicitar reportes de catálogos para el contenido en *AppStore* del desarrollador.

h) Developer Forums: Contiene un foro de desarrolladores de *Apple,* en el cual se comparte el conocimiento con desarrolladores de todo el mundo y se presentan soluciones a diversos problemas.

i) Contact Us: Permite encontrar respuestas o enviar una pregunta específica a un representante de *AppStore*.[95]

Cabe recalcar que el portal de *iTunes Connect* https://itunesconnect.apple.com se encuentra disponible únicamente para desarrolladores de *Apple*. Para conocer como se puede ser un desarrollador de *Apple* se puede referir al apartado *"2.1.1 Suscripción como desarrollador de Apple"*.

Al ingresar al portal de *iTunes Connect* lo primero que se debe hacer es acordar los contratos de la parte legal para poder enviar las aplicaciones para revisión y posterior aprobación y distribución en la tienda *AppStore,* lo cual se indicará en el siguiente sub apartado.

[95] iPhone Development Tutorial HQ. (2012). *What is iTunes Connect?* Recuperado el 28 de Marzo de 2013, de What is iTunes Connect?: http://www.iphonedevelopmenttutorialhq.com/?p=261

2.13.3 Acuerdo de contratos, impuestos y banca

Para acordar los contratos, ingresar en el módulo *Contracts, Taxes and Banking* de *iTunes Connect*, donde se encontrará una lista de contratos disponibles en la sección *"Request New Contracts"*. Para generar un nuevo contrato se presionará el botón *"Request"* junto al contrato deseado.

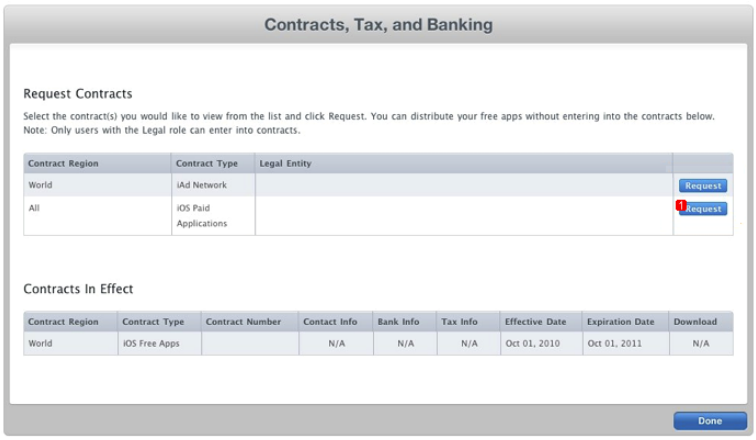


Fig. 2.338: Solicitar un nuevo contrato de aplicaciones de pago de *iOS*

Antes de generar un nuevo contrato, *iTunes Connect* valida la información legal de la persona o empresa. El nombre del desarrollador o empresa será presentado en la tienda *AppStore* como vendedor.

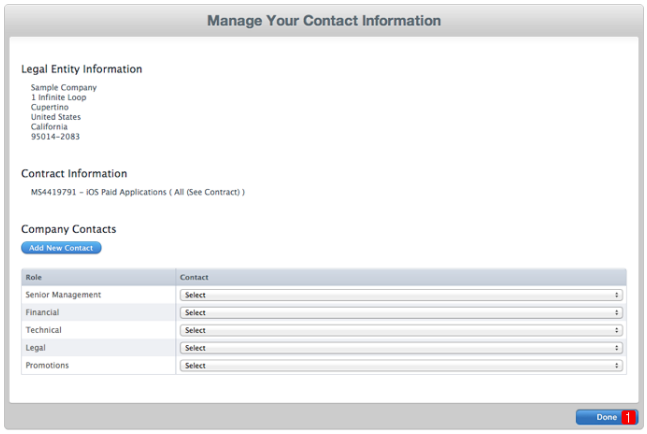


Fig. 2.339: Validación de información legal del desarrollador o empresa[96]

[96] Apple. (2012). *Managing Contracts, Taxes, and Banking*. Recuperado el 30 de Marzo de 2013, de Managing Contracts, Taxes, and Banking: http://developer.apple.com/library/ios/#documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/5_SigningContractsandBanking/SigningContractsandBanking.html#//apple_ref/doc/uid/TP40011225-CH21-SW1

El siguiente paso consiste en ingresar la información bancaria para recibir los pagos por las aplicaciones vendidas; esto es necesario para que el contrato entre en vigencia y se puedan comercializar las aplicaciones.[97] Para agregar esta información, en el modulo de contratos de *iTunes Connect*, se debe presionar *Set Up* en la columna de información bancaria.

Fig. 2.340: Editar información bancaria para que el contrato entre en vigencia

Presionar a continuación el link *"Add Bank Account"*

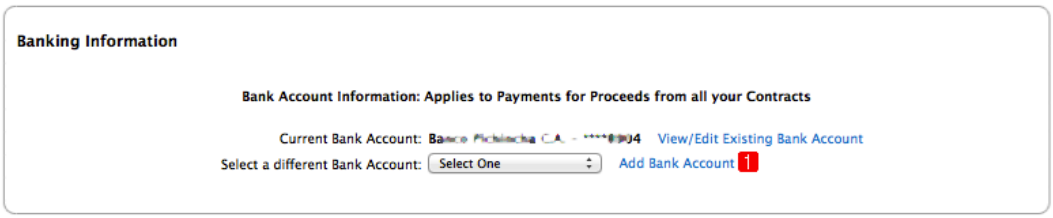


Fig. 2.341: Agregar una cuenta bancaria

Seleccionar el país correspondiente a la cuenta bancaria. Se solicitará después el número de identificación del banco, si se lo desconoce, se puede utilizar la alternativa *"Lookup Transit Number"* para buscar el banco por nombre, ciudad o código postal.[98]

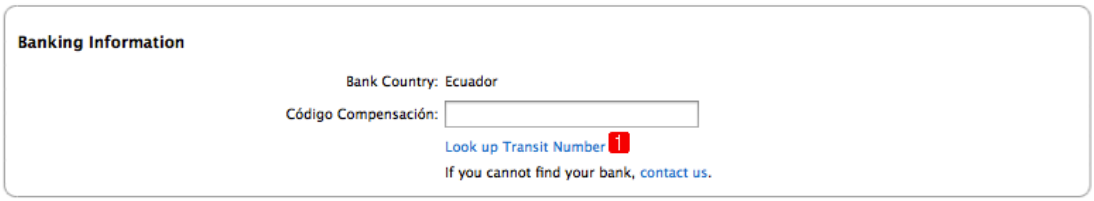


Fig. 2.342: Alternativa para buscar el banco en el cual se dispone una cuenta

[97] Apple. (2012). *Managing Contracts, Taxes, and Banking*. Recuperado el 30 de Marzo de 2013, de Managing Contracts, Taxes, and Banking: http://developer.apple.com/library/ios/#documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/5_SigningContractsandBanking/SigningContractsandBanking.html#//apple_ref/doc/uid/TP40011225-CH21-SW1

[98] Apple. (2012). *Managing Contracts, Taxes, and Banking*. Recuperado el 30 de Marzo de 2013, de Managing Contracts, Taxes, and Banking.

Seleccionar el banco deseado y confirmarlo; finalmente se debe ingresar el numero de cuenta bancaria, el nombre del titular de la misma y el tipo de moneda que maneja.

Fig. 2.343: Ingresar los datos de la cuenta bancaria del banco seleccionado en el paso anterior

Al finalizar este paso, la información de banco y cuenta bancaria aparecerá disponible en el listado de la Fig 2.341; se lo debe seleccionar y presionar el botón *Save.*

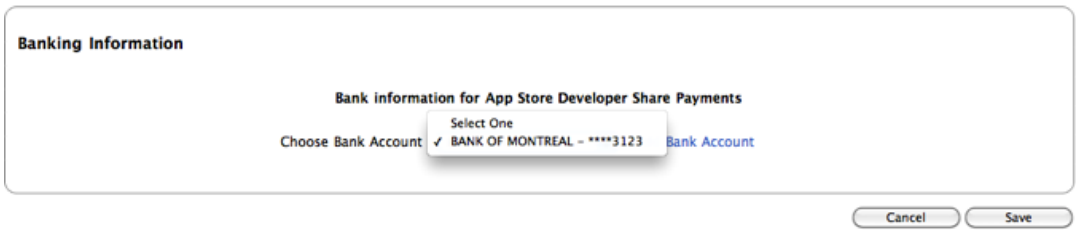


Fig. 2.344: Seleccionar la cuenta bancaria creada que aparece ahora en el listado

El ultimo paso para que el contrato entre en vigencia consiste en acordar los impuestos o *Tax* de Estados Unidos que obligatoriamente se deben pagar sin importar en que país se desarrolle o en que tienda *AppStore* se vendan las aplicaciones.[99] Para esto, se debe presionar el botón *Set Up* correspondiente a la columna *Tax Info* de la Fig. 2.340.

[99] Apple. (2012). *Managing Contracts, Taxes, and Banking*. Recuperado el 30 de Marzo de 2013, de Managing Contracts, Taxes, and Banking: http://developer.apple.com/library/ios/#documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/5_SigningContractsandBanking/SigningContractsandBanking.html#//apple_ref/doc/uid/TP40011225-CH21-SW1

Dependiendo de la dirección de la entidad legal o desarrollador, se solicitará completar uno o varios formularios de impuesto o *Tax*. Si se encuentra fuera de los Estados Unidos, se solicitará responder una serie de preguntas para dirigir al solicitante al formulario o certificación adecuado.

Al completar el formulario o certificación de impuestos, el contrato entrará en vigencia y a partir de este momento se podrá comercializar las aplicaciones en la tienda *AppStore*.[100]

Cabe recalcar que este procedimiento se debe realizar una sola vez. Además, cada vez que se presenten actualizaciones de firmware o aparezcan nuevos dispositivos *iOS*, se podrían cambian algunas clausulas del contrato por lo que *Apple* solicita leer las nuevas normas y volverlo a aprobar.

También existen otros contratos que se deben acordar en caso de:

- Querer publicar aplicaciones gratuitas en *AppStore*
- Querer publicar aplicaciones con *iAds* o publicidad

El siguiente paso para poder publicar una aplicación desarrollada en la tienda *AppStore* consiste en solicitar un certificado de distribuidor y el respectivo *Distribution Provisioning Profile*.

[100] Apple. (2012). *Managing Contracts, Taxes, and Banking*. Recuperado el 30 de Marzo de 2013, de Managing Contracts, Taxes, and Banking: http://developer.apple.com/library/ios/#documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/5_SigningContractsandBanking/SigningContractsandBanking.html#//apple_ref/doc/uid/TP40011225-CH21-SW1

2.13.4 Solicitud de certificado y perfil de distribución

Para iniciar, se debe dirigir al *iOS Provisioning Portal* a la sección *App IDs* con el fin crear un nuevo *App Id* para la aplicación *TwitterTest* que será la que se publicará en la tienda *AppStore*, si se requiere ayuda para crear un *App ID*, se puede referir al apartado *"2.9.1 Pasos para probar una aplicación en un dispositivo iOS físico"*, el *App Id* debería ser como el siguiente:

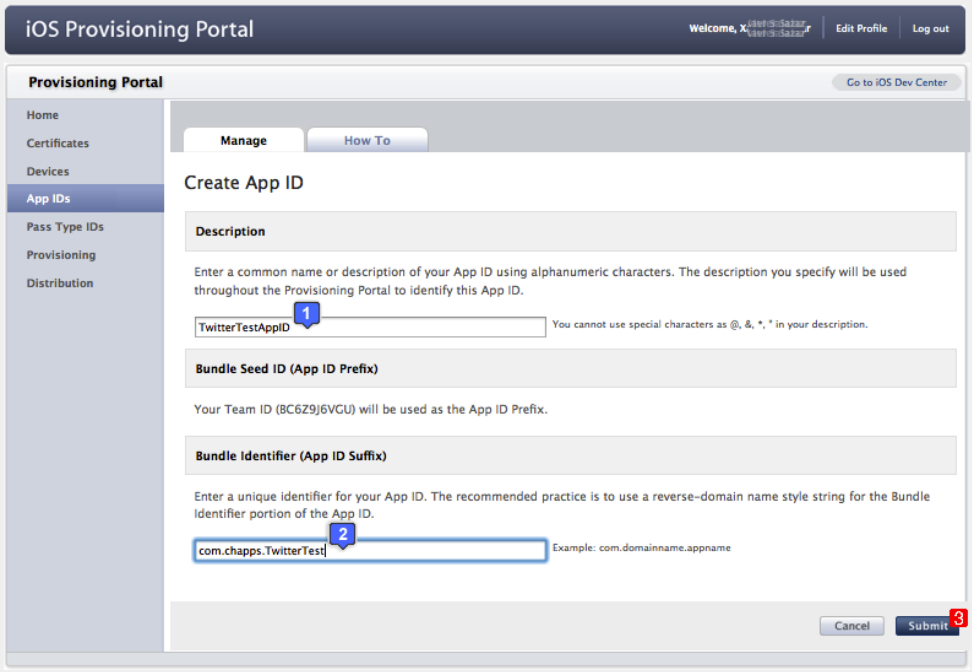


Fig. 2.345: Creación de un *App Id* para la aplicación *TwitterTest*[101]

A continuación se procederá a solicitar el certificado de desarrollador; para esto, en la sección *Certificates* del *iOS Provisioning Portal*, se debe dirigir al apartado *Distribution* y presionar el botón *Request Certificate*, como se indica en la Fig. 2.346.

Acto seguido se debe crear un *Certificate Signing Request* mediante la aplicación *Acceso a Llaveros* del *Mac*, de la misma manera como se hizo en el apartado *"2.9.1*

[101] Apple. (2012). *Apple Developer*. Recuperado el 7 de Abril de 2013, de Member Center: https://developer.apple.com/membercenter/index.action

Pasos para probar una aplicación en un dispositivo iOS físico" y enviarlo a través del portal como se indica en la Fig. 2.347.

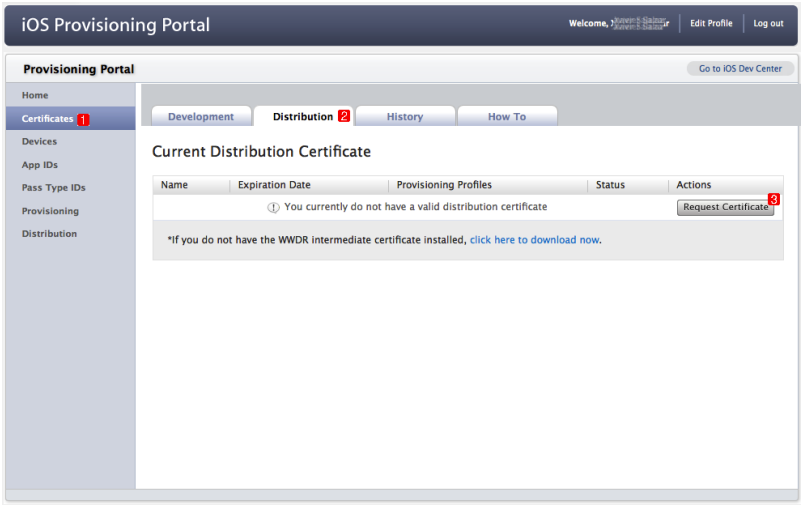


Fig. 2.346: Solicitar un certificado de desarrollador[102]

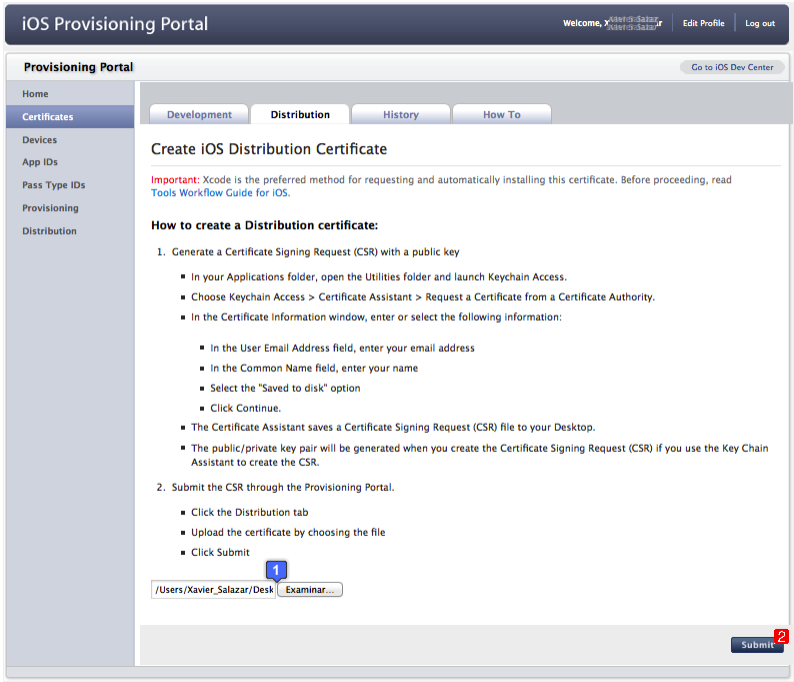


Fig. 2.347: Seleccionar el *CSR* y enviarlo para solicitar el certificado[103]

De esta manera se crea el certificado de desarrollador, pero aun no se dispone de un *Distribution Provisioning Profile*, para esto, se requiere el *App ID* y este certificado que se acaba de crear.

---

[102] Apple. (2012). *Apple Developer*. Recuperado el 7 de Abril de 2013, de Member Center: https://developer.apple.com/membercenter/index.action

[103] Apple. (2012). *Apple Developer*. Recuperado el 7 de Abril de 2013, de Member Center:

A continuación se debe dirigir a la sección *Provisioning* del *iOS Provisioning Portal* y en la pestaña *Distribution*, presionar el botón *"New Profile"*, así:

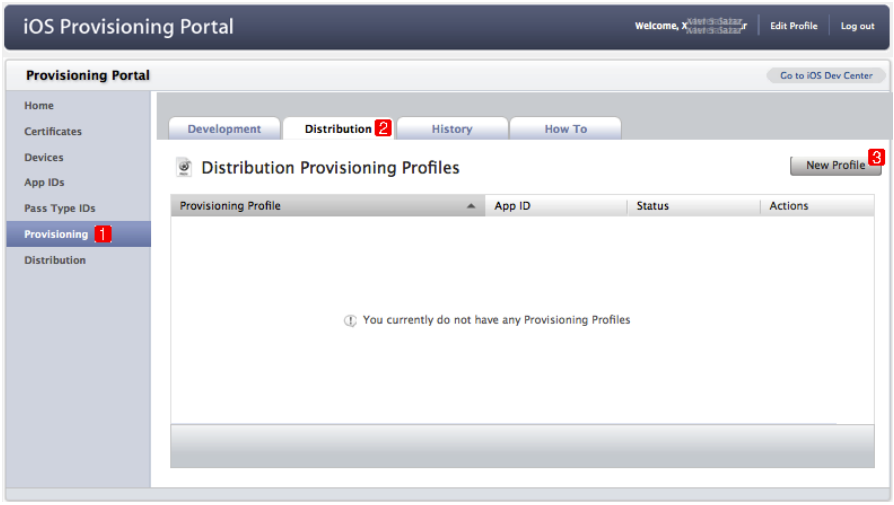


Fig. 2.348: Crear un nuevo *Distribution Provisioning Profile*[104]

Al crear el *Distribution Provisioning Profile* se debe indicar que la aplicación será distribuida a través de la tienda *AppStore*, escribir un nombre para el perfil, seleccionar el *App Id* de la aplicación a distribuir y utilizar el certificado recién creado, como se indica a continuación:

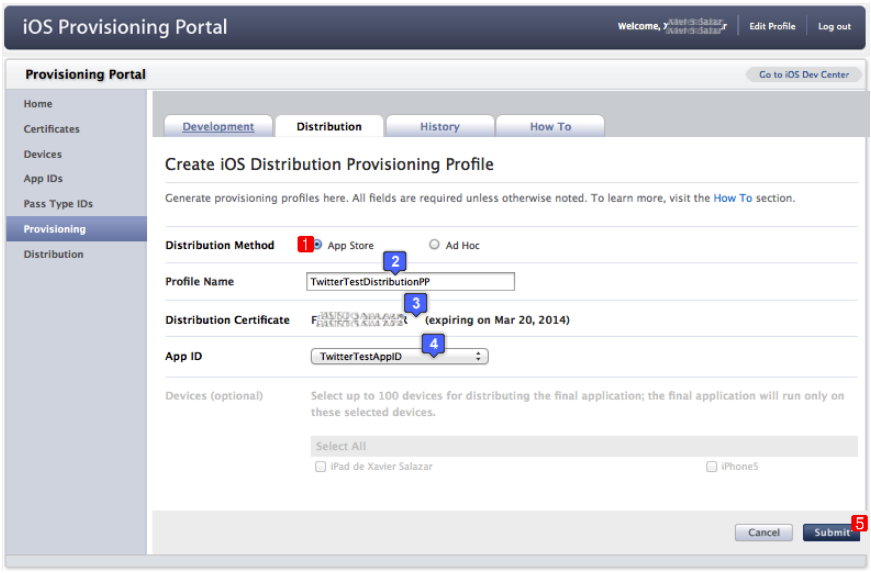


Fig. 2.349: Datos del *Distribution Provisioning Profile*[105]

---

[104] Apple. (2012). *Apple Developer*. Recuperado el 7 de Abril de 2013, de Member Center: https://developer.apple.com/membercenter/index.action

[105] Apple. (2012). *Apple Developer*. Recuperado el 7 de Abril de 2013, de Member Center.

Al finalizar estos pasos, se puede proceder a descargar tanto el certificado como el *Distribution Provisioning Profile*. Se los debe descargar y ejecutar como se indica a continuación:

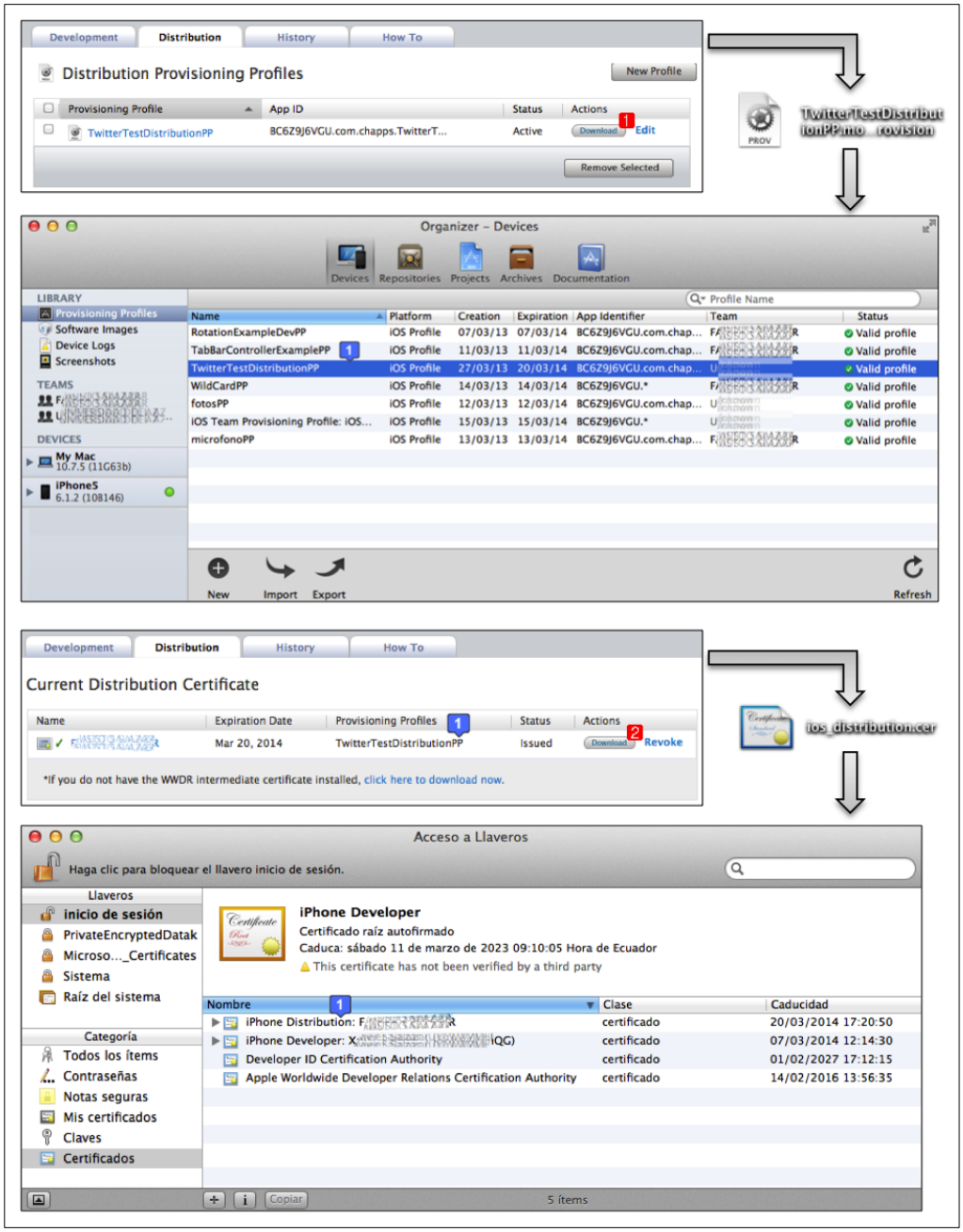


Fig. 2.350: Descargar y ejecutar el *Distribution Provisioning Profile* y el Certificado

A continuación se puede proceder a administrar la aplicación a subir en el portal *iTunes Connect*. Esto se indicará en los siguientes sub apartados.

2.13.5 Requerimientos para administrar aplicaciones en iTunes Connect

Las aplicaciones que se deseen publicar en la tienda *AppStore* deben ser administradas primeramente en el portal de *Apple iTunes Connect,* en donde se la debe ingresar junto a: descripción de funcionamiento, versión, derechos de autor, capturas de pantalla, entre otros. Para poder agregar una aplicación en el portal *iTunes Connect*, se requiere:

Para aplicaciones de tipo universal:

a) El ícono grande de la aplicación en resolución 1024x1024 pixeles, que será utilizado en *AppStore*. Debe ser un arte regular sin esquinas redondeadas en formato *JPEG, JPG, TIFF, TIF*, o *PNG.*

b) Capturas de pantalla de la aplicación en ejecución en dispositivos *iPhone* e *iPod touch* de pantalla de retina de 3.5 pulgadas, en resolución 960x640 pixeles y en formato *JPG* o *PNG*.

c) Capturas de pantalla de la aplicación en ejecución en dispositivos *iPhone 5* e *iPod touch* de quinta generación con pantalla de retina de 4 pulgadas, en resolución 1136x640 pixeles y en formato *JPG* o *PNG.*

d) Capturas de pantalla de la aplicación en ejecución en dispositivos *iPad,* en resolución 1024x768 pixeles en formato *JPEG, JPG, TIFF, TIF*, o *PNG.*

Para aplicaciones exclusivas para *iPhone* e *iPod* se necesitan los requisitos a), b) y c) anteriores; mientras que para aplicaciones exclusivas (Apple, 2012)para *iPad*, se necesitan únicamente los requisitos a) y d) anteriores.[106]

Para realizar capturas de pantalla de la aplicación en ejecución en distintos tipos de dispositivos *iOS*, se debe ejecutar la aplicación deseada en el *iOS Simulator* de *XCode* y

[106] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com

en el menú *Hardware* seleccionar el sub menú "Dispositivo" y dentro de este, seleccionar el dispositivo deseado, como se indica a continuación:

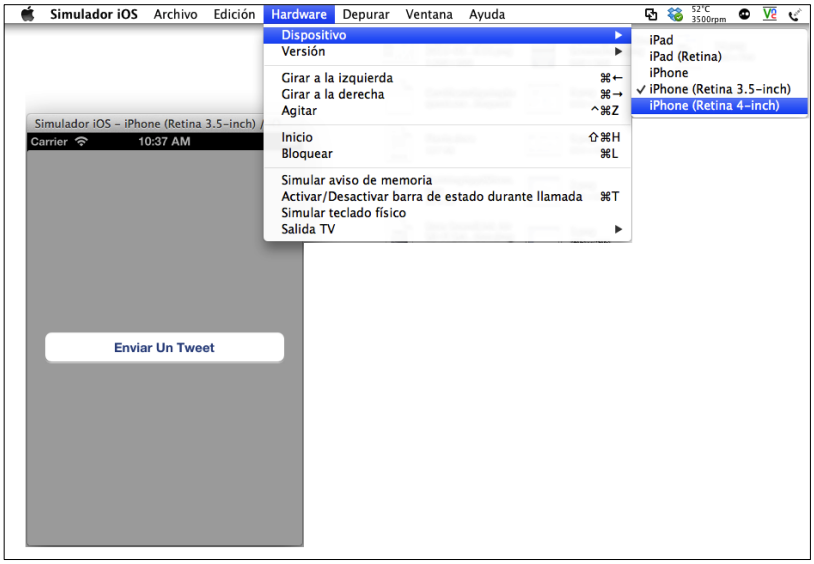


Fig. 2.351: Cambiar el tipo de dispositivo del *iOS Simulator*

El dispositivo suele presentarse en un tamaño superior a la pantalla del *Mac*, para reducir su tamaño y visualizarlo completamente se lo puede escalar en el menú "Ventana".

Fig. 2.352: Escalar para reducir el tamaño del dispositivo iOS simulado

Para realizar una captura de pantalla se debe dirigir al menú “Edición” y dentro de éste seleccionar “Copiar Pantalla”, de esta forma se copia la pantalla al portapapeles en la resolución del dispositivo en el cual se encuentre ejecutando la aplicación.

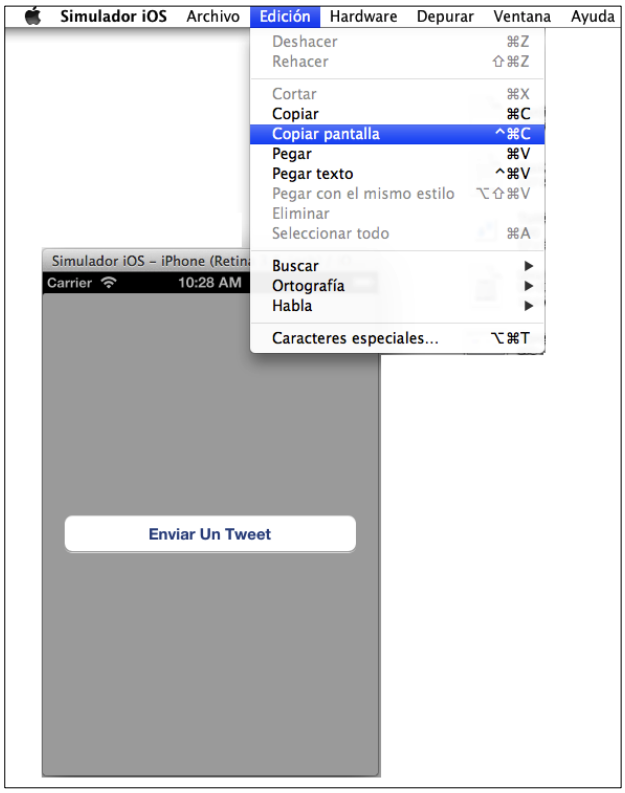


Fig. 2.353: Copiar pantalla de la aplicación en un dispositivo concreto.

Se deben realizar todas las capturas de pantalla necesarias indicadas anteriormente para poder agregar la aplicación en el portal *iTunes Connect.*

Adicionalmente, se debe decidir si la aplicación será gratuita o tendrá un valor comercial. Los precios de las aplicaciones en *Apple* se clasifican en *Tiers.* El “*Tier 1”* corresponde al valor de 0.99 *USD*, el *“Tier 2”* corresponde al valor de 1.99 *USD* y así sucesivamente hasta el *“Tier 50”*, a partir del cual los incrementos de valor por cada *Tier* son de 5 *USD.*[107] Al agregar una aplicación en *iTunes Connect* se puede visualizar la matriz completa de valores monetarios correspondientes a cada *Tier.*

[107] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com

2.13.6 Administración de aplicaciones en iTunes Connect y distribución

Como se ha indicado en el apartado *"2.13.4 Solicitud de certificado y perfil de distribución"*, la aplicación que se enviará para revisión y posterior aprobación para comercialización en la tienda *AppStore* será *TwitterTest* desarrollada en el apartado *"2.8.2 Implementación del Framework de Twitter"*. Para continuar, es necesario disponer de las imágenes de la aplicación en ejecución en los dispositivos en los que será compatible, como se ha indicado en el apartado anterior.

Ahora se debe dirigir al portal *iTunes Connect* e iniciar sesión con los datos de desarrollador. Así:

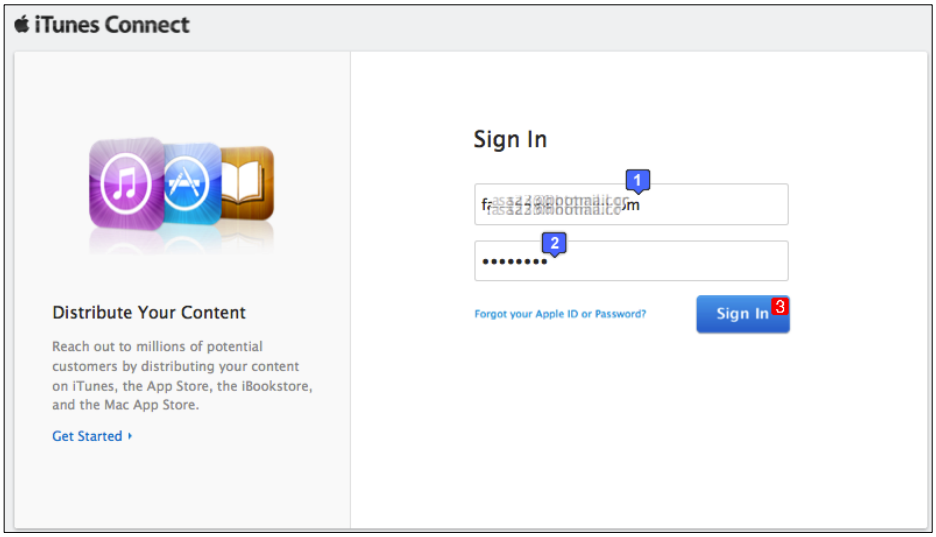


Fig. 2.354: Iniciar sesión en el portal *iTunes Connect*[108]

Para agregar una nueva aplicación se debe dirigir a la sección *"Manage Your Apps"* señalado en la Fig. 2.355. Acto seguido se debe presionar el botón *"Add New App"* en la siguiente ventana presentada, (ver Fig. 2.356). A partir de ese momento se solicitará la información necesaria para agregar la aplicación al portal *iTunes Connect*.

[108] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com

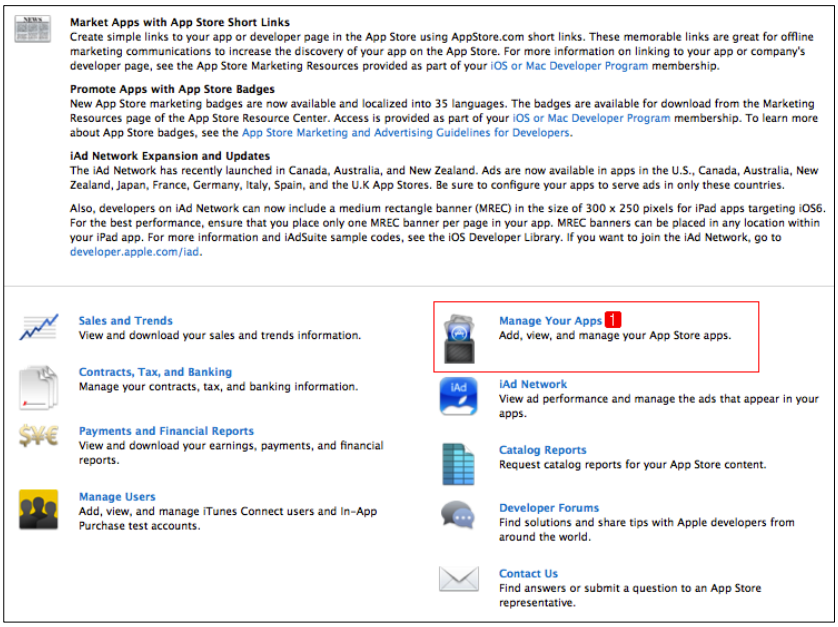


Fig. 2.355: Ingresar a la sección *Manage Your Apps*[109]

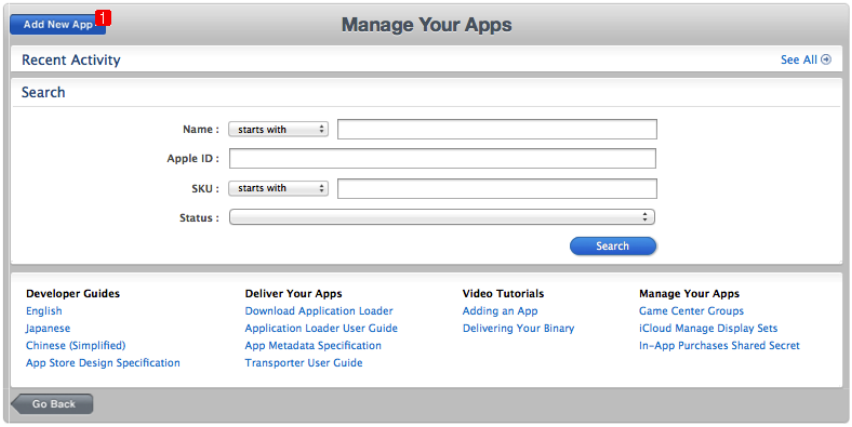


Fig. 2.356: Presionar para agregar una nueva aplicación[110]

La información que se solicita que se ingrese en la primera ventana es el lenguaje principal de la aplicación, nombre de la misma (como se mostrará en *AppStore*), un identificador único que puede contener letras y números, y; su respectivo *Bundle ID* que se lo seleccionará de un listado de *Bundle IDs* disponibles en el *iOS Provisioning Portal*. Completar esta información y presionar el botón *"Continue"*.

[109] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com
[110] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps.

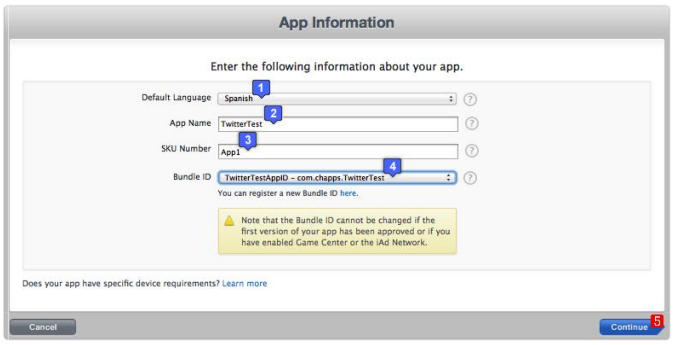


Fig. 2.357: Ingresar la información de la aplicación y su *Bundle ID*[111]

A continuación se debe seleccionar la fecha en la que se desea que este disponible la aplicación y su valor comercial. Se debe considerar que el proceso de revisión y aprobación es de siete días aproximadamente, por lo que la fecha de disponibilidad será tomada en cuenta siempre y cuando la aplicación haya sido revisada y aprobada y aún no haya pasado esta fecha, caso contrario se la publicara en *AppStore* tan pronto ésta sea aprobada.

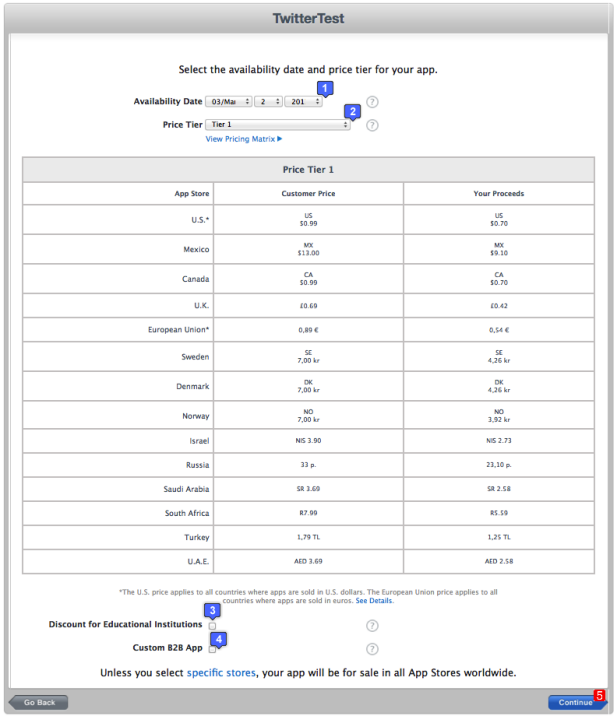


Fig. 2.358: Ingresar datos financieros y fecha de disponibilidad de la aplicación[112]

[111] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com
[112] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps.

Se debe tomar en cuenta que el valor comercial correspondiente al *Tier* seleccionado, no será el valor que reciba directamente el desarrollador, sino que recibirá éste valor menos los impuestos de *Apple* y el *Tax* correspondiente.

Al seleccionar el *Tier* correspondiente al precio de la aplicación se presenta una tabla de las principales monedas del mundo, en donde se indica el valor comercial de la aplicación y la remuneración que recibirá el desarrollador después de los impuestos pertinentes. Si se desea ver un listado completo de los precios correspondientes a cada *Tier* existente se puede presionar el link *"View Pricing Matrix"* de la Fig. 2.358.

Si se desea ofrecer un descuento sobre valor de la aplicación para instituciones educativas, se debe marcar el casillero 3 de la figura anterior. Los detalles de este descuento se encuentran en el contrato acordado previamente para poder comercializar aplicaciones de pago.

Un aplicación *B2B,* estará disponible únicamente para clientes del programa de compras en volumen que se especifique en el portal *iTunes Connect.*[113] Continuar presionando el botón *"Continue"*.

La siguiente ventana que se presentará, es relativamente extensa, debido a que en ella se ingresarán varios datos de la aplicación, para explicar mejor cada campo, se los ha numerado y dividido en las figuras 2.359, 2.360 y 2.361.

En la Fig. 2.359 se debe iniciar por ingresar el numero de versión de la aplicación que se agregará, por ejemplo: *"1.0", "1.0.1",* etc.

En el siguiente campo se debe ingresar el *copyright* de la aplicación, es decir, el nombre de la persona o entidad que posee los derechos exclusivos de la aplicación, precedido por el año en que se obtuvieron los derechos, por ejemplo: *"2013 Xavier Salazar"*.

[113] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com

En el tercer y cuarto campo se debe seleccionar la categoría y subcategoría que mejor describe a la aplicación que se está añadiendo.

La sección marcada con número cinco, describe la clasificación de la aplicación basada en el nivel de frecuencia en el que se presentan en la aplicación algunas características como violencia, contenido para adultos, uso de drogas, entre otros. Para cada característica se debe indicar si no se encuentra, si se la encuentra infrecuentemente o se la encuentra frecuentemente. Al finalizar, se presentará la edad mínima sugerida de los usuarios para los que se recomienda la aplicación; en este caso *"4+"*.

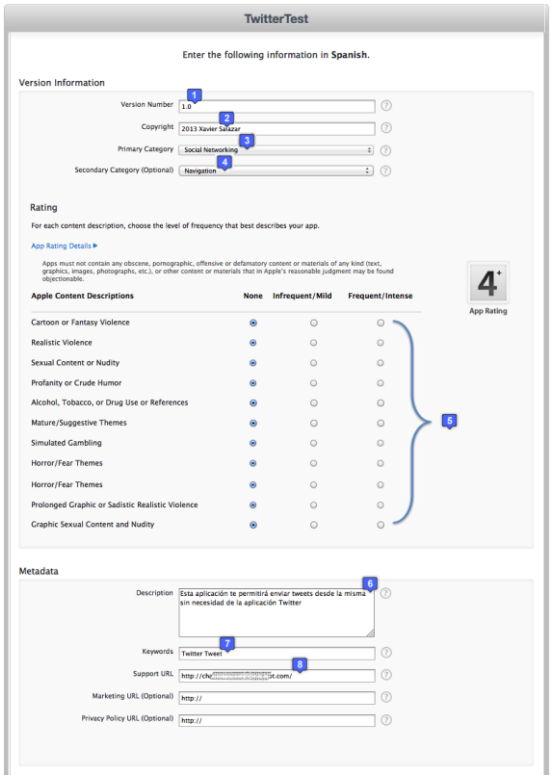


Fig. 2.359: Ingreso de versión, *copyright*, categoría y *rating* de la aplicación[114]

A continuación en la sección *Metadata* de la Fig. 2.359, se debe ingresar en el apartado *Description* (6), la descripción de la aplicación, detallando sus características y funcionalidad, la descripción no puede exceder los 4000 caracteres.

[114] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com

En el campo *Keywords* (7), se deben ingresar palabras que describan la aplicación. Cuando los usuarios buscan en *AppStore*, los términos que ingresan son comparados con estas palabras para presentar resultados mas precisos. Si se ingresan varias palabras, se las deben separar con comas.

El campo *Support URL* (8) es obligatorio, en el mismo que se debe ingresar la dirección *URL* que provea soporte para la aplicación, ésta será visible para los clientes.

En el apartado *Marketing URL* se puede ingresar un *URL* que brinde información sobre la aplicación, si se la ingresa, será de igual manera visible para los clientes en *AppStore*.

Si se desplaza hacia la parte baja de la ventana anterior, se encuentran más apartados que deben ser completados para agregar la aplicación, los cuales se presentan en la Fig. 2.360.

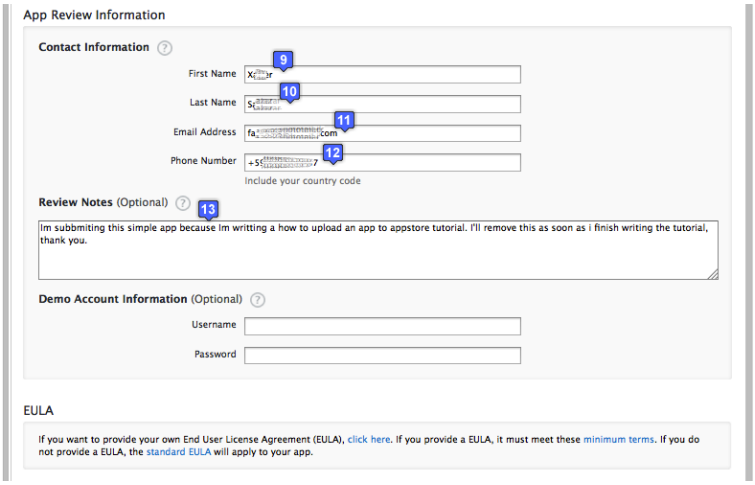


Fig. 2.360: Completar información de revisión de la aplicación[115]

En la sección *"App Review Information"*, en el campo *Contact Information*, se deben ingresar los datos del desarrollador. En el campo *Review Notes* (13), el cual es opcional, se puede llenar con información adicional sobre la aplicación que podría ayudar durante el proceso de revisión de la misma, por ejemplo configuraciones específicas de ésta, entre otros.

[115] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com

Finalmente la sección *Uploads* presentado en la Fig. 2.361, debe ser completada con las capturas de pantalla de la aplicación en ejecución en cada dispositivo en los que será compatible, así como su ícono, como se ha indicado en el apartado *"2.13.5 Requerimientos para administrar aplicaciones en iTunes Connect"*. Finalizar presionando el botón *Save*.

Fig. 2.361: Ingresar el ícono y las capturas de pantalla de la aplicación[116]

A partir de este momento, la aplicación entra en estado *"Prepare For Upload"*, en donde, para continuar se debe presionar el botón *View Details*, como se indica a continuación:

[116] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com

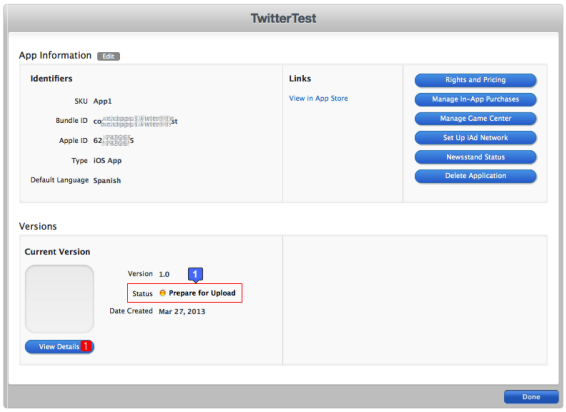


Fig. 2.362: Aplicación en estado *Prepare for Upload*[117]

Al presionar el botón antes mencionado, se presenta una ventana con los detalles de la aplicación a subir, se debe verificar que estos datos sean correctos y presionar el botón *"Ready To Upload Binary"*, así:

Fig. 2.363: Verificar datos de la aplicación y continuar[118]

A continuación se debe indicar si la aplicación utilizará criptografía para la información que manejará, e indicar que se encuentra listo para enviar la aplicación para la revisión, así:

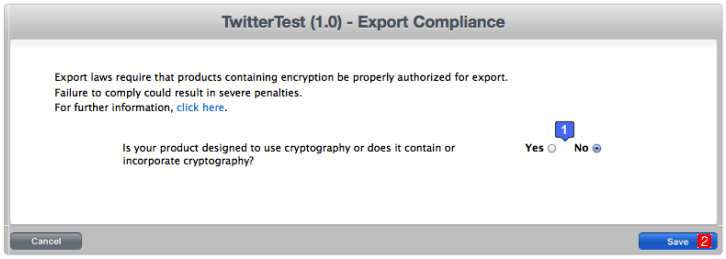


Fig. 2.364: Uso de criptografía en la aplicación[119]

[117] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com
[118] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps.
[119] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps.

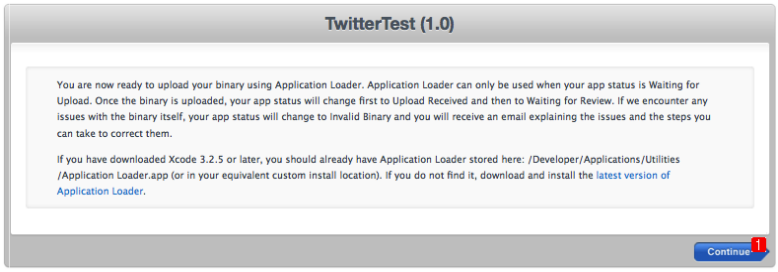


Fig. 2.365: Indicar que se está listo para enviar la aplicación para revisión[120]

Como se puede observar en la figura anterior, *Apple* recomienda el uso de *Application Loader* para el envío de la aplicación para revisión; pero, se lo puede realizar utilizando el mismo *IDE XCode*, como se explicará posteriormente. A partir de este momento, la aplicación pasa a estado *"Waiting For Upload"*.

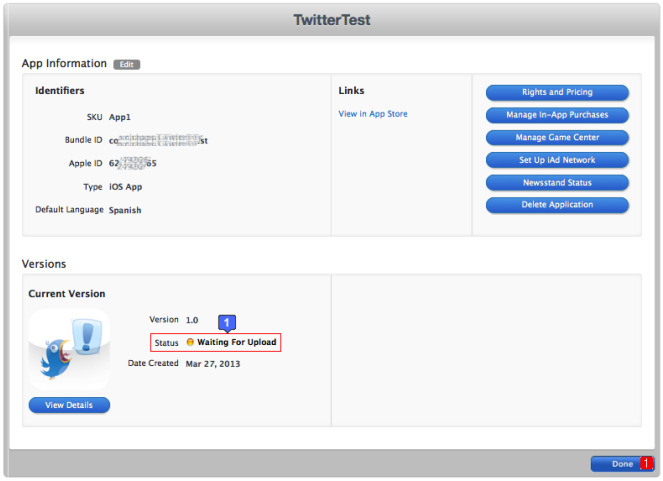


Fig. 2.366: Aplicación en estado *Waiting For Upload*[121]

El paso final consiste en enviar la aplicación para revisión mediante *XCode*, para lo cual se debe abrir el respectivo proyecto. Cabe recalcar que cualquier aplicación a enviar debe tener un ícono, si se requiere ayuda para establecer un ícono para la aplicación se puede referir al apartado: *"2.3.2 Asignación un ícono para la aplicación"*.

Dentro de las configuraciones del proyecto en la sección *Build Settings*, en el apartado *Code Signing Entitlements* se debe indicar el certificado que se utilizará para firmar las

[120] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com
[121] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps:

aplicaciones como distribuidor, el mismo que fue solicitado en el apartado *"2.13.4 Solicitud de certificado y perfil de distribución"*.

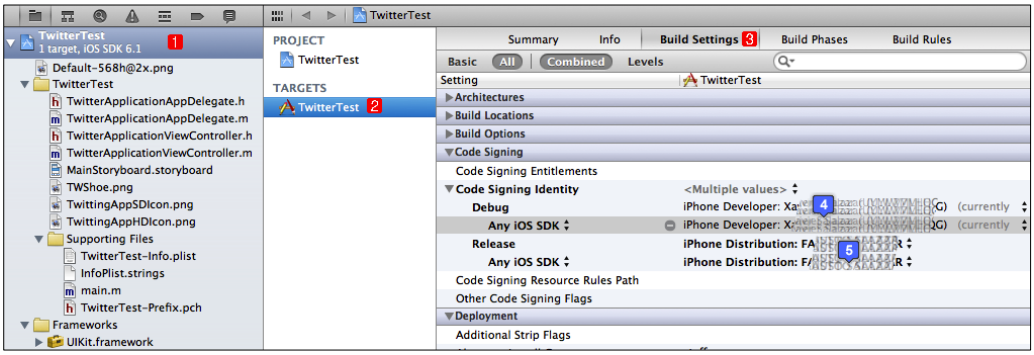


Fig. 2.367: Seleccionar el certificado de distribuidor para el firmado de la aplicación

A continuación se debe conectar un dispositivo *iOS* al *Mac* y modificar el esquema en la parte superior del proyecto de *XCode* para permitir el archivado de la aplicación desarrollada, como se indica a continuación:

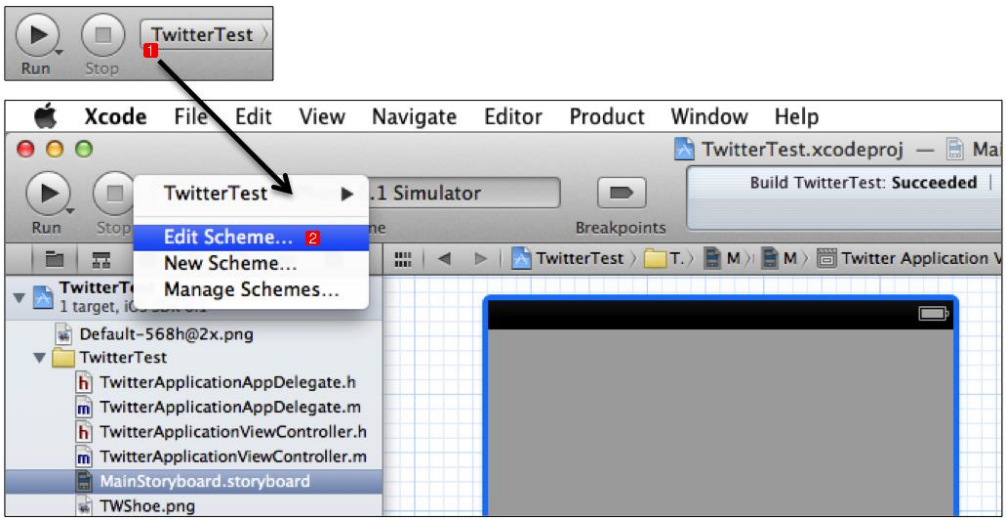


Fig. 2.368: Edición del esquema del proyecto para el archivado de la aplicación

Dentro del esquema se debe seleccionar Ar*chive*, indicar que la configuración de construcción de la aplicación será un *Release* y marcar la opción de mostrar el organizador después de terminar de archivar la aplicación; finalizar presionando *OK*.

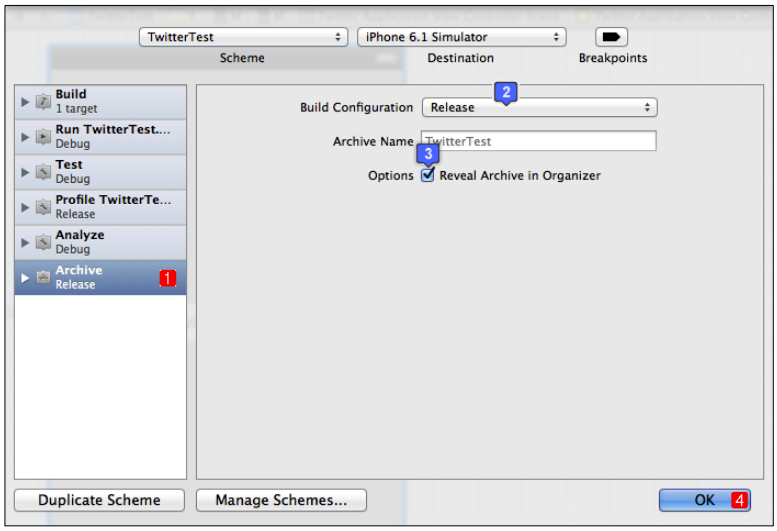


Fig. 2.369: Configuración del esquema de archivado de la aplicación

En la barra superior de *XCode* en el menú *"Product"* se puede ahora proceder a archivar la aplicación como se indica en la siguiente imagen, es importante tener un dispositivo *iOS* conectado al *Mac*, caso contrario esta opción de archivado podría estar deshabilitada.

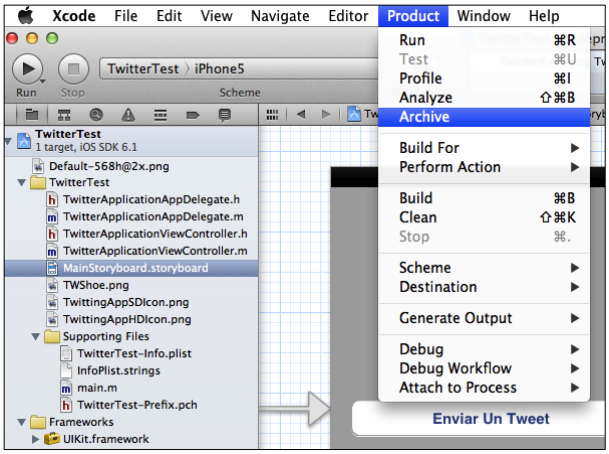


Fig. 2.370: Forma de archivar la aplicación

Fig. 2.371: Autorizar el firmado de la aplicación

Al finalizar el proceso de archivado de la aplicación, ésta se presentará en el organizador de *XCode*, en donde se puede validarla (que no tenga ningún tipo de error) o distribuirla (enviarla para revisión y futura comercialización en la tienda *AppStore*).

Fig. 2.372: Aplicación archivada y presentada en el "Organizador" de *XCode*

Si se desea, antes de enviar la aplicación para distribución se la puede validar presionando el botón "*Validate",* en donde se indica cualquier tipo de error en caso de existir alguno que imposibilitaría la distribución de la aplicación.

Para enviar la aplicación, una vez que ésta se encuentre en estado *"Waiting For Upload"* en el portal *iTunes Connect*, se debe presionar el botón *"Distribute"* de la figura anterior. Al hacerlo, se debe indicar que se desea distribuir la aplicación en la tienda *AppStore*, ingresar los datos de desarrollador y seleccionar la aplicación y el certificado de distribuidor para firmado. Tardará un momento en presentar el mensaje de envío exitoso de la aplicación para futura revisión, de acuerdo al tamaño de la misma; como se indica en la Fig. 2.373.

A continuación, la aplicación pasará a estado *"Upload Received"* y en el transcurso de unos pocos minutos pasará a estado *"Waiting For Review"* como se puede observar en la Fig. 2.374.

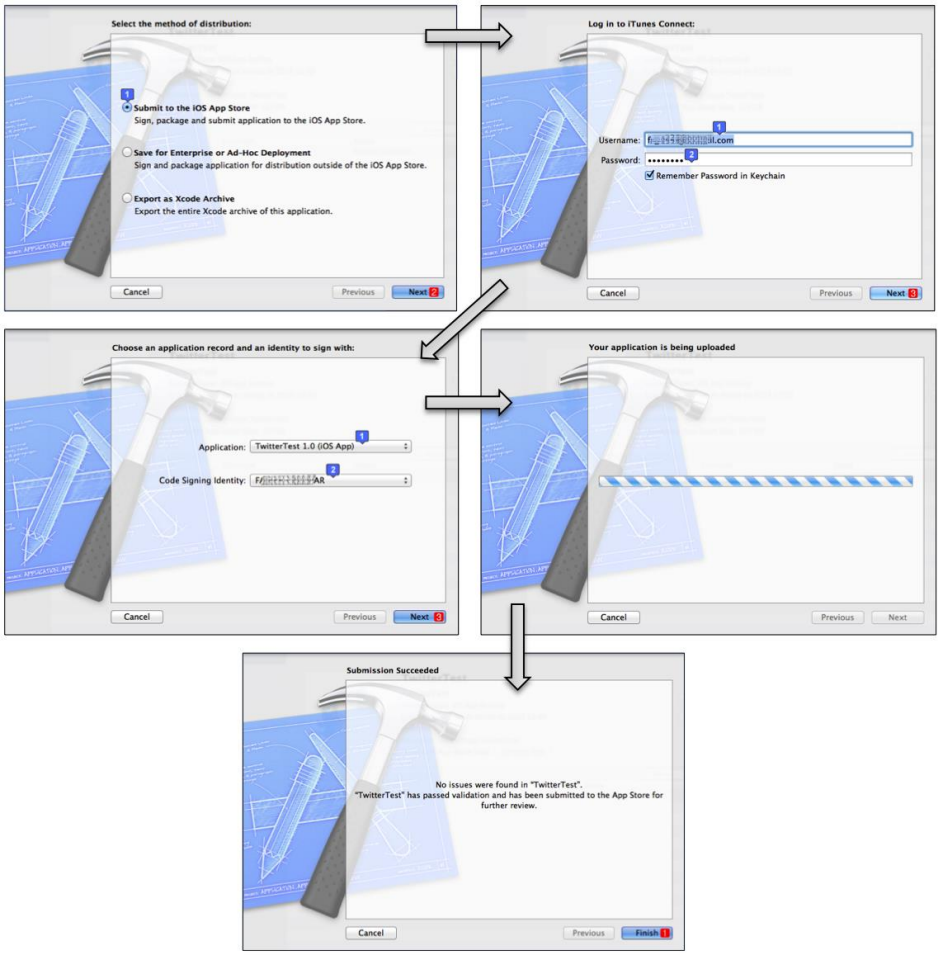


Fig. 2.373: Procedimiento para enviar la aplicación para revisión desde el Organizador de *XCode*

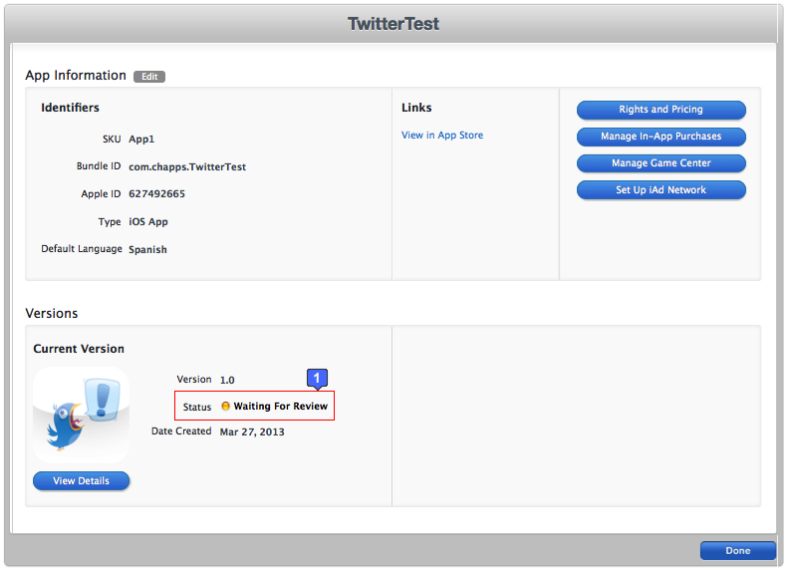


Fig. 2.374: Aplicación en estado *"Waiting For Review"*[122]

[122] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2013, de Manage Your Apps: https://itunesconnect.apple.com

Cada vez que la aplicación cambia de estado en el portal *iTunes Connect*, se reciben notificaciones vía correo electrónico sobre el estado actual de la misma.

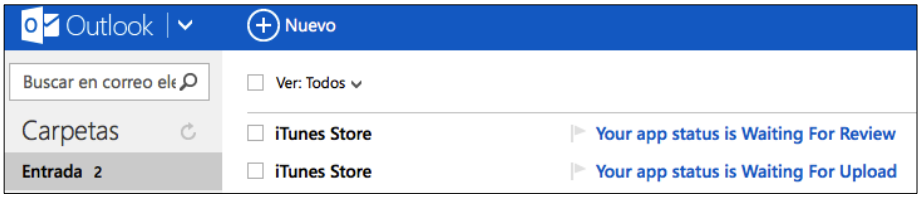


Fig. 2.375: Notificaciones vía correo electrónico del estado de la aplicación[123]

Debido a la gran cantidad de solicitudes de revisión de aplicaciones previo a la aprobación para comercialización en la tienda *AppStore*, aproximadamente después de siete días, la aplicación pasará a estado *"In Review"*. El desarrollador es notificado vía correo electrónico de este cambio de estado. Si no se encuentra ningún problema en la aplicación, se la aprueba y pasa a estado *"Processing for App Store"* y, en el transcurso de las siguientes 24 horas como máximo, pasa a estado *"Ready for Sale"*. Caso contrario, la aplicación es rechazada (estado *"Rejected"*) y se notifica al desarrollador el motivo por el cual no se ha podido aprobar la aplicación.

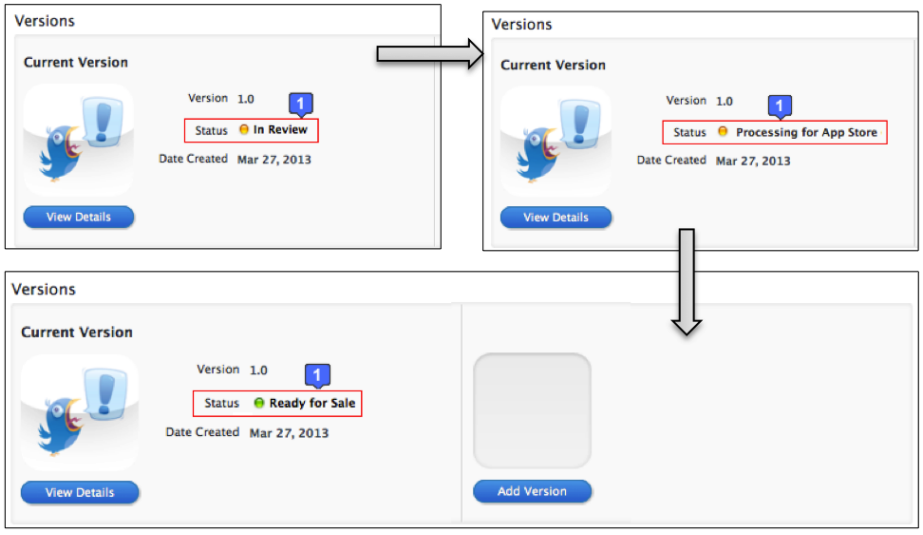


Fig. 2.376: Aplicación aprobada para su comercialización[124]

[123] Outlook. (2012). Correo personal del desarrollador del tutorial. Recuperado el 9 de Abril de 2013, de Outlook.

[124] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2012, de Manage Your Apps: https://itunesconnect.apple.com

2.14 Obtener calificaciones de usuarios para la aplicación

Cuando una aplicación es publicada para su comercialización en la tienda *AppStore*, los usuarios son los encargados de popularizarla de acuerdo a las calificaciones que realicen y reseñas que escriban sobre la misma.

Solo los usuarios que compren una aplicación podrán escribir una reseña sobre la misma y valorarla en un rango de uno a cinco estrellas, de acuerdo a su nivel de satisfacción con el costo y funcionalidad de la aplicación comprada.

Estas reseñas y valoraciones son visibles para otros usuarios que a futuro se interesen en la aplicación y quieran comprarla. Será muy difícil vender una aplicación que este llena de valoraciones bajas y reseñas negativas, debido a que los usuarios dudaran de la calidad y funcionalidad de la misma.

Lamentablemente, el sistema de valoraciones y reseñas, desde el punto de vista del desarrollador de este tutorial, esta orientado hacia malas reseñas, lo que termina afectando las ventas de la aplicación. Se afirma esto, debido a que el usuario satisfecho raramente se toma un tiempo para escribir una reseña positiva sobre la aplicación (debido a que para hacerlo debería dirigirse a la tienda *AppStore*, buscar la aplicación nuevamente y, al encontrarla, valorarla de forma positiva). Los usuarios insatisfechos son casi los únicos que la valoran y escriben reseñas negativas sobre la misma.

Es por esto que el objetivo de este apartado es el de conseguir que los usuarios de la aplicación la valoren de forma positiva y escriban buenas reseñas sobre la misma. Lo ideal, sería que el usuario satisfecho por la funcionalidad de la aplicación sea dirigido automáticamente a *AppStore* a las reseñas de la aplicación y pueda valorarla. Para esto existe la clase *AppiRater* del autor *Arash Payan* la cual será presentada en el siguiente sub apartado.

2.14.1 Motivar a usuarios a valorar positivamente la aplicación mediante AppiRater

El objetivo de la clase *AppiRater* del autor *Arash Payan* es el de motivar a los usuarios satisfechos a valorar positivamente la aplicación y escribir una buena reseña en la tienda *AppStore* sobre la misma[125]. Esta clase puede ser descargada desde el blog de su autor. Su funcionamiento es el siguiente: Cada vez que se ejecute la aplicación en el dispositivo *iOS* del usuario, *AppiRater* revisa si se ha utilizado la aplicación por 30 días y ejecutado al menos 15 veces; si se cumplen estas condiciones, se le solicitará al usuario valorar la aplicación y luego se lo dirigirá directamente a la sección de reseñas de la misma en *AppStore*. El usuario no es obligado a valorar la aplicación ni escribir alguna reseña, debido a que se le presentan las tres siguientes alternativas:

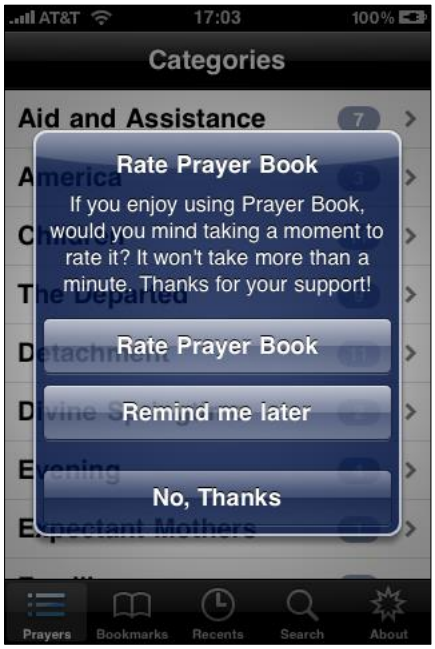


Fig. 2.377: *AppiRater* implementado en la aplicación *Prayer Book*

Al utilizar *AppiRater*, no se garantiza que el usuario valore o escriba una reseña positiva sobre la aplicación, pero se incrementan las probabilidades de que lo haga.

[125] Payan, A. (2009). *Presenting, Appirater*. Recuperado el 12 de Abril de 2013, de Presenting, Appirater: http://arashpayan.com/blog/2009/09/07/presenting-appirater/

Se pueden modificar los parámetros requeridos para presentar la ventana de evaluación de la Fig. 2.377 pero, el desarrollador de la clase *Arash Payan*, sugiere que se necesitan el menos 15 ejecuciones de la aplicación para garantizar que el usuario la haya conocido lo suficiente como para evaluarla.

Por otra parte; se requiere que el usuario este satisfecho cuando vaya a realizar la evaluación o escribir una reseña sobre la aplicación, debido a que el objetivo es conseguir calificaciones y reseñas positivas; es por esto que la clase *AppiRater* incorpora un parámetro denominado *"significant event"*.

El *significant event* de una aplicación debe ser seleccionado cuidadosamente, debido a que será el evento en el cual se produce la satisfacción del usuario; por ejemplo:

En un juego; el *significant event* sería el derrotar a un jefe.

En una aplicación de llamadas *VoIP*, el *significant event* podría ser el culminar una llamada de muy buena calidad, satisfactoriamente.

AppiRater, permite presentar la ventana de evaluación de la aplicación después de un cierto número de *significant events* producidos en la aplicación; de esta manera, el usuario estará satisfecho cuando se le solicite escribir una reseña sobre la aplicación, casi garantizando la evaluación y escritura de una buena reseña sobre la misma.[126]

En el siguiente sub apartado se explicará como incorporar la clase AppiRater a una aplicación.

[126] Payan, A. (2009). *Presenting, Appirater*. Recuperado el 12 de Abril de 2013, de Presenting, Appirater: http://arashpayan.com/blog/2009/09/07/presenting-appirater/

2.14.2 Implementación de AppiRater

En el presente apartado se explicará como implementar la clase *AppiRater* en una aplicación. Para iniciar es necesario descargar la clase desde el blog del autor: http://arashpayan.com/blog/2009/09/07/presenting-appirater/. Una vez descargado, se deben agregar tanto el archivo cabecera como el archivo de implementación al proyecto deseado. Adicionalmente se deben agregar los *frameworks* *CFNetwork*, *SystemConfiguration* y *StoreKit*.[127]

Para este apartado se creará un nuevo proyecto denominado *AppiRaterTest*, utilizando la plantilla *Single View*, cuya vista constará de un único botón (la presión de este, significará un *significant event* en la aplicación) como se indica en la siguiente figura. Si se requiere ayuda, se puede referir al apartado *"2.2 Cómo crear un nuevo proyecto"*.

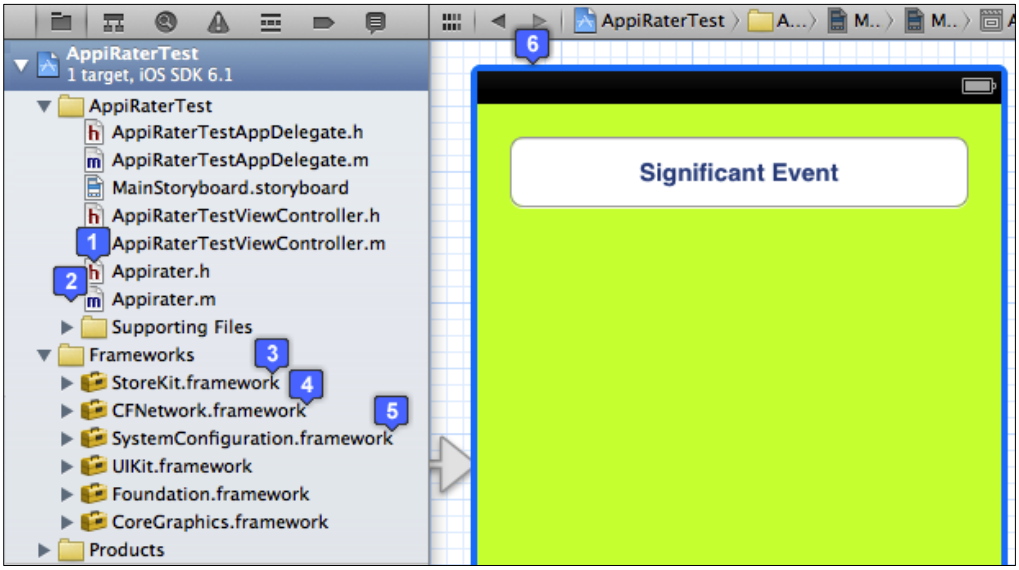


Fig. 2.378: Interface Gráfica de *AppiRaterTest* y ficheros necesarios

A continuación se debe escribir las siguientes líneas de código en el fichero *AppDelegate* de la aplicación; lo cual indicará a la clase *AppiRater* que se ha ejecutado la aplicación.

[127] Payan, A. (2009). *Presenting, Appirater*. Recuperado el 12 de Abril de 2013, de Presenting, Appirater: http://arashpayan.com/blog/2009/09/07/presenting-appirater/

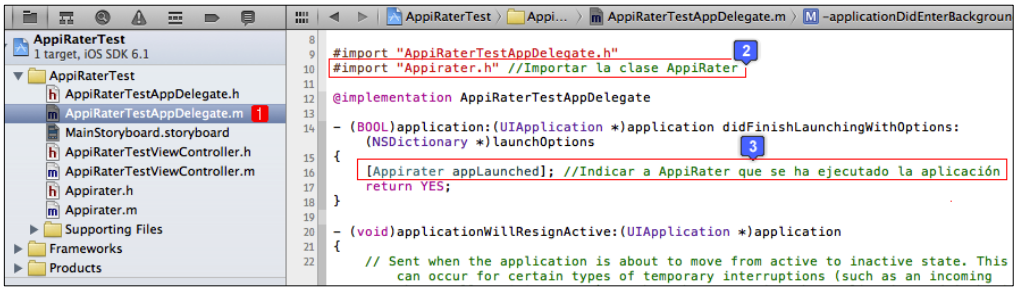


Fig. 2.379: Fichero AppDelegate del proyecto *AppiRaterTest*

Para poder direccionar al usuario a la sección de valoración y reseñas de la aplicación dentro de la tienda *AppStore*, es necesario conocer el identificador de la aplicación en *Apple*, el mismo que puede ser encontrado en el portal *iTunes Connect*, como se puede observar en la Fig. 2.374 en la sección *App Information* en el apartado *Apple ID*. Se debe tomar nota de este número identificador, porque será un parámetro requerido por *AppiRater*.

El fichero cabecera de la clase *AppiRater* define varias constantes entre las cuales, las mas importantes son:

APPIRATER_DAYS_UNTIL_PROMPT: Número de días que el usuario debe tener instalada la aplicación en su dispositivo *iOS* antes de que se presente la ventana de solicitud de evaluación.

APPIRATER_USES_UNTIL_PROMPT: Número de veces que el usuario debe utilizar la aplicación antes de que se presente la ventana de solicitud de evaluación.

APPIRATER_SIG_EVENTS_UNTIL_PROMPT: Número de *significant events* que deben ocurrir para que se presente la ventana de solicitud de evaluación.

APPIRATER_TIME_BEFORE_REMINDING: Una vez presentada la ventana de solicitud de evaluación, si el usuario ha presionado “Recordarme después”, este valor indica el número de días en el que se volverá a presentar la ventana.

APPIRATER_APP_ID: Identificador de la aplicación en *Apple*.

Se debe definir la constante APPIRATER_APP_ID con el identificador de la aplicación en *Apple* correspondiente, y establecer el valor "3" para la constante APPIRATER_SIG_EVENTS_UNTIL_PROMPT, de manera de que después de producido el tercer evento significativo (tercera presión del único botón en la interface gráfica) se solicite al usuario valorar la aplicación. Los parámetros que no se deseen considerar (como cantidad de días o cantidad de usos) para presentar la ventana de solicitud de evaluación, deben ser asignados con el valor -1. Así:

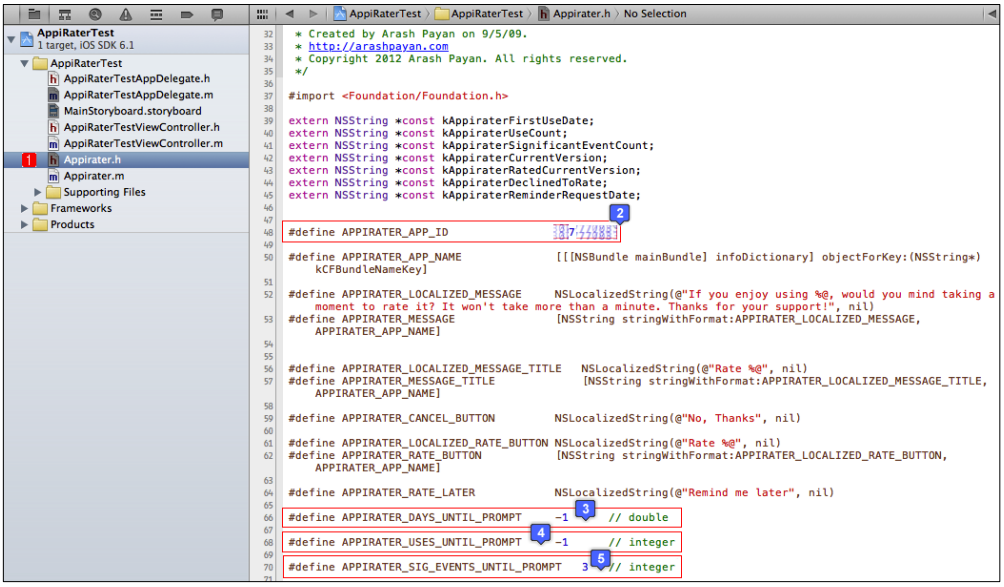


Fig. 2.380: Definición de constantes de la clase *AppiRater*

Para indicar a *AppiRater* que ha sucedido un evento significativo se debe utilizar la instrucción *[AppiRater userDidSignificantEvent:YES]*. Es decir, se escribirá esta instrucción dentro del método *PressButton* asignado al evento presionar el botón; si se requiere ayuda sobre como asignar eventos a componentes de una interface gráfica se puede referir al apartado *"2.5.2 Asignación de métodos a componentes de GUI"*.

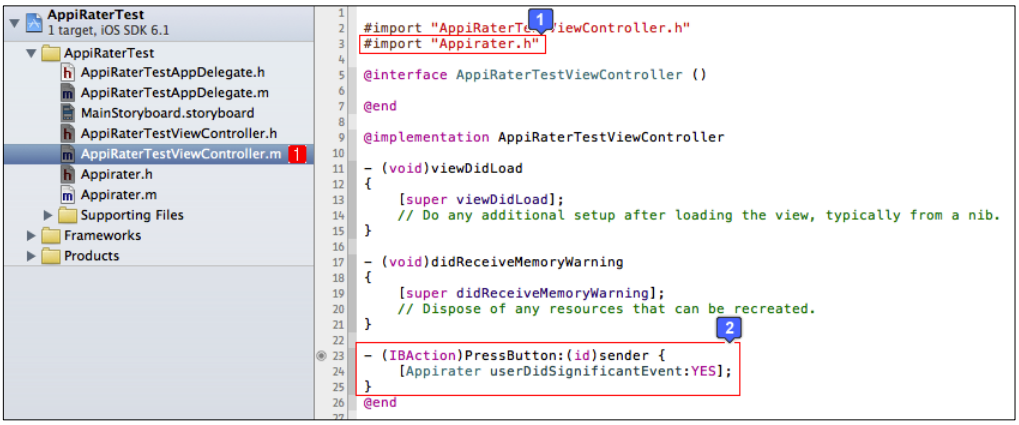


Fig. 2.381: Establecer el *significant event* dentro del método de presión del botón

Lamentablemente la clase *AppiRater* no ha sido escrita utilizando *Automatic Reference Counting* o *ARC*, por lo tanto se encontrarán sentencias de desalojamiento de memoria (*autorelease*) en el fichero de implementación *AppiRater.m*, que generarán errores en el proyecto. Por esto, se debe indicar que esta clase no utiliza *ARC*, escribiendo la sentencia *"-fno-objc-arc"* en la sección *Compiler Flags* correspondiente a la clase *AppiRater* en las configuraciones del proyecto, así:

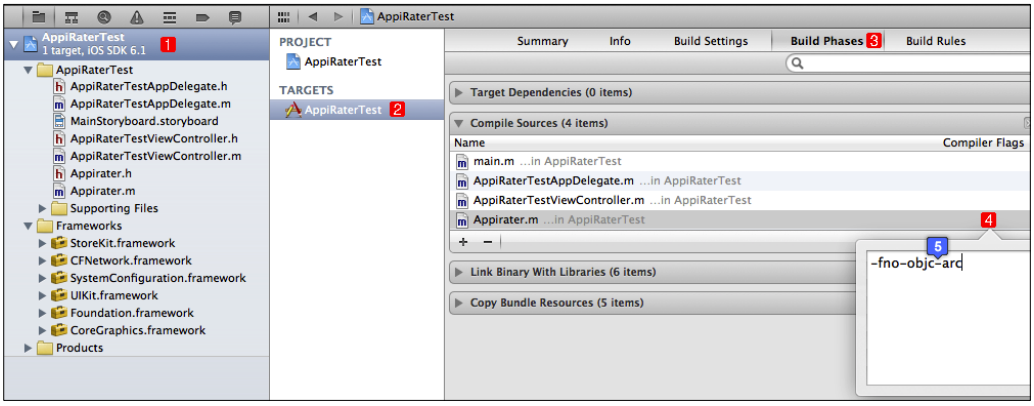


Fig. 2.382: Especificar que la clase no utiliza *Automatic Reference Counting*

Ahora se puede proceder a ejecutar la aplicación para verificar su correcto funcionamiento. La ventana de solicitud de evaluación de la aplicación se presentará

después de presionar el botón tres veces, lo cual simula tres eventos significativos de la misma.

Lamentablemente el simulador *iOS* no permite ejecutar la aplicación *AppStore* en donde se mostrará la aplicación correspondiente al *App ID* especificado en la clase *AppiRater* para poder evaluarla, por esto se recomienda ejecutar la aplicación en un dispositivo físico para verificar su correcto funcionamiento, el cual es el presentado a continuación:

Fig. 2.383: Aplicación *AppiRaterTest* en funcionamiento

2.15 Identificación única de dispositivos iOS

Si una aplicación genera una base de datos interna utilizando el *framework Core Data* y el usuario decide eliminarla, se eliminan absolutamente todos los datos de la aplicación, incluyendo la base de datos generada por la misma.

La estrategia de algunos desarrolladores para conseguir un mayor número de ventas de sus aplicaciones, consiste en desarrollar para la misma aplicación una versión gratuita con funcionalidades limitadas y una versión completa con todas las características y funcionalidades disponibles.

Si se desea aplicar esta estrategia, se presenta el siguiente problema:

Si la aplicación permite, por ejemplo, realizar llamadas *VoIP*, se podría generar una versión gratuita que permita realizar únicamente tres llamadas y; una versión de pago que permita realizar llamadas ilimitadas.

Si el usuario realiza tres llamadas en la aplicación mencionada anteriormente, y el control de llamadas se realiza mediante una base de datos generada por la aplicación mediante *Core Data*, el usuario podría realizar las tres llamadas, eliminar la aplicación y volverla a instalar; de ésta manera obtendría tres nuevas llamadas gratuitas, y nunca compraría la versión completa.

Para solucionar este problema se debe utilizar algún identificador único de los dispositivos *iOS* y almacenarlo en un servidor web, de manera de controlar que si una aplicación ha sido instalada en un dispositivo, al eliminarla y volverla a instalar, se indique al usuario que la aplicación ya fue instalada anteriormente y que si desea disponer de todas las funcionalidades debe comprar la versión completa. El identificador común de los dispositivos *iOS* es:

Identificador UDID:

El identificado *UDID* es una cadena de 40 caracteres compuesto por letras y números, que a manera de un número serial, identifica de forma única a un dispositivo *iOS* en particular, incluyendo *iPhones*, *iPods* e *iPads.* El identificador es generado de acuerdo a varias características de *hardware.*[128] Es posible visualizar el identificador *UDID* de cualquier dispositivo *iOS* en la información que se presenta al conectarlo a *iTunes.*

Fig. 2.384: Identificador *UDID* de un dispositivo *iOS*

Los desarrolladores solían tener acceso a este identificador mediante código fuente, pero lamentablemente algunos de estos desarrolladores y redes publicitarias ajenas a *Apple* utilizaron el *UDID* para relacionarlo con información adicional como localización, nombre, dirección, edad, imagen de perfil de *Facebook*, hábitos de uso de aplicaciones, entre otros; sin conocimiento ni consentimiento de los usuarios.

En el mes de septiembre del 2012, después de un ataque del grupo de *hackers* denominado *Anonymous* a un computador del *FBI*, estos declararon haber hurtado una lista de doce millones de *UDIDs*, los cuales se encontraban relacionados con información adicional sobre los usuarios. Esto representa una violación de privacidad de los usuarios de los dispositivos *iOS*; y, a raíz de esto, *Apple* decidió restringir el acceso del identificador *UDID* por parte de los desarrolladores[129]; por lo tanto, si se envía una

[128] Ars Technica. (2012). *Whats the bid deal with iPhone UDIDs?* Recuperado el 17 de Abril de 2013, de Whats the bid deal with iPhone UDIDs?: http://arstechnica.com/apple/2012/09/ask-ars-whats-the-big-deal-with-iphone-udids/

[129] Ars Technica. (2012). *Whats the bid deal with iPhone UDIDs?* Recuperado el 17 de Abril de 2013, de Whats the bid deal with iPhone UDIDs.

aplicación que acceda al identificador *UDID* para revisión y aprobación para futura comercialización en la tienda *AppStore*, esta será rechazada.

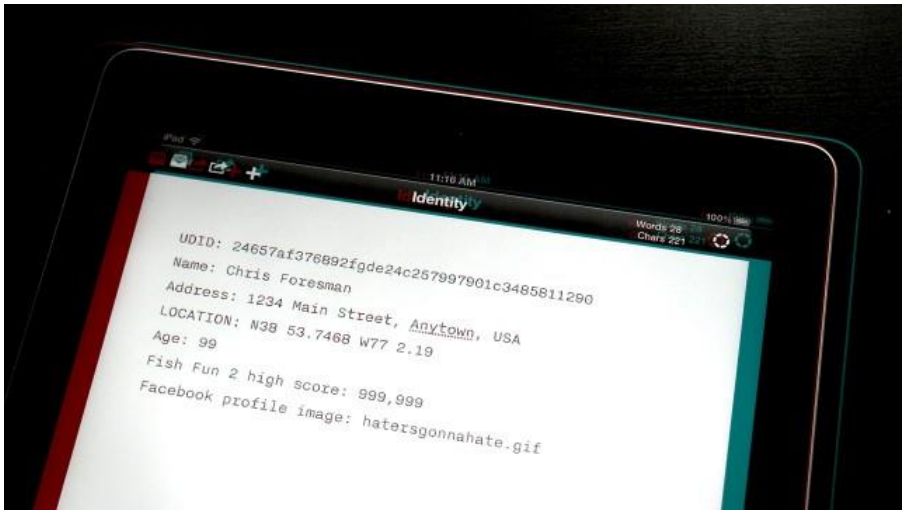


Fig. 2.385: Información que se podría relacionar con un identificador UDID[130]

Es por esto que se debe utilizar otro tipo de identificador para reconocer los dispositivos en los que han sido instaladas las aplicaciones. El identificador sugerido ese el siguiente:

Identificador basado en dirección mac y bundle identifier:

A raíz de que *Apple* decidió rechazar las aplicaciones enviadas para revisión y posterior aprobación para comercialización en la tienda *AppStore* que intenten acceder al *UDID* de los dispositivos *iOS*, apareció una alternativa del autor *Georg Kitz*, el cual, desarrolló una clase que permite generar un identificador único a partir de una combinación de la dirección *mac* y el *bundle identifier* de la aplicación. La clase permite también generar un identificador global basándose únicamente en la dirección *mac* del dispositivo, la cual, al no utilizar el *bundle identifier*, no varía entre aplicaciones[131].

[130] Ars Technica. (2012). *Whats the bid deal with iPhone UDIDs?* Recuperado el 17 de Abril de 2013, de Whats the bid deal with iPhone UDIDs
[131] Kitz, G. (2012). *UIDevice with Unique Identifier*. Recuperado el 18 de Abril de 2013, de UIDevice with Unique Identifier: https://github.com/gekitz/UIDevice-with-UniqueIdentifier-for-iOS-5

La implementación es la siguiente:

- Se deben descargar las clases necesarias desarrolladas por el autor y alojadas en el portal web *GitHub:* https://github.com/gekitz/UIDevice-with-UniqueIdentifier-for-iOS-5

- Copiar las clases *NSString+MD5Addition* y *UIDevice+IdentifierAddition* al proyecto.

- Si el proyecto utiliza *ARC*, se debe especificar que ninguna de estas clases agregadas en el apartado anterior utilizan *ARC*, como se indica en la Fig. 2.382.

- En el fichero del proyecto donde se deseen utilizar los identificadores, importar el fichero *UIDevice+IdentifierAddition.*

- Utilizar *[[UIDevice currentDevice] uniqueDeviceIdentifier]* y *[[UIDevice currentDevice] uniqueGlobalDeviceIdentifier]* para obtener el identificador único y el identificador global único respectivamente.[132]

Se puede referir al proyecto *UniqueIdentifierTest* para comprender la implementación de la clase, este proyecto es presentado en la siguiente figura:

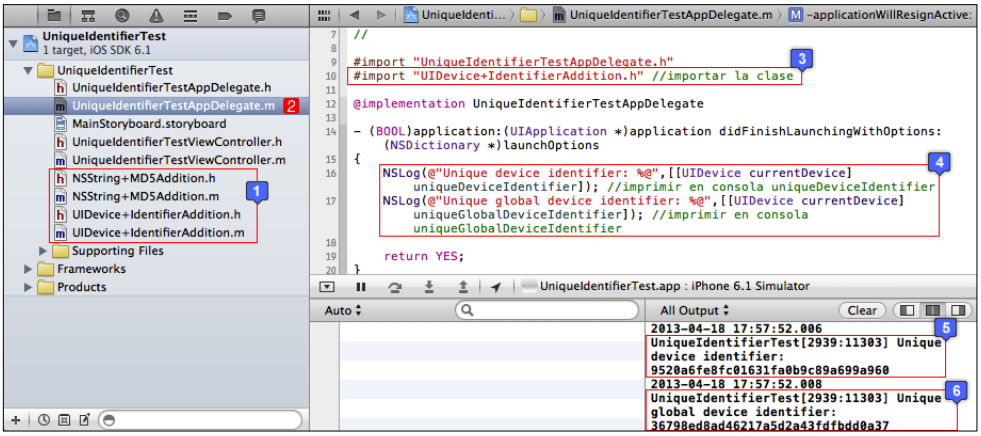


Fig. 2.386: Proyecto *UniqueIdentifierTest*

[132] Kitz, G. (2012). *UIDevice with Unique Identifier*. Recuperado el 18 de Abril de 2013, de UIDevice with Unique Identifier: https://github.com/gekitz/UIDevice-with-UniqueIdentifier-for-iOS-5

La instrucción *NSLog(NSString)* permite imprimir en la consola, una vez que se haya ejecutado la aplicación.

Cabe recalcar que se recomienda utilizar este identificador de forma responsable, comunicando al usuario que se desea acceder a cualquier tipo de información que se requiera, para que éste lo conozca y lo autorice.

Se recomienda almacenar el identificador en una base de datos de un servicio web, sin asociarlo con otros datos de usuario, para que este identificador no sea restringido por *Apple* a futuro, debido a la necesidad que podrían tener los desarrolladores de conocer en que dispositivo se han instalado sus aplicaciones.

A continuación se explicará como implementar un servicio web en una aplicación *iOS*, en el cual se podrían alojar los datos de la misma.

2.16 Implementación del servicio Amazon Web Service: Simple DB

El servicio web *Amazon Simple DB* como se ha indicado en el apartado *"1.4 Servicio Web de Amazon: AWS Simple DB"*, es un almacén de datos no relacionales de alta disponibilidad y flexibilidad, con poca o nula carga administrativa, lo cual facilita la administración de bases de datos, permitiendo al desarrollador encargarse únicamente de almacenarlos y consultarlos mediante el servicio web.

Ofrece una interface de servicios web que permite crear y almacenar conjuntos de datos, también permite obtener resultados a través de consultas. Las respuestas de las consultas son inmediatas debido al indexado de datos que maneja, el mismo que permite cambiar en cualquier momento el modelo de datos, sin ningún problema.

El servicio, se encarga de crear y gestionar réplicas de datos; después, las distribuye geográficamente para brindar alta disponibilidad y capacidad de duración.

El servicio cobra únicamente por los recursos realmente consumidos en cuanto a almacenamiento de datos y distribución de solicitudes.[133] Para conocer la fijación de precios, estructura de datos y la *API* que ofrece el servicio, se puede referir al apartado *"1.4 Servicio Web de Amazon: AWS Simple DB"*.

El objetivo de este apartado, consiste en explicar cómo se realiza la implementación del servicio *AWS Simple DB* en una aplicación *iOS.*

[133] Amazon. (2012). *Amazon SimpleDB*. Recuperado el 27 de Julio de 2012, de Amazon SimpleDB: http://aws.amazon.com/es/simpledb/

2.16.1 Conexión de una aplicación con el servicio AWS Simple DB

Para empezar se debe dirigir al portal web de los servicios ofrecidos por *Amazon*: http://aws.amazon.com/es/ y registrarse para obtener las credenciales de acceso a los mismos. Lamentablemente para el registro, se solicitará el ingreso de una tarjeta de crédito vigente, no existe una tarifa de inscripción, pero se debe pagar el consumo realizado en caso de exceder los límites gratuitos presentados en el apartado *"1.4 Servicio Web de Amazon: AWS Simple DB"*.

Una vez registrado, se pueden acceder a los recursos del portal. Inicialmente se requiere descargar el *SDK* de *iOS*, el cual incluye los *frameworks* necesarios, así como ejemplos de proyectos que implementan los servicios web. Para descargar el *SDK*, se debe dirigir a la dirección http://aws.amazon.com/es/tools/ en donde se encuentran los *SDKs* disponibles para las distintas plataformas, y presionar el enlace *Install*. Así:

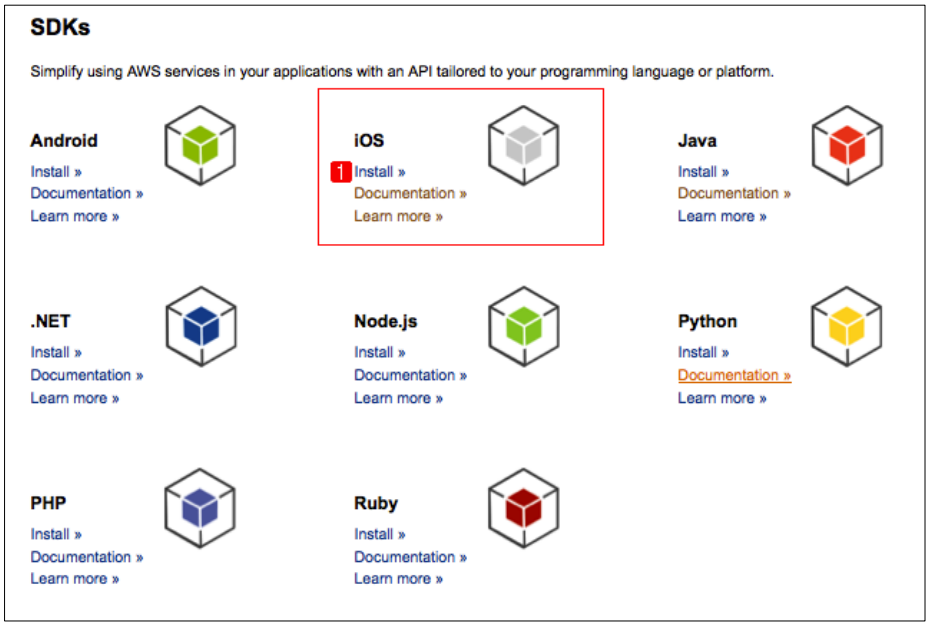


Fig. 2.387: *SDKs* de *AWS* disponibles para las distintas plataformas[134]

[134] Amazon. (2012). *Amazon SimpleDB*. Recuperado el 25 de Abril de 2013, de Tools for Amazon Web Services: http://aws.amazon.com/es/tools/

Al registrarse como usuario, se le asignarán credenciales de seguridad únicas para acceder a los servicios, se debe tomar nota de estas credenciales debido a que son requeridas para poder conectar a la aplicación con *AWS*. Para conocer las credenciales de acceso, el usuario se debe dirigir a su cuenta en el portal de *AWS*, al apartado "Credenciales de Seguridad". Como se indica en la siguiente figura:

Fig. 2.388: Credenciales de Seguridad del usuario registrado[135]

Además, es necesario conocer la existencia de una aplicación de *HTML* y *Javascript*, que permite explorar el *API* de *Amazon SimpleDB*, sin necesidad de escribir código fuente, el cual es conocido como *Scratchpad* y puede ser descargado en la sección de herramientas del desarrollador o *"Developer Tools"* de *Amazon SimpleDB*. El enlace es el siguiente: http://aws.amazon.com/developertools/Amazon-SimpleDB

Al descargar el *Scratchpad*, se debe abrir en un navegador el archivo "*index.html",* en donde se deben ingresar las credenciales de seguridad de la Fig. 2.388 y seleccionar el *API* que se desee explorar como se indica a continuación:

[135] Amazon. (2012). *Amazon Web Services*. Recuperado el 25 de Abril de 2013, de Security Credentials: https://portal.aws.amazon.com/gp/aws/securityCredentials

Fig. 2.389: *Scratchpad* que permite explorar las API de Amazon SimpleDB[136]

A continuación se creará una aplicación que permita grabar y consultar los datos de una asignatura almacenada en el *Amazon SimpleDB*, lo cual se podrá verificar mediante el *Scratchpad*. Para esto es necesario crear un nuevo proyecto utilizando la plantilla *"Single View Application"*, el cual será nombrado *AWSSimpleDBTest*. Adicionalmente, de los *frameworks* incluidos en el *SDK* que se ha descargado anteriormente, se deben agregar al proyecto los *frameworks "AWSRuntime"* y *"AWSSimpleDB"*. Si se requiere ayuda, se puede referir al apartado *"2.8.1 Cómo añadir frameworks"*. Finalizar construyendo una interface de usuario que incorpore un botón que permita almacenar una asignatura en el *SimpleDB* y otro botón que permita mostrar la asignatura almacenada en el mismo, se sugiere crear una interface gráfica similar a la siguiente:

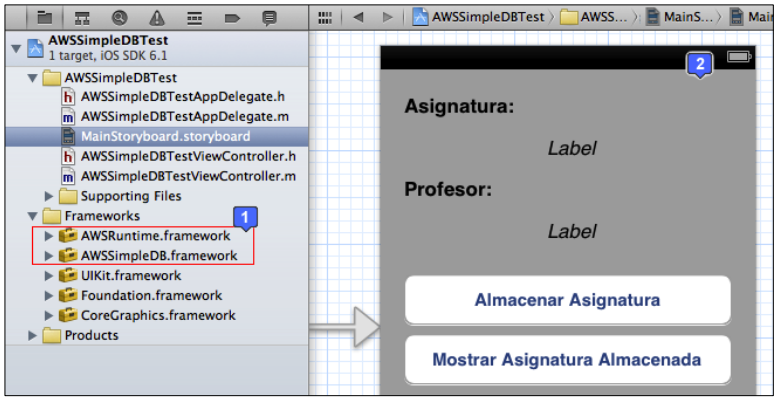


Fig. 2.390: Interface gráfica y ficheros del proyecto *AWSSimpleDBTest*

[136] Amazon. (2012). *Amazon Simple DB*. Recuperado el 25 de Abril de 2013, de Developer Tools: http://aws.amazon.com/developertools/Amazon-SimpleDB

Asignar los eventos *grabarAsignatura* y *mostrarAsignatura* a los botones de la interface correspondientes. De igual manera asignar las variables *lblAsignatura* y *lblProfesor* a las etiquetas correspondientes; si se requiere ayuda se puede referir a los apartados *"2.5.1 Asignación de variables a componentes de GUI" y "2.5.2 Asignación de métodos a componentes de GUI"*. En el fichero cabecera del *View Controller*, se debe crear también una variable para manejar el cliente *Amazon SimpleDB* denominado *sdbClient*, y un método "*crearDominioAsignaturas*" que permita insertar el dominio "Asignaturas" en caso de no existir en la *SimpleDB*. Para crear la variable *sdbClient*, es necesario importar la clase *AWSSimpleDB*. El fichero cabecera del *View Controller* quedaría como el que se indica a continuación:

Fig. 2.391: Archivo cabecera del *View Controller* de *AWSSimpleDBTest*

En el fichero de implementación, se debe importar inicialmente la clase *AWSRuntime*, acto seguido definir constantes para las credenciales de seguridad, el dominio (Asignaturas) y los atributos del mismo (Nombre, Profesor).

En el evento *viewDidLoad*, es decir, al presentar la vista, se debe inicializar mediante las credenciales de seguridad el *sdbClient*, en donde también se indicará su *endpoint* que indica a donde serán enviadas las solicitudes.

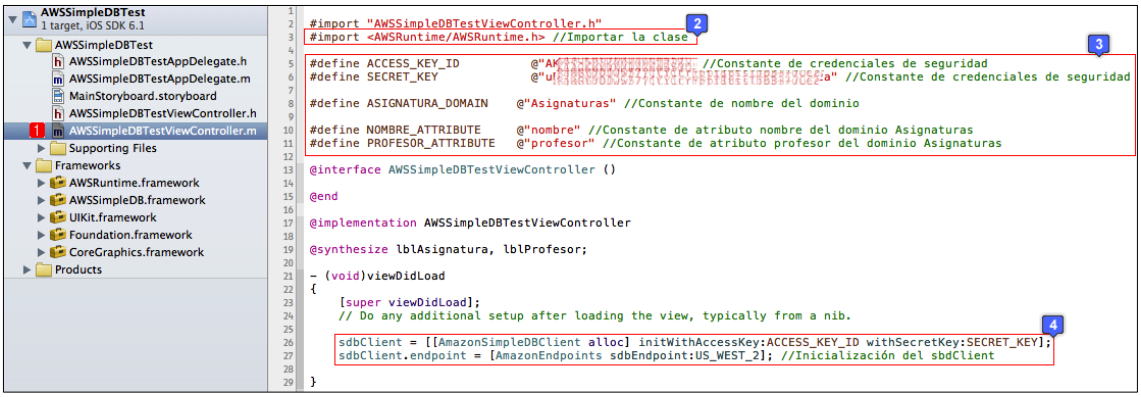


Fig. 2.392: Constantes y *viewDidLoad* del archivo de implementación de *AWSSimpleDBTest*

A continuación se debe implementar el método *crearDominioAsignaturas* que permite crear el dominio "Asignaturas" en la *SimpleDB*, en donde se alojarán las asignaturas con sus respectivos profesores, como se indica a continuación:

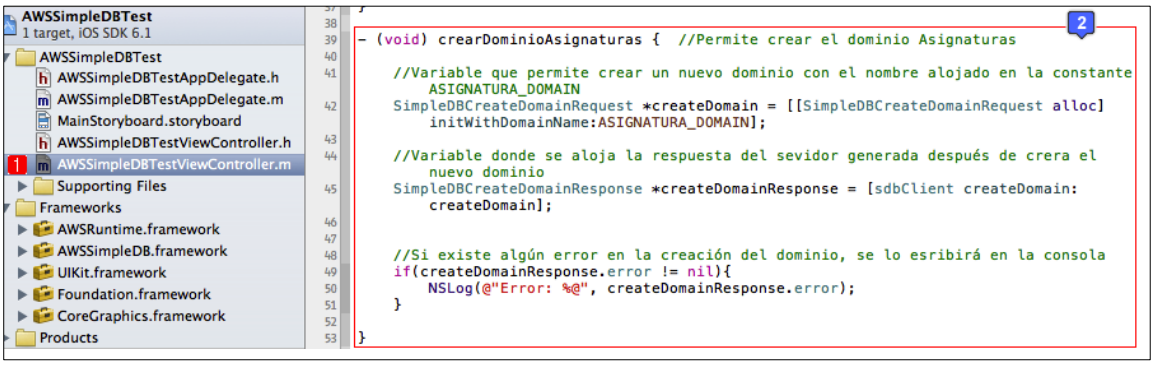


Fig. 2.393: Método que permite crear un dominio en la *SimpleDB*

Finalmente se deben implementar los métodos que permitan grabar y leer los registros grabados en el dominio que se creará en la *SimpleDB.*

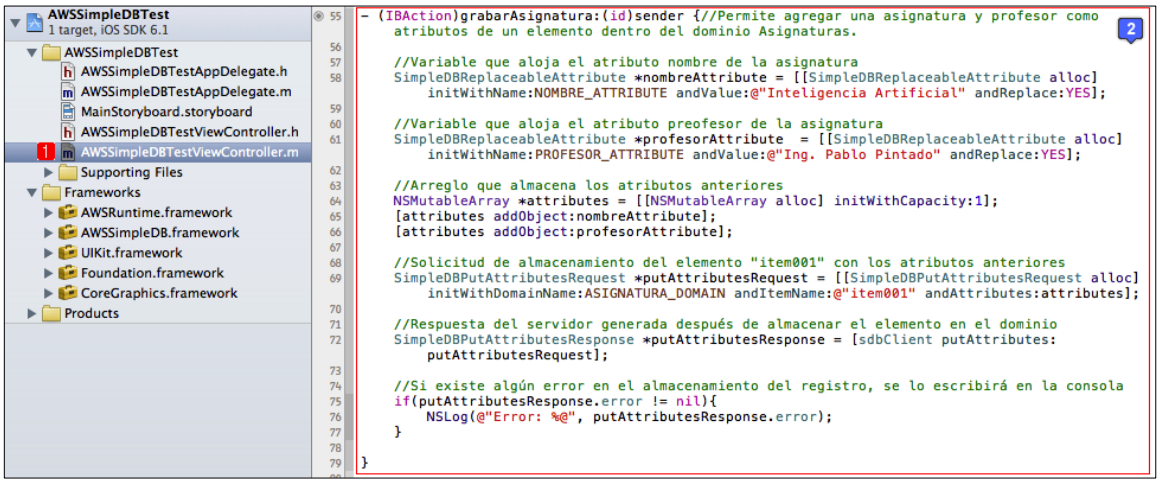


Fig. 2.394: Método que permite almacenar registros en un dominio de la *SimpleDB*

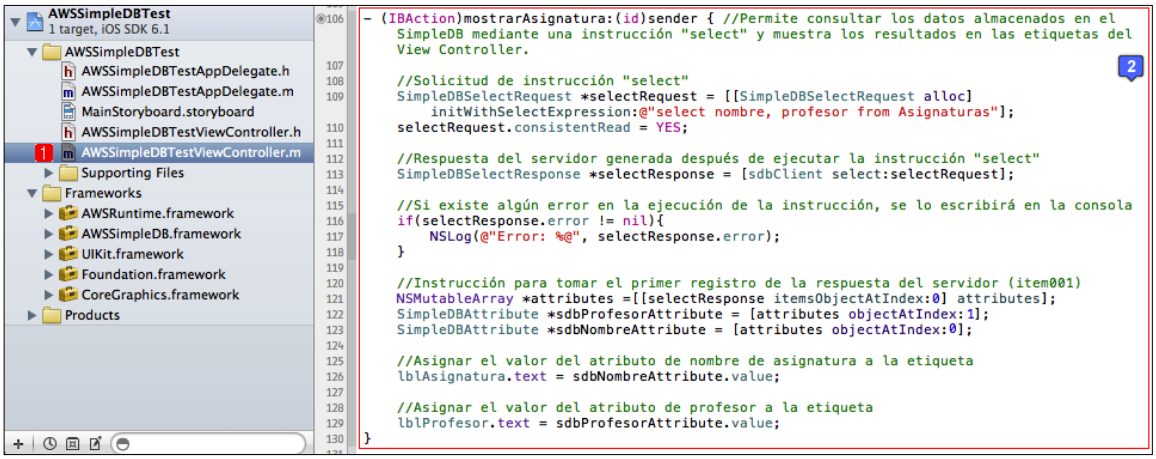


Fig. 2.395: Método que permite leer registros almacenados en un dominio de la *SimpleDB*

Para poder depurar la aplicación y verificar en consola la correcta conexión de la misma con la *SimpleDB*, se deben escribir unas sentencias en el fichero *AppDelegate* de la misma.[137] Cuando se verifique el correcto funcionamiento, se deberían borrar debido a que no se recomienda subir una aplicación a la tienda *AppStore* con estas sentencias de depuración, las mismas que son presentadas a continuación.

[137] Amazon. (2012). *Amazon SimpleDB*. Recuperado el 25 de Abril de 2013, de Tools for Amazon Web Services: http://aws.amazon.com/es/tools/

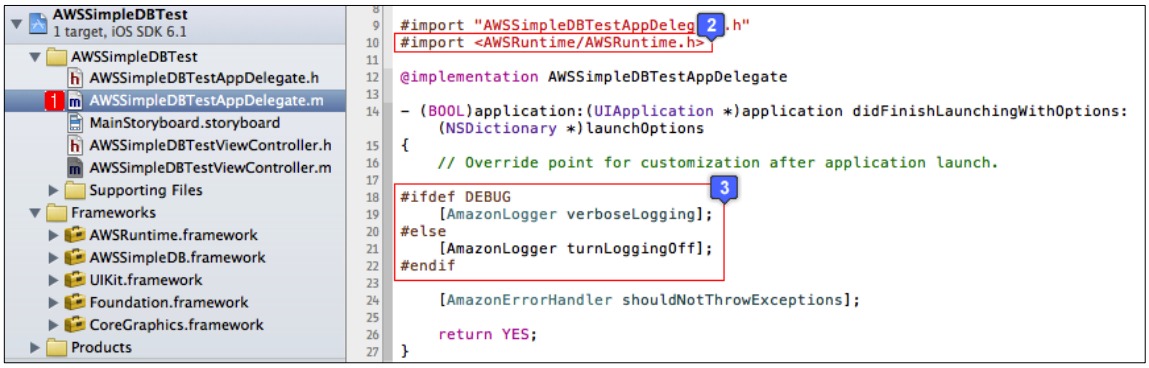


Fig. 2.396: Sentencias que permiten la depuración de la conexión con la *SimpleDB (AppDelegate)*

La primera vez que se ejecute la aplicación se debe llamar al método *crearDominioAsignaturas* para crear el dominio, por ejemplo en el evento *viewDidLoad*. Se lo debe llamar solo una vez y en las siguientes ejecuciones ya no, debido a que ya estaría creado el dominio. Adicionar entonces al evento *viewDidLoad* la instrucción *[self crearDominioAsignaturas];* y ejecutar la aplicación.

Presionar primero el botón "Almacenar Asignatura" y después el botón "Mostrar Asignatura Almacenada", para comprobar que se ha almacenado el registro en el *SimpleDB*. En la práctica, se recomienda implementar un *Activity Indicator* en la aplicación, para que el usuario sepa que la aplicación está enviando y recibiendo información de internet.

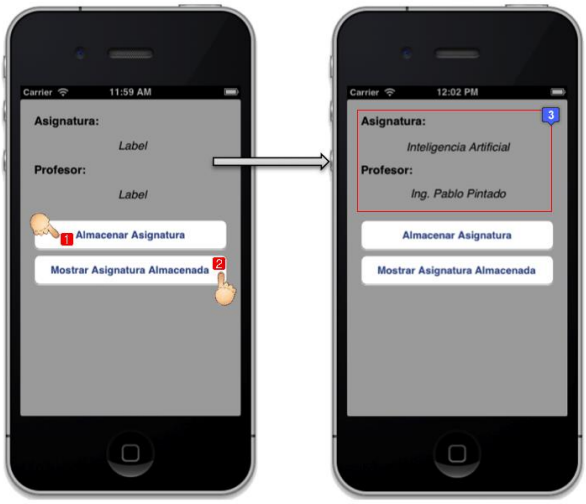


Fig. 2.397: *AWSSimpleDBTest* en ejecución

En todo momento se puede utilizar el *Scratchpad* descargado anteriormente para comprobar el almacenamiento de la información en el *SimpleDB.* Se lo puede abrir y llenar con las credenciales de seguridad; y, si se explora la *API ListDomains*, se debería encontrar el dominio Asignaturas.

Si se explora la *API Select*, se puede ingresar la instrucción *"select * from Asignaturas"* para visualizar los contenidos del dominio, como se indica a continuación:

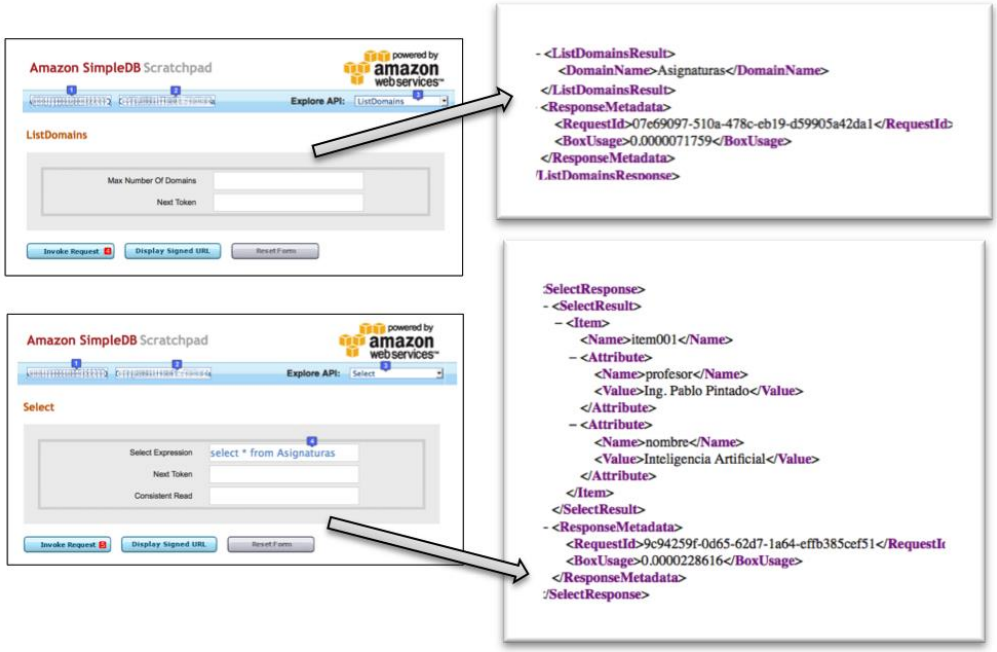


Fig. 2.398: Visualización de la información en el *SimpleDB* mediante el *Scratchpad*[138]

[138] Amazon. (2012). *Amazon Simple DB.* Recuperado el 25 de Abril de 2013, de Developer Tools: http://aws.amazon.com/developertools/Amazon-SimpleDB

2.16.2 Seguridad en el servicio web Amazon SimpleDB

En el apartado anterior, para demostrar el alojamiento de los datos en un *SimpleDB,* se desarrolló una aplicación que incorpora las credenciales de seguridad dentro del código fuente de la misma, lo cual puede ser muy inseguro, debido a que estas credenciales proveen el acceso completo a los recursos *AWS* del desarrollador, así como a su información de facturación. Además, deben ser renovadas con frecuencia por motivos de seguridad[139], y por lo tanto, si son embebidas en una aplicación, se requerirá una actualización de la misma para renovarlas.

La solución para esto, consiste en utilizar un *Token Vendig Machine (TVM)* de *AWS.* Mediante éste, la aplicación utilizará credenciales temporales que expirarán después de cierto período de tiempo; de esta forma, si las credenciales son robadas, tendrán un cierto límite de tiempo de validez, o podrían ser revocadas.

La funcionalidad de un *TVM* sería la siguiente: una aplicación se comunica con un *TVM* para solicitar credenciales temporales para el usuario. El *TVM* solicita las credenciales temporales de un servicio de *tokens* de seguridad de *AWS* y las devuelve a la aplicación. La aplicación utiliza las credenciales temporales para las interacciones con *AWS.* Cuando las credenciales expiran, la aplicación se contacta con el *TVM* para solicitar nuevas credenciales.[140]

---

[139] Amazon. (2012). *Amazon Web Services*. Recuperado el 25 de Abril de 2013, de Security Credentials: https://portal.aws.amazon.com/gp/aws/securityCredentials

[140] Amazon. (2012). *Autenticating Users of AWS Mobile Applications with a Token Vending Machine*. Recuperado el 29 de Abril de 2013, de Autenticating Users of AWS Mobile Applications with a Token Vending Machine: http://aws.amazon.com/articles/4611615499399490

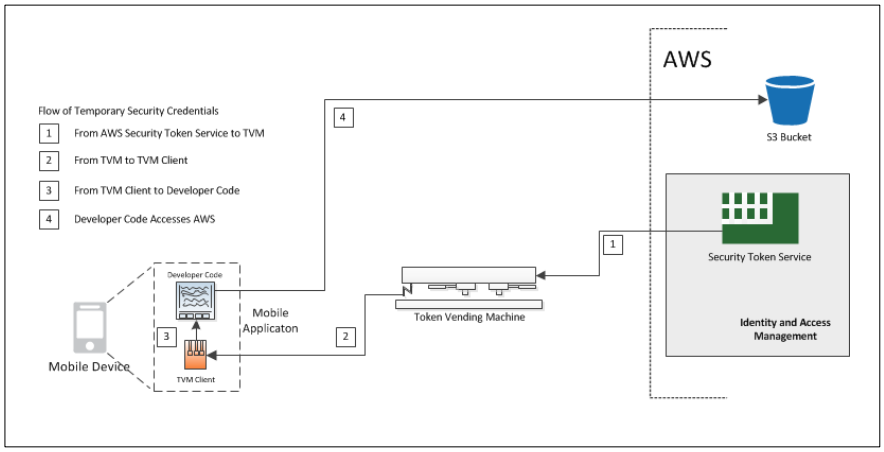


Fig. 2.399: Flujo de credenciales temporales desde *AWS Security Token Service* hasta *iOS App*[141]

AWS provee dos tipos de *TVM: Anonymous TVM* que no almacena información del usuario del dispositivo *iOS e Identity TVM* que soporta registro e inicio de sesión de los usuarios, de manera de limitar las funcionalidades de acuerdo al tipo de usuario identificado. Para almacenar el identificador único de un dispositivo *iOS* no se necesita autenticación del usuario por lo que se trabajará con un *Anonymous TVM*.

Anonymous TVM: La aplicación inicia la comunicación con el *TVM* llamando al método *registerdevice* con un identificador único (*UID*) y una llave secreta. El *TVM* asocia el *UID* con la llave secreta y las almacena para una futura referencia. La llave secreta es usada para asegurar las comunicaciones subsecuentes entre la aplicación y el *TVM.*[142]

141 Amazon. (2012). *Autenticating Users of AWS Mobile Applications with a Token Vending Machine*. Recuperado el 29 de Abril de 2013, de Autenticating Users of AWS Mobile Applications with a Token Vending Machine: http://aws.amazon.com/articles/4611615499399490

142 Amazon. (2012). *Autenticating Users of AWS Mobile Applications with a Token Vending Machine*. Recuperado el 29 de Abril de 2013, de Autenticating Users of AWS Mobile Applications with a Token Vending Machine.

Después de este registro inicial, la aplicación obtiene las credenciales temporales llamando al método *gettoken* con el *UID*, una firma criptográfica y la fecha. La firma criptográfica es un *hash* generado de la fecha usando la llave secreta. Esta firma autentifica el dispositivo con la *TVM:* El *TVM* utiliza el *UID* para buscar la llave secreta y luego la utiliza para verificar la firma.

El *TVM* responde enviando de regreso las credenciales temporales, que han sido encriptados utilizando la llave secreta.

La aplicación desencripta las credenciales temporales utilizando la llave y luego las puede utilizar para acceder a los servicios *AWS* para las cuales han sido autorizadas. Eventualmente, se alcanzará el tiempo de expiración de estas credenciales, en este punto, la aplicación llama al método *gettoken* otra vez para obtener nuevas credenciales temporales.[143]

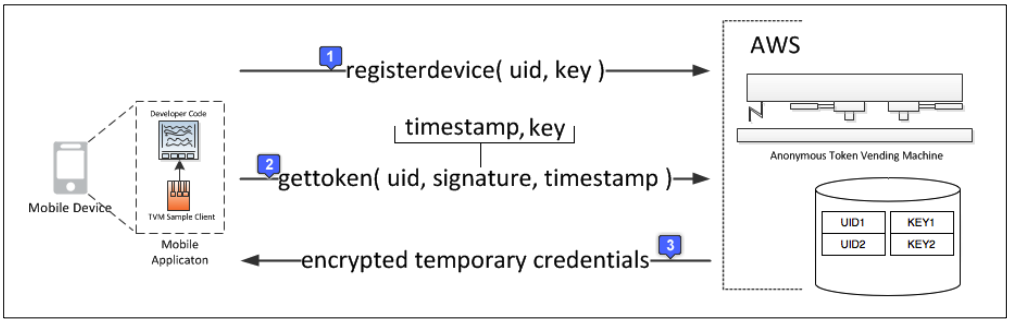


Fig. 2.400: Interacción entre un *TVM* anónimo y un dispositivo *iOS*[144]

[143] Amazon. (2012). *Autenticating Users of AWS Mobile Applications with a Token Vending Machine*. Recuperado el 29 de Abril de 2013, de Autenticating Users of AWS Mobile Applications with a Token Vending Machine: http://aws.amazon.com/articles/4611615499399490

[144] Amazon. (2012). *Autenticating Users of AWS Mobile Applications with a Token Vending Machine*. Recuperado el 29 de Abril de 2013, de Autenticating Users of AWS Mobile Applications with a Token Vending Machine.

2.16.3 Implementación de un Anonymous Token Vending Machine

Para mayor seguridad a nivel del servicio web *SimpleDB*, se propone implementar un *TVM* anónimo, el cual generará credenciales de seguridad temporales para los usuarios de una aplicación *iOS*, como se ha explicado en el sub apartado anterior.

Para esto, se debe descargar el comprimido necesario (*AnonymousTVM-1.3.0.zip*) desde el siguiente enlace: http://aws.amazon.com/code/8872061742402990. A continuación se debe crear la aplicación en la nube de *AWS* que funcionará como *TVM*. El servicio de *AWS* que permite gestionar e implementar aplicaciones en la nube se llama *AWS Elastic Beanstalk.*[145]

Para crear una nueva aplicación se debe iniciar sesión en *AWS* y dirigirse a la sección *Elastic Beanstalk* o seguir el siguiente enlace: https://console.aws.amazon.com/elasticbeanstalk. Presionar el botón *"Customize"* y a continuación presionar el botón "*Start*", de esta forma se presenta la ventana en donde se deben completar los detalles de la misma:

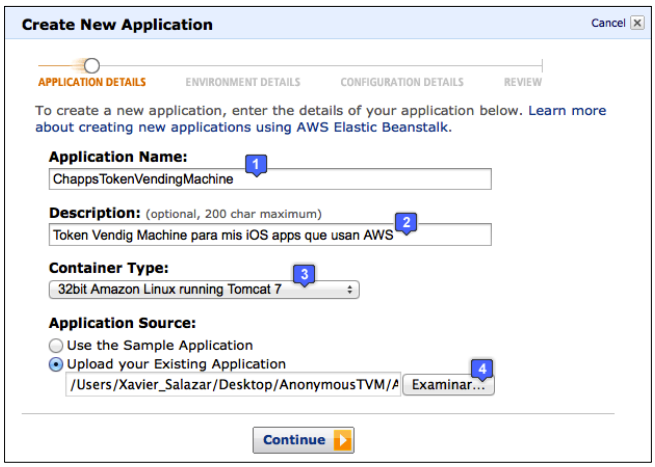


Fig. 2.401: Completar los detalles de la aplicación TVM[146]

[145] Amazon. (2012). *Token Vending Machine for Anonymous Registration*. Recuperado el 2 de Mayo de 2013, de Token Vending Machine for Anonymous Registration: http://aws.amazon.com/code/8872061742402990

[146] Amazon (2012). Elastic Beanstalk Recuperado el 3 de Mayo de 2013, de Elastic Beanstalk: https://console.aws.amazon.com/elasticbeanstalk

Ingresar el nombre de la aplicación (1) junto a la descripción de la misma (2). En el apartado tipo de contenedor (3) se debe seleccionar un contenedor *Tomcat*, ya que el *TVM* funciona únicamente en contenedores de este tipo. En el origen de la aplicación, seleccionar la opción *"Upload your existing application"*, presionar el botón *"Browse"* (4) y seleccionar el archivo *AnonymousTVM.war*, que se encuentra dentro del comprimido descargado anteriormente y finalizar presionando el botón *"Continue"*.

A continuación se debe ingresar los detalles del entorno en donde se ejecutará la aplicación. Así:

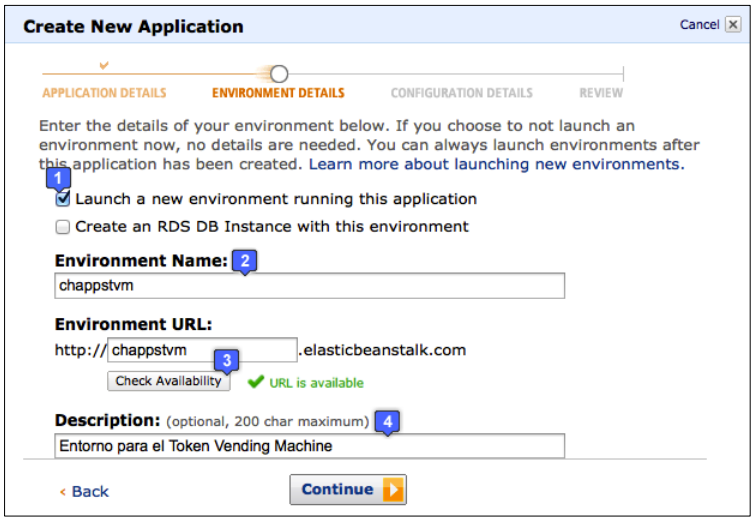


Fig. 2.402: Completar los detalles del entorno de la aplicación *TVM*[147]

Asegurarse de seleccionar únicamente la opción *"Launch a new Environment running this application"* (1), escribir un nombre para el entorno (2), la *URL* del entorno es automática, basada en el nombre del mismo y debe ser única, para verificar que la *URL* sea única, se debe presionar el botón *"Check Availability"* (3).[148] En el campo descripción, escribir una breve descripción del entorno y finalizar presionando el botón *"Continue"*.

---

147 Amazon (2012). Elastic Beanstalk Recuperado el 3 de Mayo de 2013, de Elastic Beanstalk: https://console.aws.amazon.com/elasticbeanstalk

148 Amazon. (2012). *Token Vending Machine for Anonymous Registration*. Recuperado el 2 de Mayo de 2013, de Token Vending Machine for Anonymous Registration: http://aws.amazon.com/code/8872061742402990

Acto seguido se deben ingresar los detalles de la configuración, así:

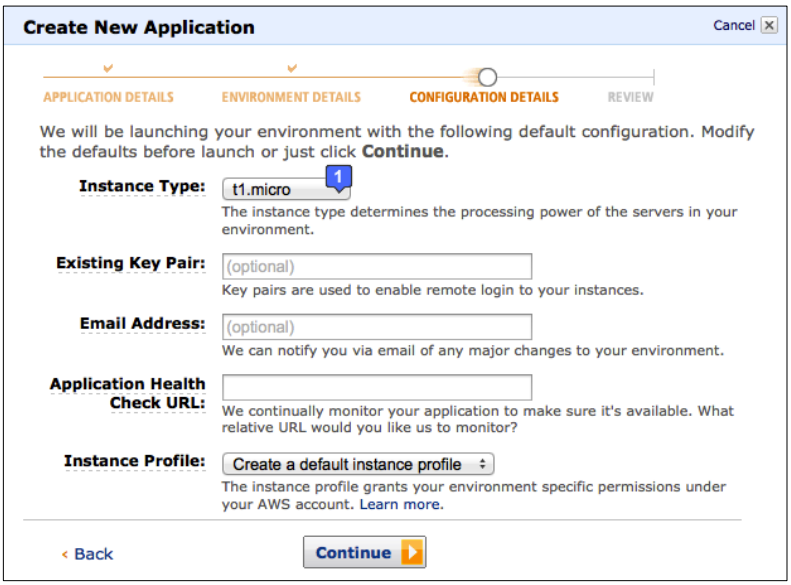


Fig. 2.403: Completar detalles de configuración de la aplicación *TVM*[149]

La configuración seleccionada es para el entorno de la aplicación. Se debe seleccionar *"t1.micro"* para el tipo de instancia (1), que determina el poder de procesamiento de los servidores. En realidad, cualquiera funcionaría, pero es el mejor y puede ser escalado como se necesite.[150] Dejar el segundo apartado vacío, ya que no es necesario para el *TVM*, a menos que se requiera accederlo remotamente. Escribir el correo electrónico personal en el tercer apartado para recibir notificaciones de los principales cambios en el entorno. Dejar el apartado *Application Health Check URL:* con el valor por defecto "/" y continuar presionando el botón *"Continue"*.

Finalmente revisar la información de la aplicación presentada en la siguiente ventana y continuar presionando el botón *"Finish"*.

A continuación *Elastic Beanstalk* ejecutará el *TVM*. Cuando se esté ejecutando, se lo debe configurar de la siguiente manera:

---

[149] Amazon (2012). Elastic Beanstalk Recuperado el 3 de Mayo de 2013, de Elastic Beanstalk: https://console.aws.amazon.com/elasticbeanstalk

[150] Amazon. (2012). *Token Vending Machine for Anonymous Registration*. Recuperado el 2 de Mayo de 2013, de Token Vending Machine for Anonymous Registration: http://aws.amazon.com/code/8872061742402990

En el menú "*Actions*", presionar "*Edit/Load Configuration*", como se indica a continuación:

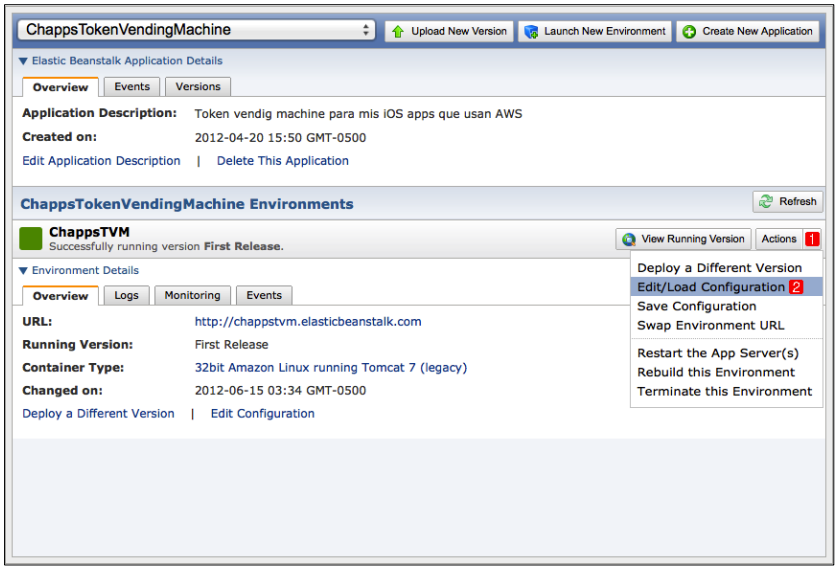


Fig. 2.404: Editar la configuración del *TVM* creado[151]

En la ventana presentada, seleccionar la pestaña *"Container"* e ingresar las credenciales de seguridad en los apartados correspondientes de la sección *"Environment Properties"*. Finalizar presionando el botón *"Apply Changes"*.

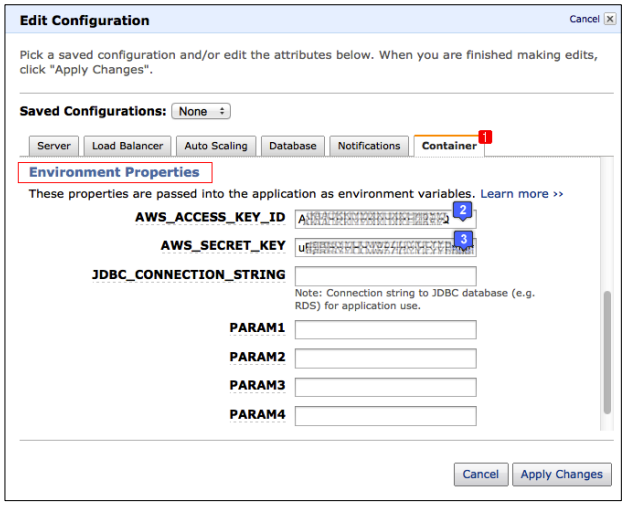


Fig. 2.405: Ingresar las credenciales de seguridad para el *TVM*[152]

[151] Amazon (2012). Elastic Beanstalk Recuperado el 3 de Mayo de 2013, de Elastic Beanstalk: https://console.aws.amazon.com/elasticbeanstalk

[152] Amazon (2012). Elastic Beanstalk Recuperado el 3 de Mayo de 2013, de Elastic Beanstalk.

2.16.4 Desarrollo de una aplicación *iOS* comunicada con un Amazon SimpleDB mediante un Token Vending Machine

Una vez implementada la aplicación que funcionará como un *TVM*, corresponde explicar como realizar la comunicación de una aplicación *iOS* con un *SimpleDB* utilizando el *TVM* para obtener las credenciales de seguridad temporales.

Se creará un nuevo proyecto denominado *TVMTest*, utilizando la plantilla *"Single View Application"*. La interface gráfica constará de dos botones, uno para almacenar y otro para mostrar un registro almacenado en el *SimpleDB*. Agregar al proyecto los *frameworks AWSRuntime* y *AWSSimpleDB* del *SDK* de *AWS* descargado anteriormente. De igual manera agregar el *framework Security* incluido en el *IDE*. Si se requiere ayuda, se puede referir al apartado *"2.8.1 Cómo añadir frameworks"*. El proyecto debería lucir similar al siguiente:

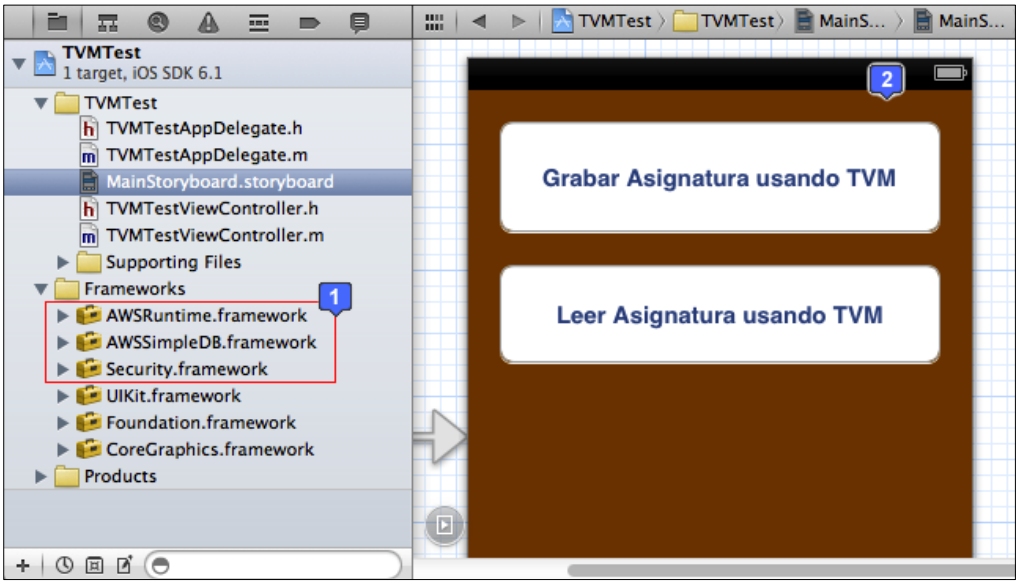


Fig. 2.406: Interface gráfica y *frameworks* de *TVMTest*

Asignar los eventos *grabarAsignatura* y *leerAsignatura* a los botones de la interface correspondientes. Si se requiere ayuda se puede referir al apartado *"2.5.2 Asignación de métodos a componentes de GUI"*. El fichero cabecera del *View Controller* quedaría como el siguiente:

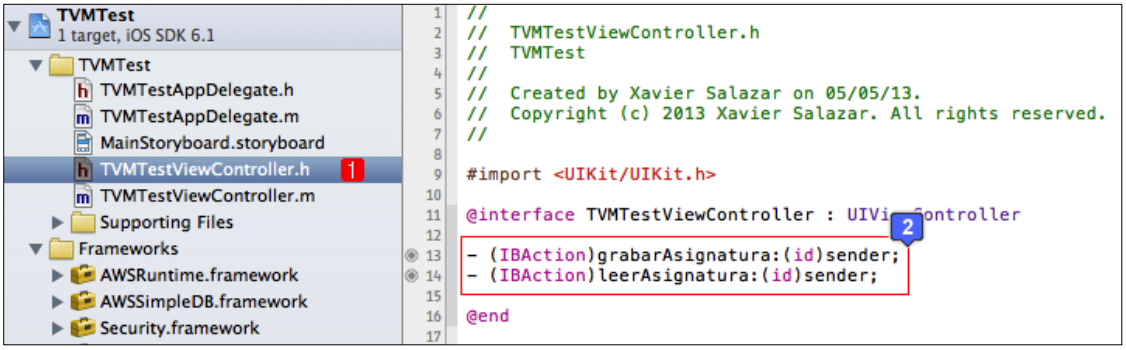


Fig. 2.407: Archivo cabecera del *View Controller* de *TVMTest*

La variable del cliente *SimpleDB*, en este caso será administrada por una clase llamada *Amazon Client Manager*, la cual será la encargada de obtener las credenciales de seguridad temporales mediante el *TVM*. Esta clase, junto con otras clases necesarias para el funcionamiento del *TVM*, se encuentran en uno de los ejemplos incluidos en el *SDK* de *Amazon* descargado anteriormente. Estas clases han sido extraídas y agrupadas en un solo comprimido *TVMUtilities.zip*, que puede ser encontrado en la carpeta de aplicaciones. Se debe proceder a descomprimir y agregar todas estas clases al proyecto. Se agruparán las clases en una carpeta *TVMClasses.*

Se debe también escribir el *URL* del *TVM* en el fichero implementación de la clase *Amazon Client Manager*, de forma de que hasta el momento el proyecto luciría como el presentado en la Fig. 2.408.

La mayoría de las clases del *TVM* no soportan *ARC*, lo que se debe indicar en los ajustes del proyecto, escribiendo la sentencia *"-fno-objc-arc"* como se había indicado en la Fig. 2.382. y se indica nuevamente en la Fig. 2.409.

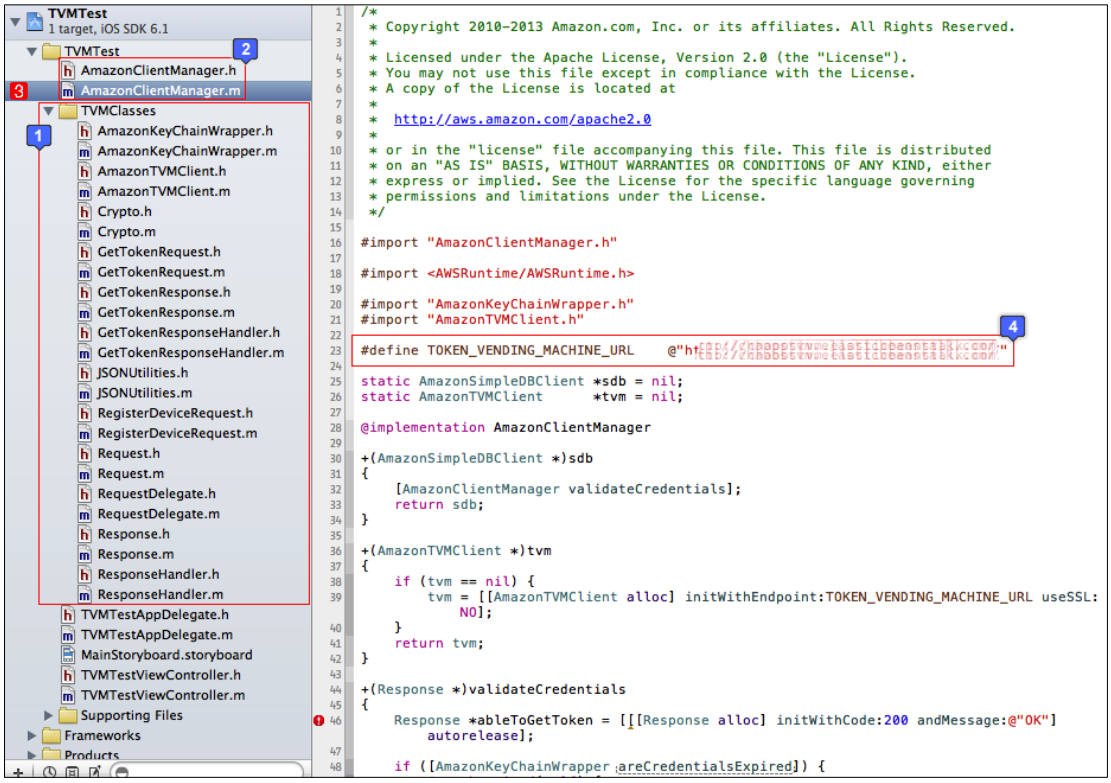


Fig. 2.408: Clases necesarias para comunicación con el *TVM*, *URL* del *TVM*

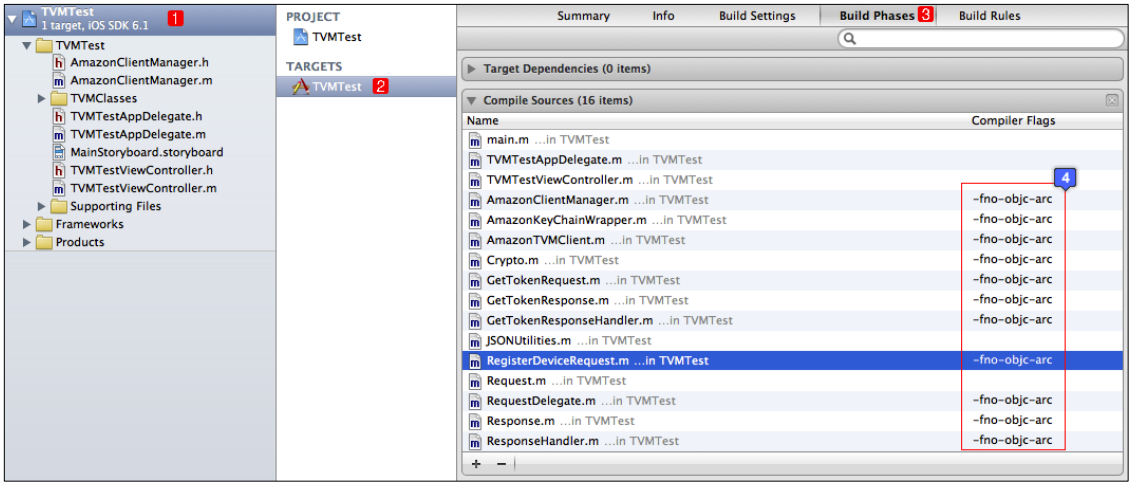


Fig. 2.409: Indicar las clases que no soportan *ARC*

En el fichero de implementación del *View Controller*, se debe importar inicialmente la clase *AmazonClientManager*, junto a la clase *AmazonSimpleDB*. Acto seguido definir constantes para el dominio (Asignaturas) y los atributos del mismo (Nombre, Profesor).

Implementar también los métodos que permitan grabar y leer los registros almacenados en el dominio Asignatura de la *SimpleDB*, como se indica en las siguientes figuras:

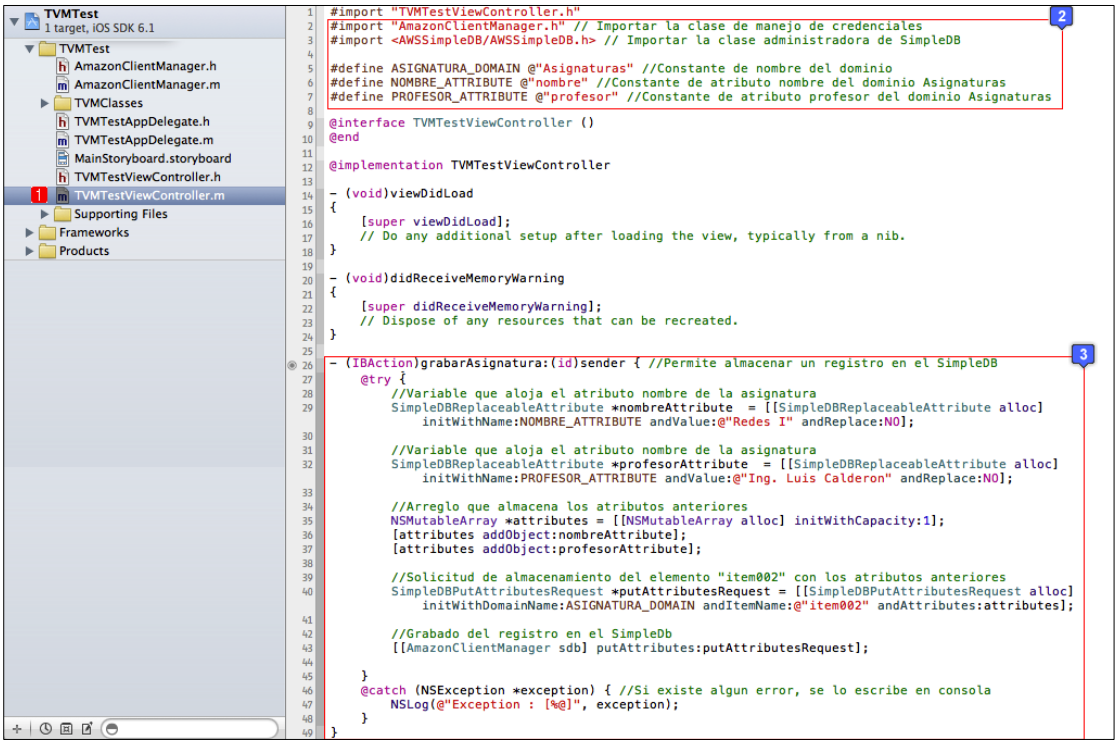


Fig. 2.410: Método que permite grabar un registro en la *SimpleDB* utilizando el *TVM*

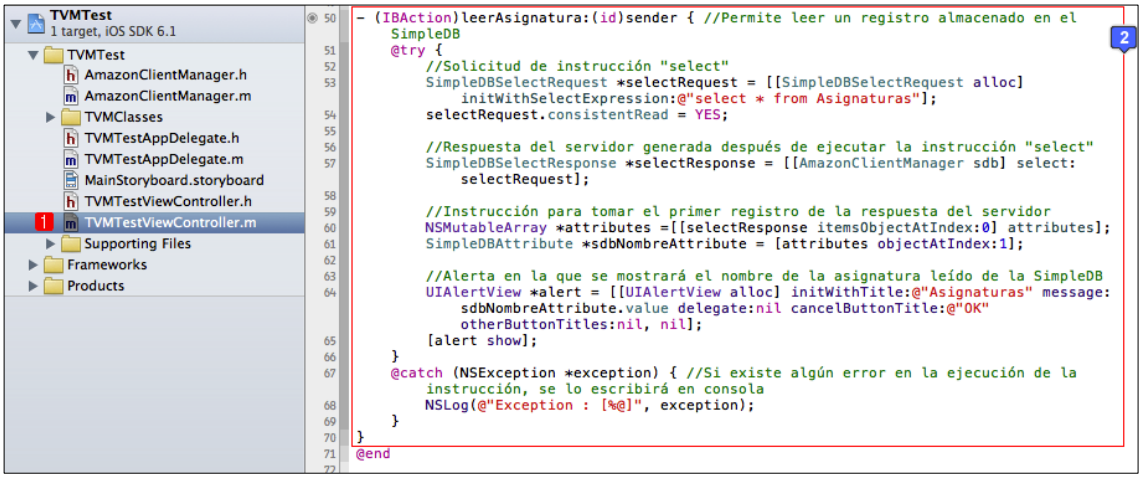


Fig. 2.411: Método que permite leer un registro de la *SimpleDB* utilizando el *TVM*

Al ejecutar la aplicación, mediante los botones correspondientes se puede almacenar y/o leer un registro asignatura en la *SimpleDB* utilizando el *TVM* implementado en el apartado anterior. Cuando se ejecute la aplicación, se debe presionar primero el botón

"Grabar asignatura usando el TVM" y después el botón *"Leer asignatura usando el TVM"* para presentar el registro almacenado en la *SimpleDB* en una ventana de alerta, como se indica en la Fig. 2.412.

Se recomienda implementar un *Activity View* que se muestre en los pocos segundos que dura la comunicación de la aplicación con el *TVM*, para que el usuario sepa que se está realizando alguna tarea.

Fig. 2.412: Aplicación *TVMTest* en ejecución

2.17 Cómo generar valor agregado a través de la aplicación

Se puede generar ganancias mediante la distribución de aplicaciones publicadas sin precio de venta en la tienda *AppStore*, es decir, mediante la distribución de aplicaciones que pueden ser descargadas de forma gratuita.

La forma de generar ganancias mediante aplicaciones gratuitas es presentando publicidad de *Apple* o *iAds* en las mismas. Es por esto que la mayoría de las aplicaciones que se encuentran gratuitas en la tienda *AppStore* presentan publicidad.

Con solo unas pocas líneas de código, se puede presentar publicidad en una aplicación, y; por lo tanto, recibir un porcentaje de los ingresos generados por la publicidad *iAd* que se haya mostrado en la misma.

En los siguientes sub apartados se explicará como implementar un banner de publicidad en una aplicación *iOS* y se mencionarán algunas de las mejores practicas que se deben considerar al publicar aplicaciones con publicidad en la tienda *AppStore*.

Cabe mencionar que para poder distribuir aplicaciones con publicidad, se debe aprobar un contrato adicional en el portal *iTunes Connect*, Lamentablemente, al utilizar un espacio considerable en la pantalla del dispositivo *iOS*, la publicidad puede resultar molestosa para los usuarios, y/o ralentizar la carga de contenidos de la misma; es por esto que cuando una aplicación se presenta en una versión gratuita y una de pago, generalmente, la versión de pago indica en su descripción que se encuentra libre de publicidad.

2.17.1 Implementación de un banner publicitario *iAd* en una aplicación

Para indicar como implementar un banner *iAd* en una aplicación *iOS*, se creará un nuevo proyecto utilizando la plantilla *"Single View Application"* y se lo nombrará *iAdTest.* La interface gráfica de usuario constará únicamente de una vista (*UIView*) de color gris. Además se debe agregar al proyecto el *framework iAd.framework.* Si se requiere ayuda se puede referir al apartado *"2.8.1 Cómo añadir frameworks"*. Hasta este punto, el proyecto luciría como el siguiente:

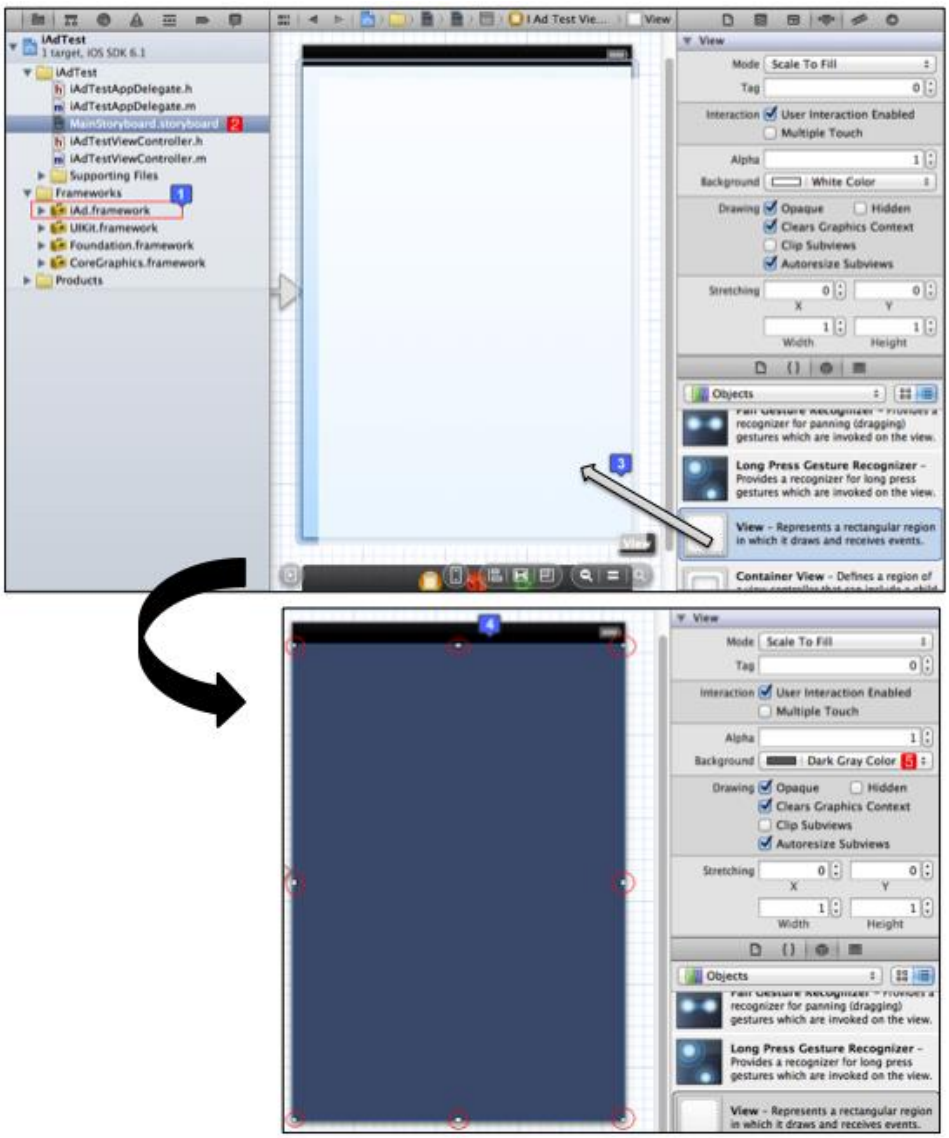


Fig. 2.413: Interface gráfica y archivos del proyecto *iAdTest*

A continuación, se debe asignar la variable *contentView* a la vista arrastrada sobre el *View Controller* en el paso anterior, si se requiere ayuda se puede referir al apartado *"2.5.1 Asignación de variables a componentes de GUI"*.

En el fichero cabecera del *View Controller*, se debe importar la clase *iAd*, y crear una variable de tipo *AdBannerView*. Se debe declarar adherencia a protocolos de delegado de *AdBannerView* para poder manejar eventos exclusivos del mismo dentro del archivo de implementación. A continuación se presenta el fichero cabecera del *View Controller:*

Fig. 2.414: Fichero cabecera del *View Controller* de *iAdTest*

En el fichero de implementación del *View Controller*, se debe empezar por realizar el *synthesize* de la variable *contentView*. Se implementará un método que permita dibujar el banner de manera animada dentro de la vista y que permita posicionarlo y dimensionarlo automáticamente de acuerdo a la orientación del dispositivo *iOS*. Este método se llamará *layoutAnimated:(BOOL)animated* y se lo puede encontrar en los ejemplos de proyectos que implementan *iAds* que ofrece *Apple*[153]. Además, es compatible con *iOS* 6 e *iOS* 5.

[153] Apple. (2012). *iAd Suite*. Recuperado el 5 de Mayo de 2013, de iAd Suite: https://developer.apple.com/library/ios/#samplecode/iAdSuite/Introduction/Intro.html#//apple_ref/doc/uid/DTS40010198-Intro-DontLinkElementID_2

Se debe ejecutar este método cada vez que ocurra el evento *viewDidAppear*, de forma de ajustar el banner automáticamente de acuerdo a la orientación de la vista o dispositivo *iOS*.

En el evento *viewDidLoad* del fichero, se debe inicializar la vista del banner, indicar cual será su delegado y agregarlo a la vista. De igual manera, se recomienda que la forma de inicialización del banner sea compatible con las distintas versiones de *iOS*. A continuación se presenta el contenido del archivo de implementación del *View Controller* hasta el momento:

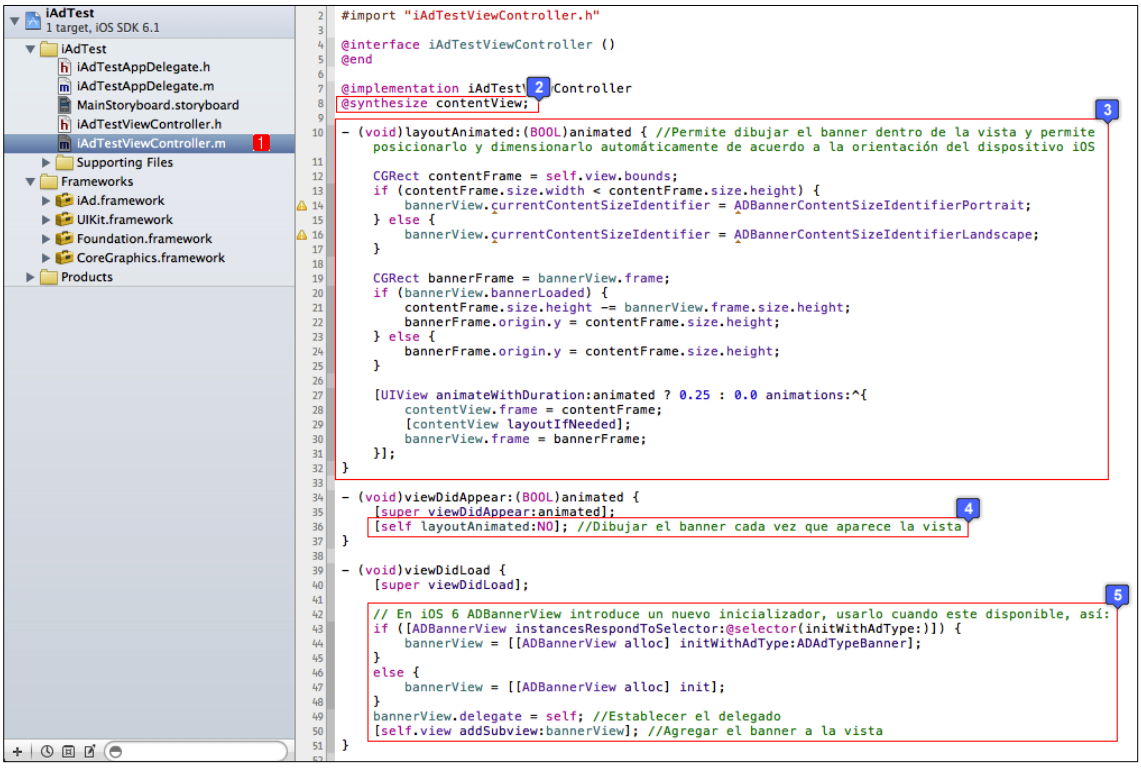


Fig. 2.415: Fichero de implementación del *View Controller* de *iAdTest*

Adicionalmente, se deben implementar algunos métodos de delegado del objeto *ADBannerView* en el fichero de implementación del *View Controller*, después de los métodos de la Fig. 2.415, ya que se ha configurado a este fichero como delegado del *AdBannerView*.

Éstos métodos serán los encargados de dibujar el banner publicitario de acuerdo a algunos eventos que pudieran ocurrir. Se lo debe realizar de esta forma

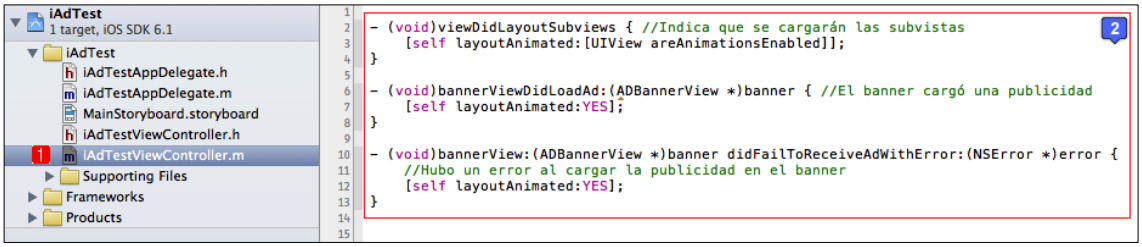


Fig. 2.416: Implementar eventos de delegado del *AdBannerView*

Se debe escribir de igual manera en este archivo de implementación el método necesario para soportar la rotación de las interfaces gráficas, como se ha indicado en el sub apartado *"2.4.4 Rotación de interfaces y sus componentes"*. En este ejemplo se escribirá un método que permita esta característica para *iOS* 5 y otro que lo permita para *iOS* 6, de forma de escribir una aplicación compatible con las dos versiones de *iOS*. Así:

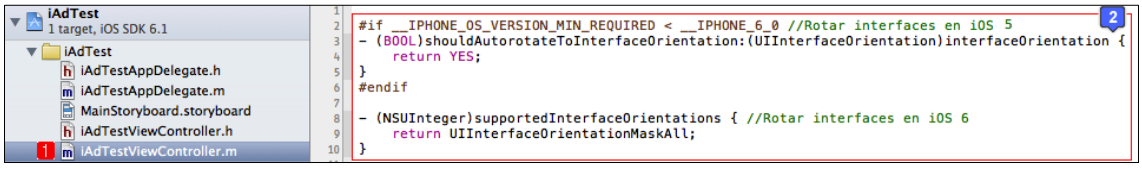


Fig. 2.417: Implementar métodos para rotar interfaces de usuario en *iOS 5* e *iOS 6*

Finalmente se puede ejecutar la aplicación para verificar su correcto funcionamiento; pero, como se puede observar en la Fig. 2.415, se presentan unas advertencias de uso de métodos que han sido deprecados con el lanzamiento de *iOS 6*, los cuales se deben seguir utilizando si se desea que la aplicación sea compatible con sistemas operativos *iOS* anteriores. Estas advertencias podrían llegar a ser molestosas para el desarrollador, por lo que para eliminarlas, se puede escribir la instrucción *"-Wno-deprecated"* en el apartado *Compiler Flags* correspondiente a la clase *iAdTestViewController* en las configuraciones del proyecto.

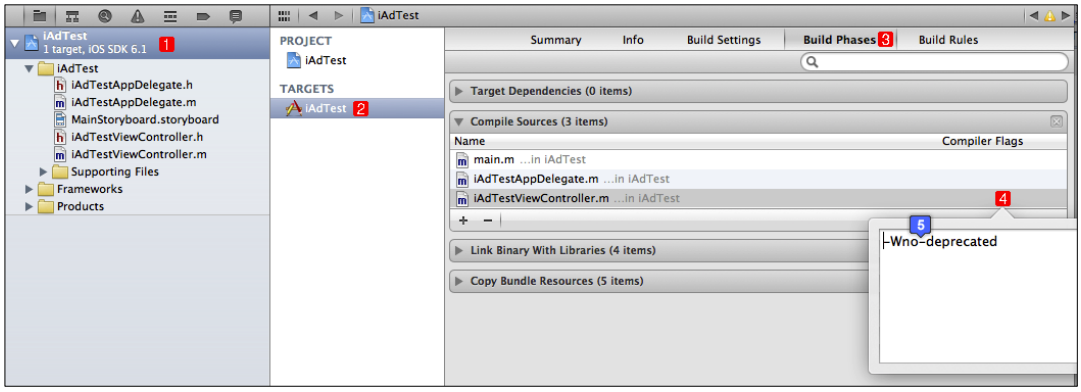


Fig. 2.418: Instrucción para evitar las advertencias de uso de métodos deprecados

De esta forma al compilar la aplicación previo a la ejecución de la misma, ya no se presentarán estas advertencias. A continuación se presenta la aplicación en ejecución:

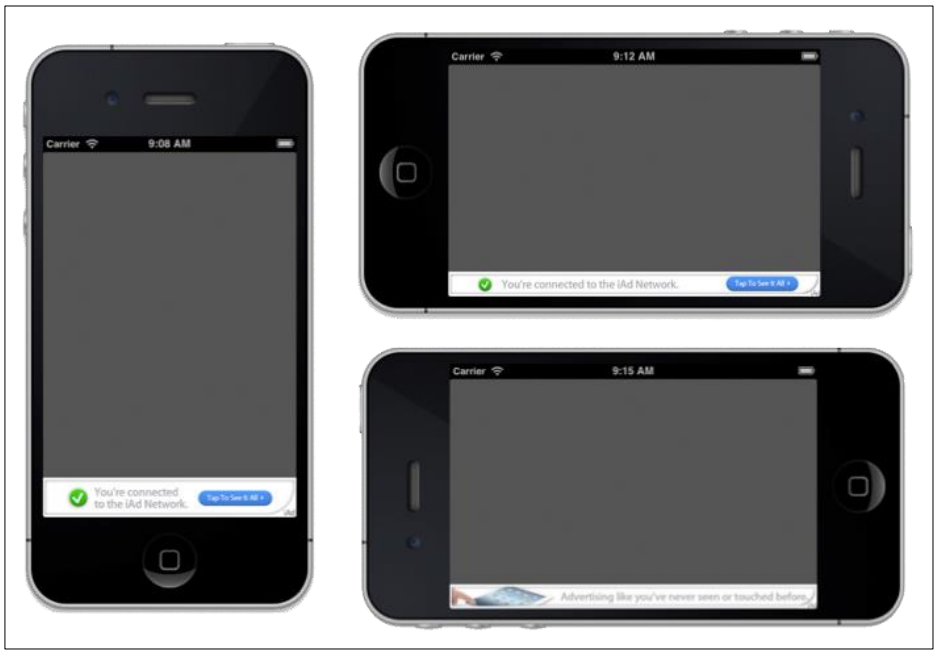


Fig. 2.419: Aplicación *iAdTest* en ejecución

Como se puede observar la aplicación presenta la publicidad adecuadamente de acuerdo a la orientación del dispositivo *iOS*. Cuando el usuario presiona el banner publicitario, se muestra una vista de publicidad que se sobrepone a la aplicación; por esto, es importante que se suspenda cualquier tipo de proceso que se esté realizando, debido a que el usuario no interactuará en estos momentos con la aplicación sino con la vista de

la publicidad. Para esto existen los siguientes métodos de delegado, que deberían ser implementados en la clase *iAdTestViewController:*

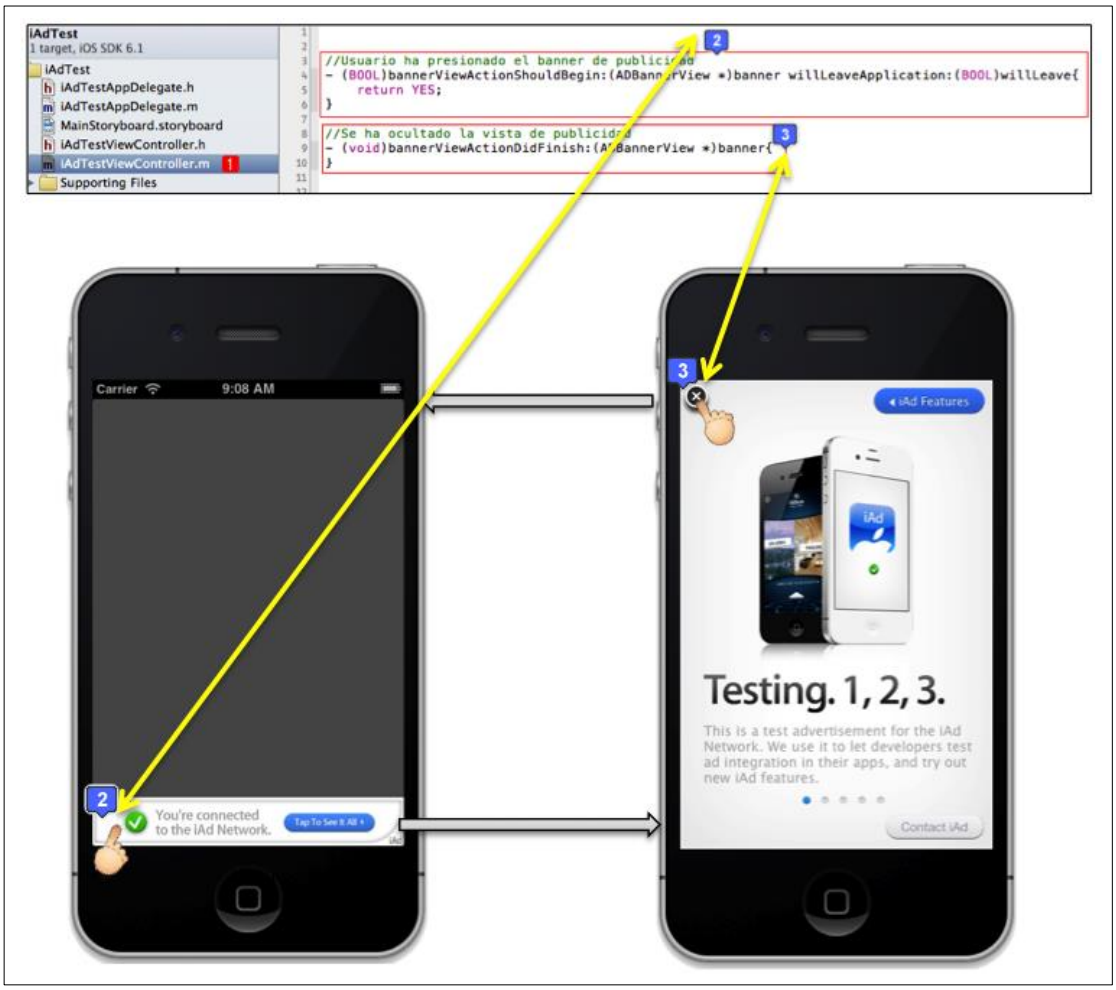


Fig. 2.420: Métodos para manejar eventos de presentación y ocultamiento de vista publicitaria

2.17.2 Mejores practicas al implementar publicidad en aplicaciones

Al integrar banners publicitarios en aplicaciones para generar una remuneración adicional, *Apple* recomienda seguir las siguientes mejores prácticas, de forma de maximizar la oportunidad de generar ingresos y asegurar una mejor experiencia de usuario con la aplicación:

Acordar el contrato de la red *iAd* y habilitar la aplicación para mostrar publicidad *iAd* en el portal *iTunes Connect*[154]: Para esto, se debe dirigir al portal de *iTunes Connect*, revisar y completar el formulario respectivo de forma de acordar el contrato de red *iAd* localizado en el módulo *"Contracts, Tax and Banking Information"*. Una vez acordado el contrato, se lo podrá encontrar junto con los otros contratos vigentes, como se indica a continuación:

Fig. 2.421: Contrato de red *iAd* acordado junto a los otros contratos vigentes[155]

[154] Apple. (2012). *iAd Implementation Best Practices*. Recuperado el 13 de Mayo de 2013, de iAd Implementation Best Practices: http://developer.apple.com/library/ios/#technotes/tn2264/_index.html

[155] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2012, de Contracts, Tax and Banking: https://itunesconnect.apple.com

A continuación se debe habilitar la publicidad de la red *iAd* en la aplicación, en el modulo *"Manage Your Applications"*, antes de enviar la aplicación para revisión y posterior aprobación para comercialización en *AppStore*, como se indica a continuación:

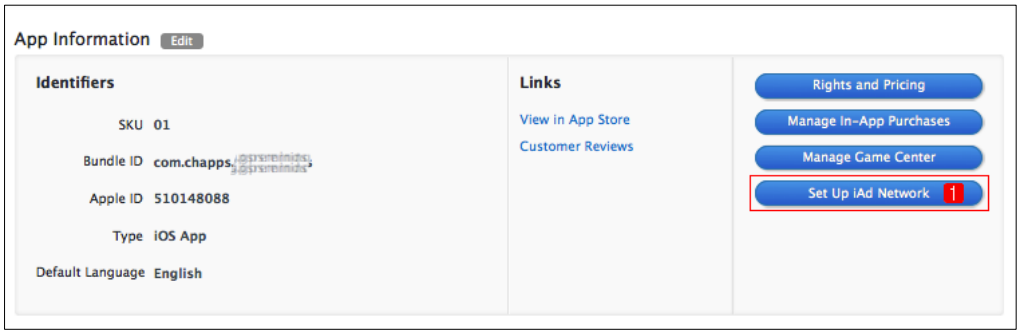


Fig. 2.422: Configurar red *iAd* en una aplicación en *iTunes Connect*[156]

Soportar los últimos requerimientos de *iOS*: La publicidad de la red *iAd*, puede ser ejecutada únicamente en aplicaciones compatibles con *iOS* 4 o posterior. Se recomienda que la aplicación sea compatible con el último *iOS* disponible.

Crear aplicaciones universales: Para asegurar que la aplicación puede presentar los banners publicitarios a todos los usuarios, se debe asegurar de desarrollar una aplicación universal compatible con todos los tipos de dispositivos *iOS*.

Presentar sólo un banner publicitario por vista: Asegurarse de presentar máximo un banner por cada pantalla de la aplicación para maximizar su rendimiento, si se presenta mas de un banner en una misma ventana, se rechazará la aplicación en el proceso de revisión de la misma.

Manejar los cambios de orientación del banner dinámicamente: Se recomienda consultar el ejemplo desarrollado en el sub apartado anterior, en donde se manejan los

[156] Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2012, de Manage Your Apps: https://itunesconnect.apple.com

cambios de orientación del banner de forma dinámica: se lo posiciona y redimensiona dinámicamente de acuerdo a la orientación del dispositivo *iOS*.

Ocultar los banners en blanco o vacíos: Si se produce un error en la carga del banner publicitario, por ejemplo debido a la inexistencia de una conexión de red, se lo debe ocultar.

Compartir instancias de banner entre vistas: Si una aplicación es por ejemplo basada en pestañas, se recomienda generar un único banner para presentar en la aplicación. De forma de mejorar el rendimiento de la misma.

Mostrar los banners *iAd* por lo menos durante treinta segundos: Si la aplicación utiliza un temporizador para alternar entre publicidades presentadas en el banner, se recomienda presentar cada publicidad durante al menos treinta segundos, de modo de optimizar el rendimiento de publicidad en la aplicación.

Si se siguen estas mejores practicas, es garantizado que se maximizará la oportunidad de generar ingresos por presentar publicidad en la aplicación. Se asegurará, de igual manera, una mejor experiencia de usuario con la aplicación.[157]

[157] Apple. (2012). *iAd Implementation Best Practices*. Recuperado el 13 de Mayo de 2013, de iAd Implementation Best Practices: http://developer.apple.com/library/ios/#technotes/tn2264/_index.html

Conclusiones

Al finalizar el desarrollo del tutorial, se puede afirmar que se han presentado los temas principales para el desarrollo y distribución de aplicaciones para dispositivos *iOS;* empezando por los temas más básicos de manera de que el lector pueda empezar desde cero con el desarrollo de estas aplicaciones.

Se han creado aplicaciones demostrativas para la mayoría de los temas presentados en el tutorial, de manera de que el lector pueda comprender de mejor manera cada uno de ellos.

Además de los temas tratados en los que se ha explicado como crear aplicaciones con distintos fines, se han realizado distintos análisis que permitirán a los lectores comprender por qué se realizan las cosas y no solamente seguir los pasos de cada sub apartado del tutorial.

Se puede afirmar que se realizó un tutorial de calidad debido a que se trataron temas adicionales como la seguridad dentro de una aplicación *iOS*, así como el proceso de distribución de aplicaciones, que para muchos usuarios, representa una dificultad debido a lo complicado que puede resultar publicar una aplicación para su comercialización en la tienda *AppStore*.

Se desarrollaron también temas como sugerencias para desarrollar aplicaciones de calidad, como por ejemplo la implementación de una clase que permita calificar directamente a la aplicación en la tienda *AppStore*.

De esta forma se espera que después de haber leído el tutorial, el lector se encuentre en la capacidad de desarrollar aplicaciones *iOS* estéticamente atractivas, funcionales, seguras y de calidad.

Capítulo 3: Desarrollo de una aplicación iOS de recordatorios por GPS mediante el tutorial

En el presente capítulo, se desarrollará una aplicación *iOS* cuya funcionalidad será la de presentar recordatorios o notificaciones de acuerdo a la posición *GPS* del usuario. Se utilizará el tutorial del capitulo anterior para implementar la aplicación. Se indicará el apartado al cual se hará referencia a medida que se desarrolle la aplicación.

Para el desarrollo de esta aplicación, se presentará inicialmente el análisis y diseño de la misma, en donde se la describirá brevemente y se explicará su ámbito y funcionalidad. Dentro de este análisis y diseño, se realizará también el levantamiento de requisitos de la aplicación, en donde se desarrollará un diagrama de casos de uso y se describirá cada uno de los mismos. Adicionalmente se presentará varios diagramas de funcionalidad, el diseño de base de datos y diseño de interface de la aplicación.

Acto seguido, se implementará la aplicación haciendo referencia a algunos de los apartados y sub apartados del tutorial desarrollado en el capítulo anterior.

Una vez desarrollada la aplicación, se realizarán los casos de prueba que se consideren pertinentes antes de enviarla para revisión y futura comercialización en la tienda *AppStore*, haciendo referencia a los respectivos apartados del tutorial del capítulo anterior.

Se implementará en la aplicación las sugerencias del tutorial, como la implementación de publicidad *iAd* para generar ingresos adicionales, la motivación a los usuarios para calificar a la aplicación mediante el uso de la clase *AppiRater* y la generación de una versión gratuita con funcionalidades limitadas.

Se concluirá el capítulo presentando estadísticas de descargas y ventas de la aplicación.

3.1 Análisis y diseño de la aplicación

En el presente apartado se realizará el análisis y diseño de la aplicación de recordatorios por *GPS* para *iOS* que se desarrollará. Para esto, se realizará inicialmente la descripción general de la aplicación *iOS* junto a la funcionalidad y el ámbito de la misma.

Se presentarán después los requisitos específicos de la aplicación, así como la descripción de sus actores y diagrama de casos de uso. Se complementará el diagrama con la descripción de casos de uso.

Adicionalmente se presentará el diagrama de secuencias de la aplicación para ejemplificar algunas de las secuencias como mantenimiento y reporte de lugares, mantenimiento y reporte de tareas, realización de una tarea, contacto con el desarrollador, configuración de la aplicación y compartir en redes sociales.

Se desarrollará también el diagrama de actividades de la aplicación para comprender de mejor manera el funcionamiento de la misma.

El diagrama de base de datos se implementará utilizando el *framework Core Data* dentro del mismo *IDE XCode;* para el diseño de interface se presentará capturas de pantalla de la aplicación en ejecución y el *Storyboard* de la aplicación.

3.1.1 Funcionalidad y ámbito de la aplicación

Funciones del Sistema: Esta aplicación *iOS* permitirá a los usuarios de un dispositivo *iOS* ser notificados sobre tareas por realizar, en el momento en el que lleguen a una determinada distancia del lugar especificado para la misma. Tanto la distancia, como el lugar de recordatorio son programados por el usuario en la misma aplicación.

Ámbito de la aplicación iOS:

- Mantenimientos de ingreso, eliminación y modificación de datos de lugares.
- Mantenimiento de ingreso, eliminación y modificación de datos de tareas a realizar (relacionadas con el punto anterior).
- Reporte de tareas a realizar filtradas por el lugar en donde se encuentre el usuario al momento de ejecutar la aplicación.
- Capacidad de modificación de varios parámetros de la aplicación incluyendo:
  - Idioma: Inglés y Español
  - Distancia del lugar para presentar el recordatorio
  - Privacidad en la notificación del recordatorio
  - Selección de unidades del parámetro distancia (Km / Millas)
  - Capacidad de indicar el uso de la aplicación en las redes sociales *Facebook* y *Twitter*.

3.1.2 Levantamiento de requisitos

Requisitos específicos:

Descripción de actores:

Usuario: Es el que maneja prácticamente todas las funciones del sistema. Una vez que disponga de la aplicación en su dispositivo *iOS*, el usuario podrá realizar las siguientes acciones detalladas en los siguientes casos de uso:

Diagrama de Casos de Uso:

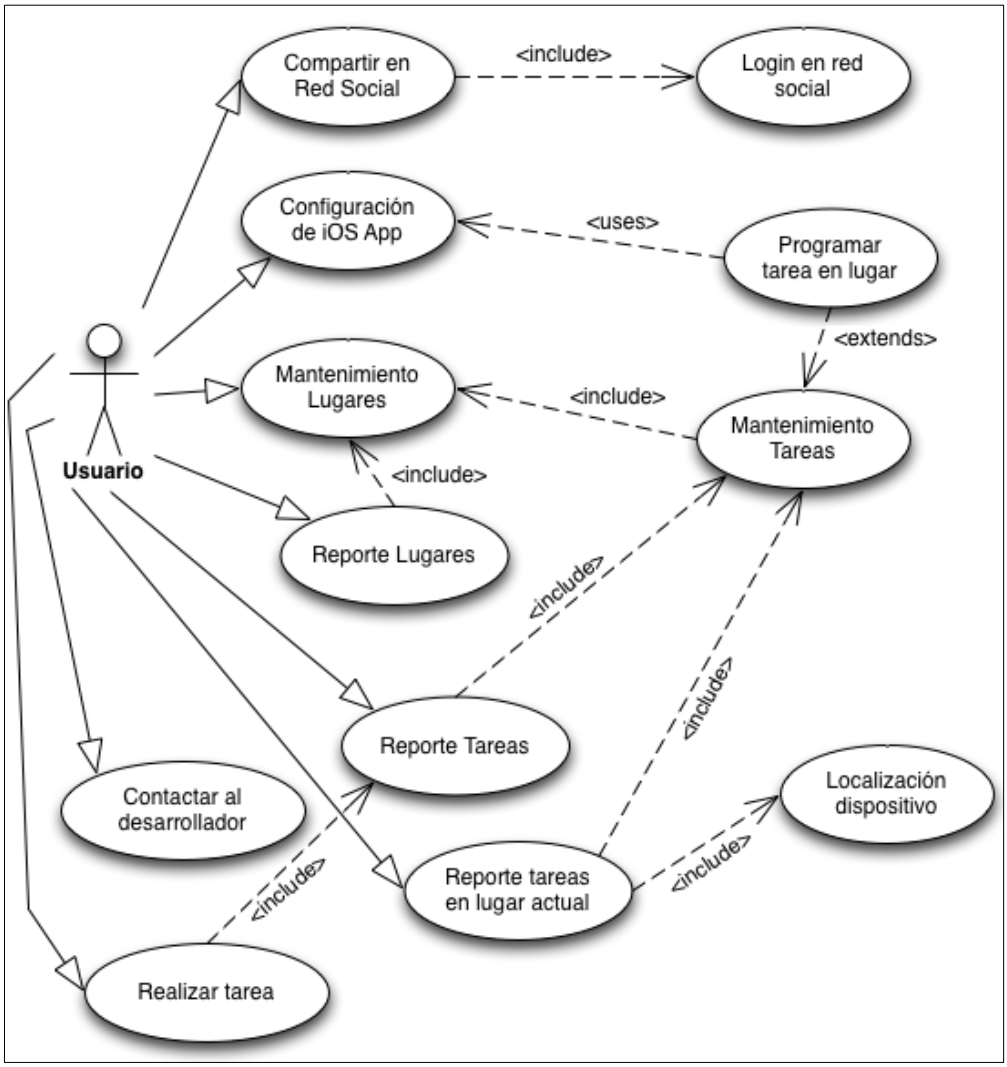


Fig. 3.1: Diagrama de casos de uso de la aplicación *iOS* de recordatorios por *GPS*

Descripción de casos de uso:

Caso de uso 1	Compartir en Redes Sociales / Login en red social
Actor:	Usuario
***Descripción*:**	Permitirá indicar al usuario mediante las distintas redes sociales (*Facebook, Twitter*) que se encuentra utilizando la aplicación *iOS* (*GPSReminds*)
REQUISITOS ASOCIADOS	
R.1.1: En la vista de ajustes, el sistema **presentará** botones para compartir tanto en *Facebook* como en *Twitter*, el usuario decide en que red social publicará, o lo puede realizar en las dos si lo desea. R.1.2: El sistema **solicitará el log in** del usuario en la red social deseada y luego presentará una ventana: R.1.2.1: En el caso de *Facebook*, luego de autenticado y autorizado a la aplicación, se **presentará** un botón para compartir, junto a la foto de perfil y nombre del usuario para confirmar la publicación. R.1.2.2: En el caso de *Twitter*, se **utilizará la característica de iOS 5** que permite compartir fácilmente en ésta red social presentando una ventana con el contenido del tweet que se enviará, el usuario presionará un botón que confirma el envío del *tweet*.	

Caso de uso 2	Configurar Aplicación iOS
Actor:	Usuario
***Descripción*:**	Permitirá al usuario configurar los parámetros por defecto de forma de presentar los recordatorios de la aplicación iOS.
REQUISITOS ASOCIADOS	
R.2.1: En la sección de configuraciones, el sistema **presentará** cada uno de los parámetros de la aplicación R.2.2: Para el caso del **idioma**, se podrá seleccionar entre inglés y español. Se seleccionará el idioma a partir de un listado de idiomas, al momento de seleccionarlo, automáticamente toda la aplicación cambiará al idioma seleccionado. R.2.3 En el caso de la **distancia del recordatorio**, se ingresará un valor numérico para indicar la distancia ya sea en kilómetros o millas, que indicará la distancia hasta el lugar en donde se desea que se presente el recordatorio. R.2.4 En el caso de las **unidades**, se seleccionará mediante un componente *"Control"* si se desea utilizar como unidad de medida: kilómetros o millas. R.2.5 En el caso de la **privacidad**, se seleccionará mediante un componente *"Switch"* si se desea que al presentar la notificación se especifique la tarea o no. (En caso de que el propietario del dispositivo no lo disponga al momento de presentarse la notificación)	

Caso de uso 3	Mantenimiento de Lugares
Actor:	Usuario
***Descripción*:**	Permitirá al usuario ingresar, modificar y eliminar lugares, tomando en cuenta la integridad referencial por si se ha asignado una tarea a un determinado lugar.
REQUISITOS ASOCIADOS	
R.3.1: En la sección de lugares, se **presentará un botón (+)**, el cual al ser presionado permitirá añadir un nuevo lugar. R.3.2: Para añadir un lugar **se presentará un mapa** en donde si se activa el *switch* de localización, se mostrará al usuario su localización actual en el mapa representada por un punto azul. R.3.3: Para seleccionar un lugar, se **deberá mantener presionado el mapa durante al menos un segundo** en el lugar deseado para colocar un *pin* que indicará la localización del lugar. R.3.4 Acto seguido, o previo a esté, se **deberá obligatoriamente nombrar al lugar** en un campo de texto. R.3.5 Para **grabar el lugar se deberá presionar el botón** de color rojo: "grabar".	

Caso de uso 4	Mantenimiento de Tareas
Actor:	Usuario
***Descripción*:**	Permitirá al usuario ingresar, modificar y eliminar tareas, tomando en cuenta la integridad referencial.
REQUISITOS ASOCIADOS	
R.4.1: En la sección de tareas, se **presentará un botón (+)**, el cual al ser presionado permitirá añadir una nueva tarea. R.4.2: Se **presentará una ventana con un campo de texto** en donde se escribirá el título de la tarea. R.4.3: Opcionalmente **se podrá agregar notas adicionales** para la tarea en otro campo de texto. R.4.4: A partir de un **"Picker View" se seleccionará el lugar** en donde se desea ser recordado de ésta tarea R.4.5 Para **grabar la tarea se deberá presionar el botón** de color rojo: "grabar".	

Caso de uso 5	Reporte de Lugares
Actor:	Usuario
***Descripción*:**	Permitirá al usuario visualizar cada uno de los lugares existentes en donde podrá modificar o eliminar a los mismos.
REQUISITOS ASOCIADOS	
R.5.1: En la vista de lugares se **presentará un listado de lugares**, junto a cada uno de los cuales se presentará un botón de color azul, el cual al ser presionado permitirá modificar el lugar. R.5.2: Si se **desea eliminar un lugar previamente agregado, existirá un botón "editar"** el cual al ser presionado permitirá eliminar cada uno de los lugares no deseados. De igual manera, **se podrá eliminar un lugar específico realizando un desliz con el dedo de izquierda a derecha o inversamente** sobre el mismo, acto seguido se presentará el botón para eliminarlo. R.5.3: Con el objetivo de mantener la integridad referencial, **al eliminar un lugar relacionado con una tarea, se eliminarán también todas la tareas relacionadas al mismo**, es decir se eliminará en cascada.	

Caso de uso 6	Reporte de Tareas / Realizar tarea
Actor:	Usuario
***Descripción*:**	Permitirá al usuario visualizar cada una de las tareas agregadas en donde se las podrá modificar y eliminar. Se podrá también marcar una tarea como realizada.
REQUISITOS ASOCIADOS	
R.6.1: En la sección de tareas, se **presentará un listado de tareas**, junto a cada una de las cuales se presentará un botón de color azul, el cual al ser presionado permitirá modificarla. R.6.2: Si se desea **marcar una tarea como realizada, se la deberá presionar.** R.6.3 Si se **desea eliminar una tarea previamente almacenada, existirá un botón "editar"** el cual al ser presionado permitirá eliminar cada una de las tareas no deseadas. De igual manera, **se podrá eliminar una tarea específica realizando un desliz con el dedo de izquierda a derecha o inversamente** sobre la misma, acto seguido se presentará el botón para eliminarla.	

Caso de uso 7	Contactar al desarrollador
Actor:	Usuario
***Descripción*:**	Permitirá al usuario contactar al desarrollador vía correo electrónico
REQUISITOS ASOCIADOS	
R.7.1: En la sección de configuraciones, el la sección "**acerca de la aplicación", se presentará un botón** que indique que al presionarlo se puede contactar al desarrollador. R.7.2: Al presionar el botón anterior, se **abrirá la aplicación nativa "mail" del dispositivo iOS** en donde la dirección del destinatario será la dirección de correo electrónico del desarrollador de esta aplicación.	

Caso de uso 8	Programar tarea en lugar / Realizar Tarea / Localización dispositivo
Actor:	Sistema
***Descripción*:**	Permitirá al usuario visualizar todas las tareas que tenga pendientes en el lugar en donde se encuentra actualmente
REQUISITOS ASOCIADOS	
R.8.1: En la vista "Aquí", **se presentará el lugar actual en donde se encuentre el usuario junto a la distancia hasta el mismo**, si es que esta distancia es menor a la configurada en la sección de parámetros. Para esto, se utilizará la localización actual del dispositivo iOS. R.8.2: Junto al lugar, **se presentará todas las tareas en caso de existir alguna** pendiente o ya realizada en el mismo. R.8.3: Se **podrá marcar una tarea como realizada al presionarla, o en su defecto, visualizar sus notas adicionales si se presiona el botón azul** junto a cada una de las mismas.	

3.1.3 Diagramas de funcionalidad de la aplicación

Diagrama de Secuencias: Mantenimiento y reporte de lugares

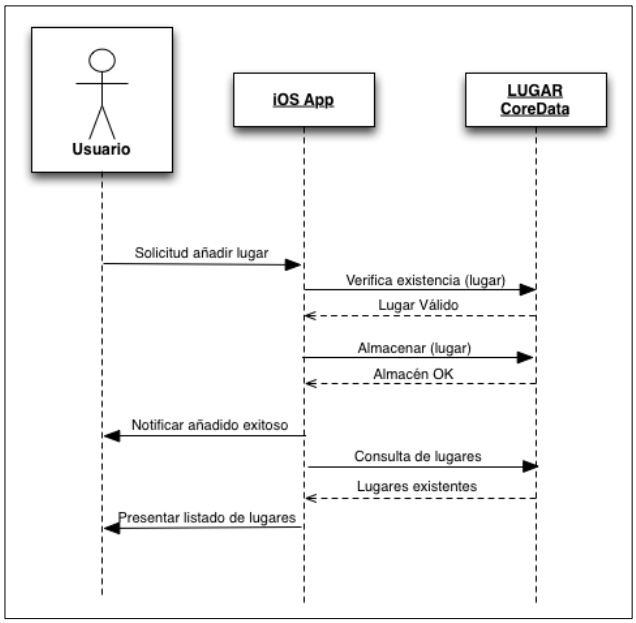


Fig. 3.2: Diagrama de Secuencias - Mantenimiento y reporte de lugares

Diagrama de Secuencias: Mantenimiento y reporte de tareas

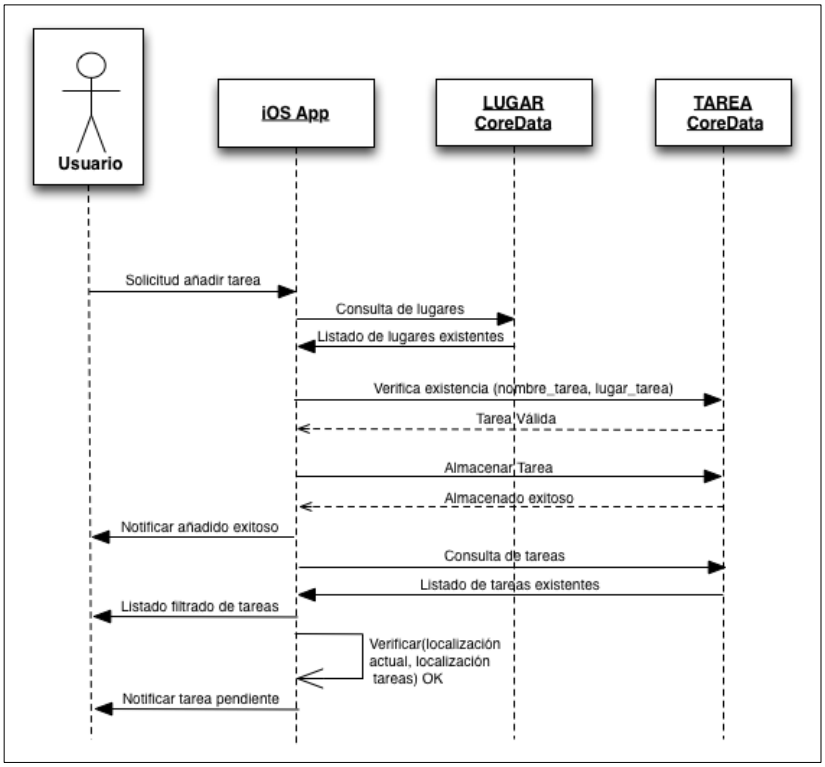


Fig. 3.3: Diagrama de Secuencias - Mantenimiento y reporte de tareas

Diagrama de Secuencias: Realizar una tarea

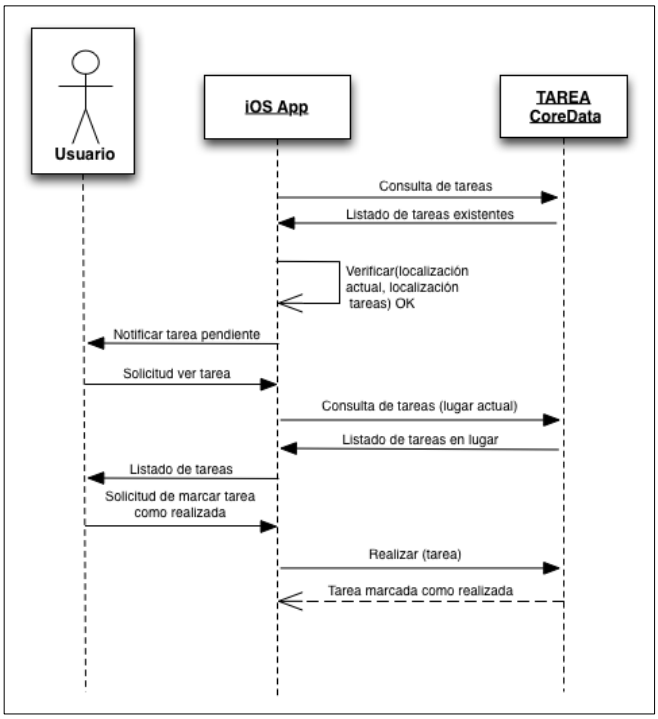


Fig. 3.4: Diagrama de Secuencias - Realizar una tarea

Diagrama de Secuencias: Contactar al desarrollador

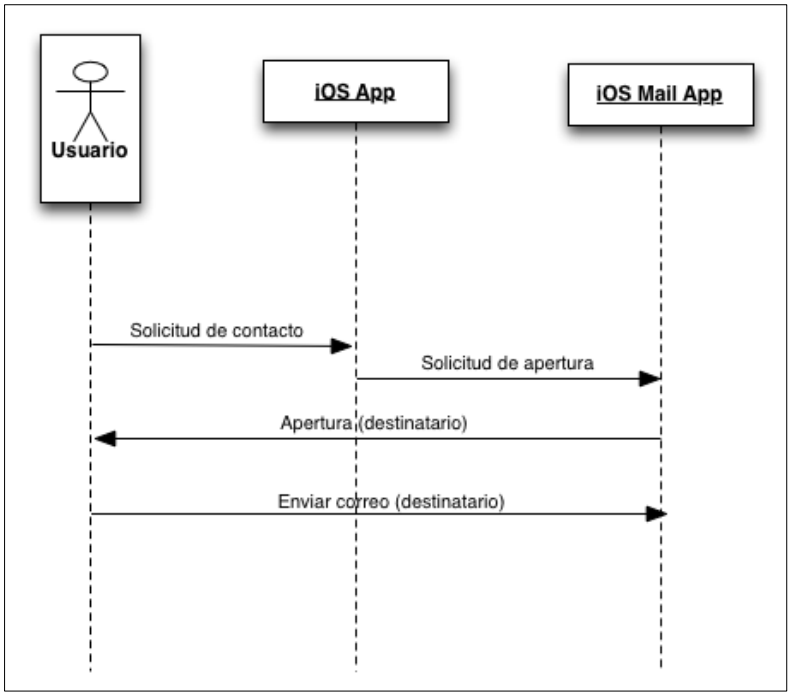


Fig. 3.5: Diagrama de Secuencias - Contactar al desarrollador

Diagrama de Secuencias: Configurar aplicación *iOS*

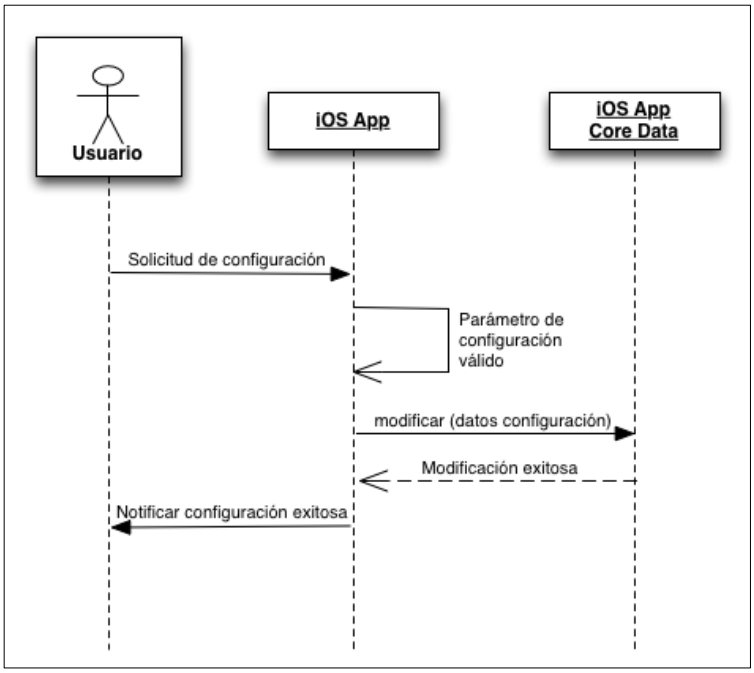


Fig. 3.6: Diagrama de Secuencias – Configurar la aplicación *iOS*

Diagrama de Secuencias: Compartir en Redes Sociales

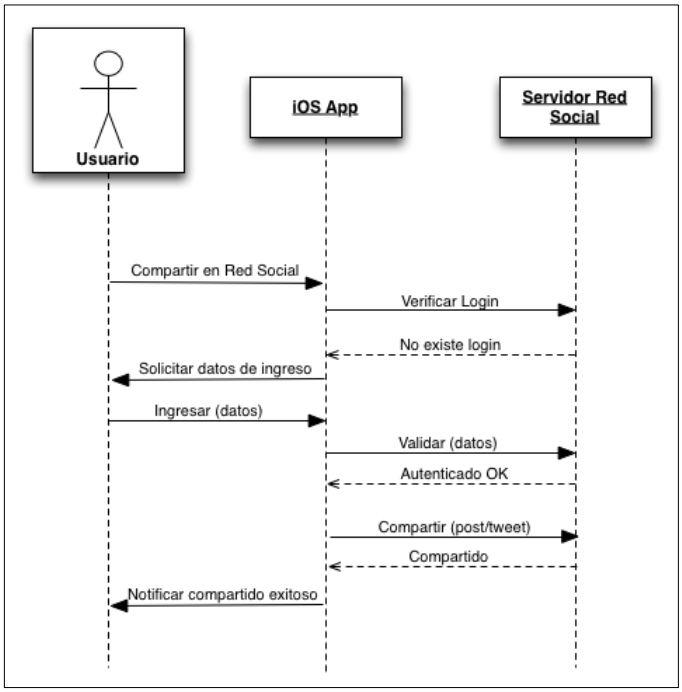


Fig. 3.7: Diagrama de Secuencias – Compartir en Redes Sociales

Diagrama de Actividades

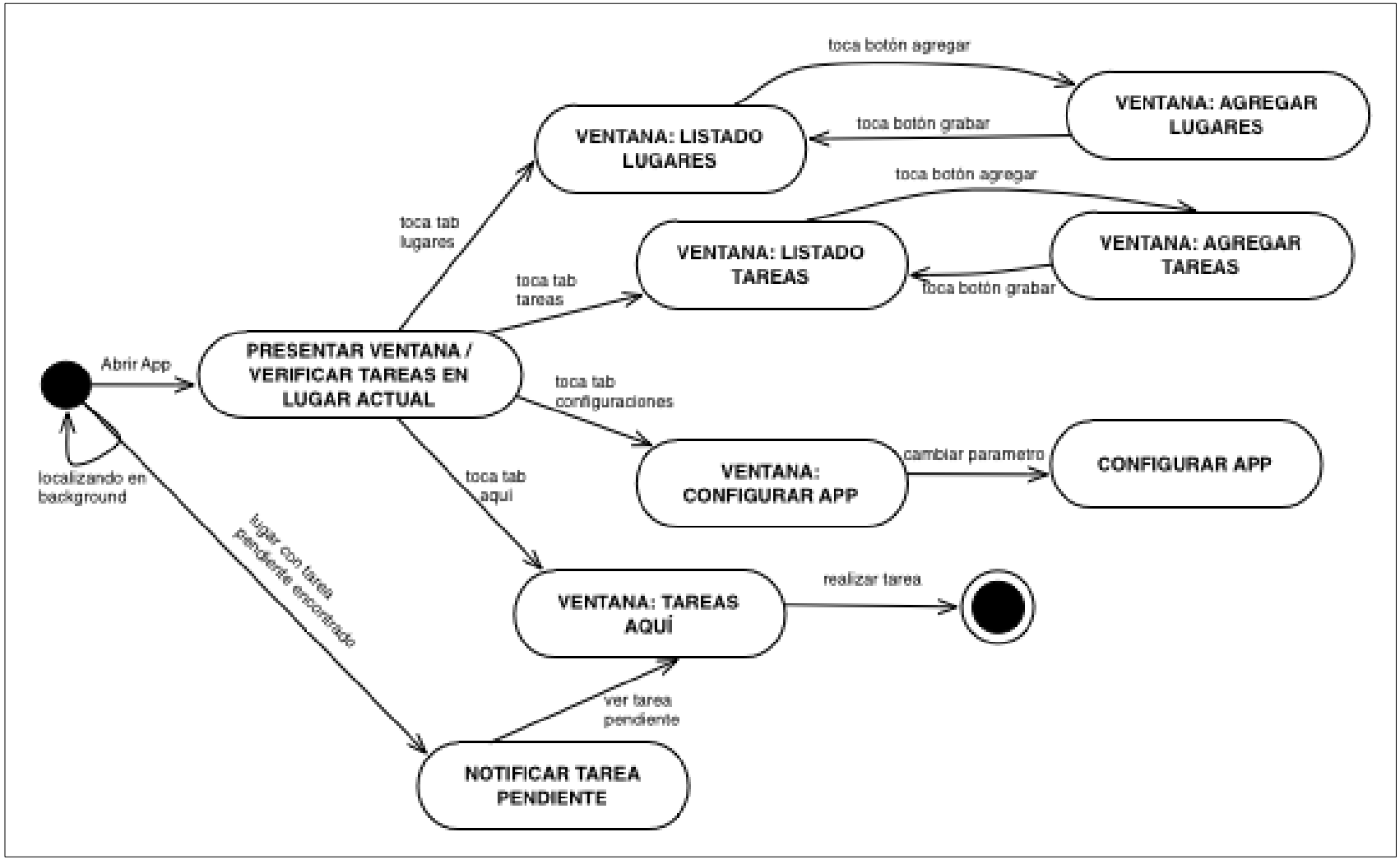


Fig. 3.8: Diagrama de actividades de la aplicación de recordatorios por *GPS GPSReminds*

3.1.4 Diseño de Base de Datos

La base de datos será implementada directamente en la herramienta de modelado de datos que ofrece el *IDE XCode* asociado con *Core Data* como se indicó en el sub apartado *"2.7.2 Creación de una aplicación que implementa Core Data"*.

Para la entidad lugar se establecerán atributos para almacenar el nombre, latitud y longitud del mismo.

Para la entidad recordatorio se establecerán atributos para almacenar su descripción y notas adicionales. Se establecerá también un atributo *booleano* que indique si se ha completado la tarea o recordatorio y otro que indique si el recordatorio ha sido notificado.

Adicionalmente se creará la relación "1..N" entre la entidad lugar y recordatorio respectivamente, debido a que un lugar puede tener varios recordatorios a la vez. A continuación se presenta el diagrama de base de datos propuesto para la aplicación *GPSReminds:*

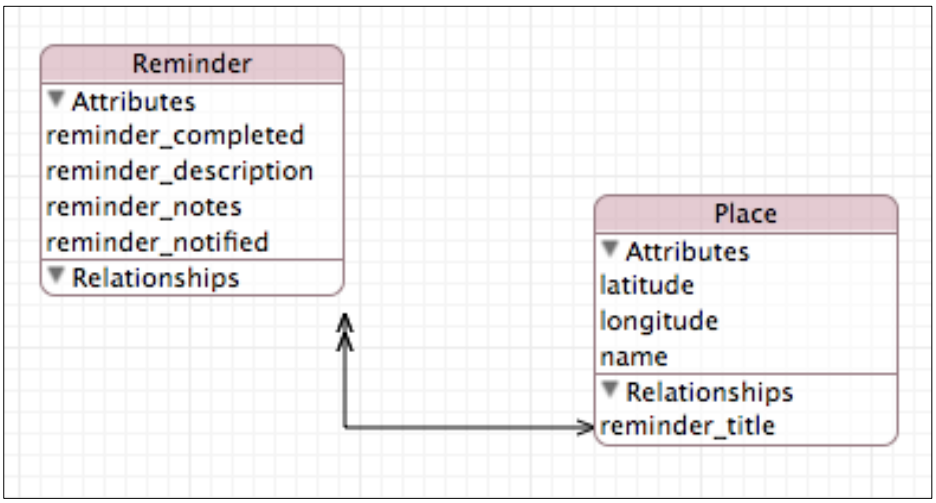


Fig. 3.9: Diagrama de base de datos de *GPSReminds*

3.1.5 Diseño de Interface

Interfaces de software:

La ventana principal de la aplicación deberá mostrar todas las vistas que la conforman. Estas vistas serán almacenadas dentro de un *"Tab Bar Controller"* en una pestaña diferente cada una. Al seleccionar cada pestaña se debería presentar el módulo o vista correspondiente. El contenido de las pestañas será: "Lugares", "Tareas", "Aquí" y "Ajustes" como se indica en la siguiente figura:

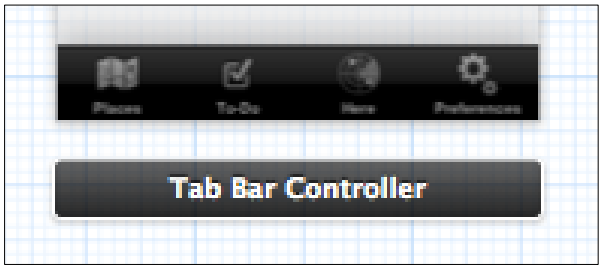


Fig. 3.10: Tab Bar Controller de la aplicación *GPSReminds*

Interfaces de usuario:

Interface para lugares: Se creará una interface de listado de lugares conformado por un componente *Table View Controller*, en donde se listará cada lugar que haya ingresado el usuario. Para el ingreso de los lugares, el usuario dispondrá de un mapa en donde deberá colocar un *pin* y escribir el nombre para el mismo.

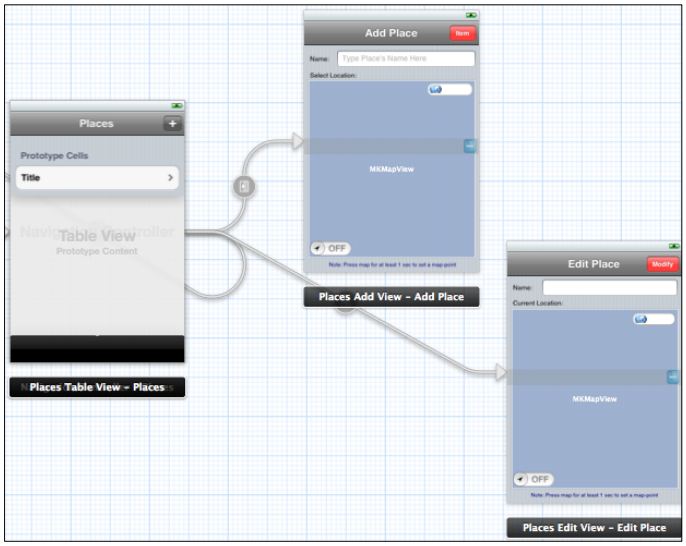


Fig. 3.11: Interface gráfica de usuario para los lugares

Interface para recordatorios: Se creará una interface de listado de tareas o recordatorios conformado por un componente *Table View Controller*, en donde se listará cada recordatorio que haya ingresado el usuario. Para el ingreso de los recordatorios, se dispondrá de un campo para ingresar su título, otro campo para agregar notas adicionales en caso de existir y un *Picker View* para seleccionar el lugar en donde se desea que se presente la notificación de la tarea.

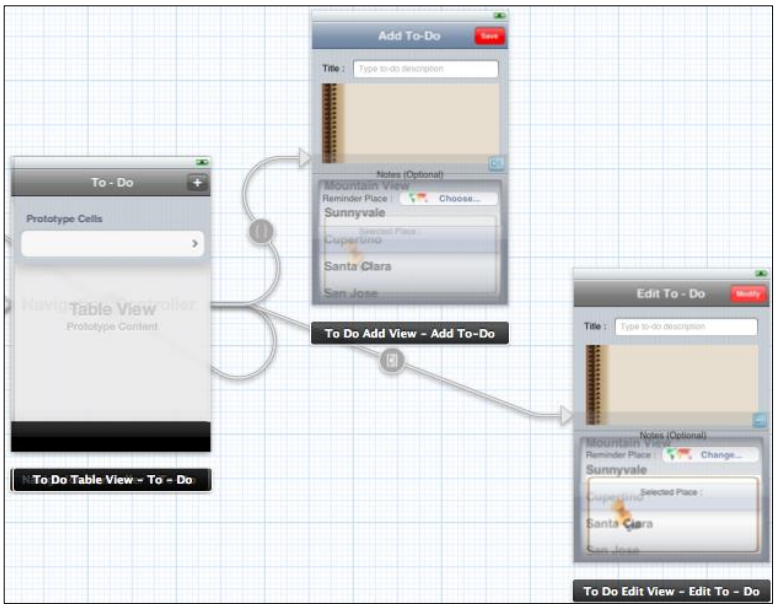


Fig. 3.12: Interface gráfica de usuario para los recordatorios

Interface para reporte de tareas en la ubicación actual (Pestaña "Aquí"): Se creará una interface de usuario que indique cada uno de los lugares que se encuentren cercanos a la ubicación actual del usuario. Cada lugar presentará las tareas (pendientes o no) que se deben realizar en el mismo. Adicionalmente si la tarea tiene notas adicionales, serán presentadas en otra vista.

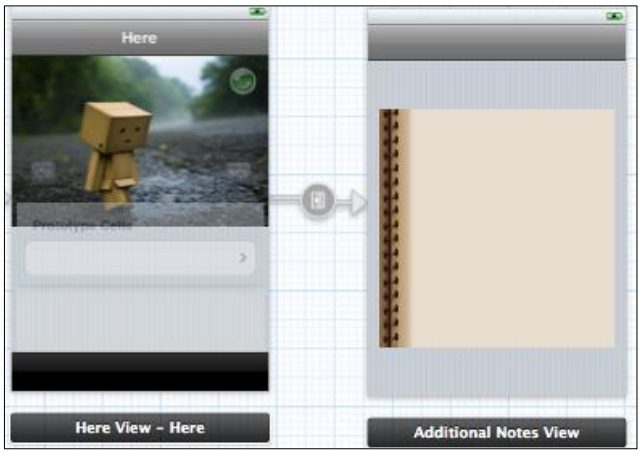


Fig. 3.13: Interface gráfica de usuario para el reporte de tareas de acuerdo a la ubicación actual del usuario

Interface para ajustes o configuraciones de la aplicación: Se creará una interface de listado de ajustes en donde se listarán parámetros de la aplicación como idioma, distancia, vista previa de la notificación, acerca de la aplicación, entre otras.

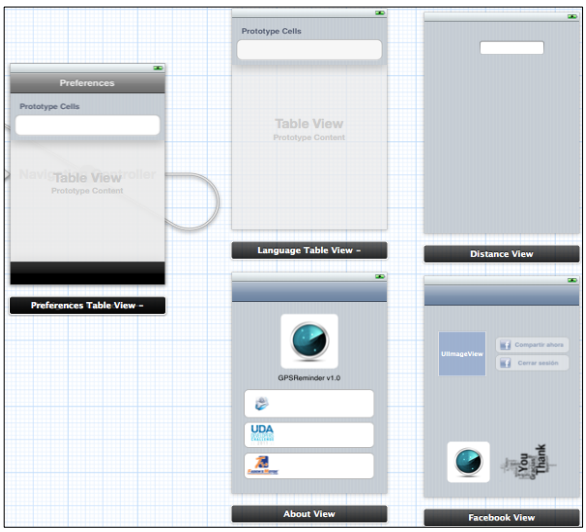


Fig. 3.14: Interface gráfica de usuario para los ajustes de la aplicación

3.2 Desarrollo de la aplicación

Para el desarrollo de la aplicación se utilizará la plantilla *"Tabbed Application"* presentada en el sub apartado *"2.6.2 Implementación de un Tab Bar Controller (Control de Pestañas)"* en donde se crearán las cuatro pestañas de la Fig. 3.10. Se construirá la interface gráfica de usuario de cada vista correspondiente a las pestañas anteriores y después se las implementará escribiendo código fuente en la clase controladora de cada una de ellas.

Se implementará la base de datos de la Fig. 3.9 y se generarán las clases que componen la misma como se ha explicado en el sub apartado *"2.7.2 Creación de una aplicación que implementa Core Data"*. Se investigará la implementación del *framework Core Location* de forma de poder utilizar el *GPS* del dispositivo *iOS.*

Para los listados de lugares, recordatorios y ajustes de la aplicación se implementarán componentes *Table View* como se ha explicado en el sub apartado *"2.5.4 Eventos comunes de Table Views"*.

En las configuraciones de la aplicación, al presionar algún ajuste específico del *Table View Controller*, se llamará a otra vista mediante el componente *Navigation Controller*. Para su implementación se hará referencia al sub apartado *"2.6.1 Implementación de un Navigation Controller (Control de Navegación)"*.

Se permitirá indicar en las redes sociales *Facebook* y *Twitter* que el usuario está utilizando la aplicación. Para esto, se hará referencia a los sub apartados: *"2.8.2 Implementación del framework de Twitter"* y *"2.8.3 Implementación del framework de Facebook"*.

Se implementará la aplicación en dos idiomas (inglés y español) de manera de obtener una mejor acogida por parte de los usuarios. El idioma se podrá establecer desde los ajustes de la aplicación.

3.3 Pruebas de la aplicación

Una vez desarrollada la aplicación *GPSReminds* se procederá a realizar las pruebas pertinentes para la misma.

Pruebas de Funcionalidad: Permitirán verificar la correcta funcionalidad de la aplicación que se ha desarrollado.

- **Interacción con un usuario:** Solicitar a un usuario que desconozca la aplicación que la pruebe y realice comentarios sobre la misma, de forma de comprobar que éste pueda agregar lugares y asignar tareas para los mismos. De igual manera el usuario debe poder configurar la aplicación en la manera que desee, es decir modificar los parámetros de distancia, idioma, privacidad en notificaciones, entre otras.

- **Validación de entradas:** Validar las entradas que recibe la aplicación. Para validar las entradas que recibe la aplicación se han elaborado los siguientes cuadros o casos de prueba:

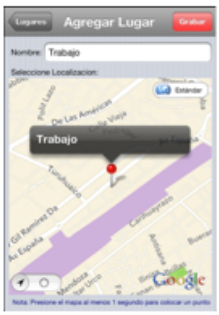
Vista	Descripción de Entrada	Valor	Salida Esperada	Observaciones
	**NOMBRE:** Campo que contiene el nombre del lugar a agregar			
	Se ingresa un nombre correctamente	Trabajo	Trabajo	El sistema acepta el valor como nombre del lugar a agregar
	Si el campo está vacío	-	Error: Escriba un nombre para el lugar	El sistema no permite agregar el lugar pues falta su nombre
	PIN: Marcador de un lugar en el mapa			
	Se coloca un pin en el mapa	PIN	PIN	El sistema acepta el pin como indicador de un lugar en el mapa
	No se coloca un pin en el mapa	-	Error: Coloque un punto en el mapa	El sistema no permite agregar el lugar pues falta su ubicación
	NOMBRE, PIN:			
	No se coloca un pin ni se escribe el nombre	-	Error: Coloque un punto en el mapa, escriba un nombre para	El sistema no permite agregar el lugar pues falta su nombre y ubicación
	Se coloca un pin en el mapa y se escribe un nombre para el lugar	Trabajo, PIN	Trabajo, PIN	El sistema agrega el nuevo lugar

Fig. 3.15: Caso de prueba: Validación de entradas de la vista Agregar Lugar

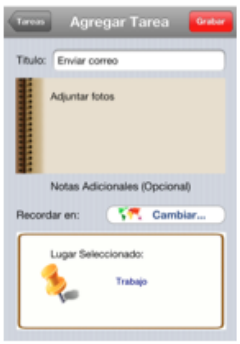
Vista	Descripción de Entrada	Valor	Salida Esperada	Observaciones
	**TITULO**: Campo que contiene el nombre de la tarea a agregar			
	Se ingresa un titulo de tarea correctamente	Enviar correo	Enviar Correo	El sistema acepta el valor como titulo de la tarea a agregar
	Si el campo está vacío	-	Error: Escriba un titulo para la tarea	El sistema no permite agregar la tarea pues falta su titulo
	NOTAS ADICIONALES: Campo que contiene las notas adicionales de la tarea a agregar			
	Se ingresan notas adicionales	Adjuntar fotos	Adjuntar fotos	El sistema acepta el valor como notas adicionales
	No se ingresan notas adicionales	-	-	El sistema acepta el valor como notas adicionales, pues no son requeridas
	RECORDAR EN: Picker View que permite seleccionar el lugar en donde se notificará al usuario sobre la tarea			
	No se selecciona un lugar	-	Error: Seleccione un lugar en donde quiera ser recordado	El sistema no permite agregar la tarea pues falta el lugar en donde se
	Se selecciona un lugar	Trabajo	Trabajo	El sistema acepta el lugar para asignarlo a la tarea
	TITULO, RECORDAR EN:			
	No se escribe un titulo ni se selecciona un lugar	-	Error: Escriba un titulo para la tarea y selecione un lugar	El sistema no permite agregar la tarea pues falta su titulo y lugar
	Se escribe un titulo y se selecciona un lugar	Enviar correo, Trabajo	Enviar correo, Trabajo	El sistema agrega la nueva tarea

Fig. 3.16: Caso de prueba: Validación de entradas de la vista Agregar Tarea

Vista	Descripción de Entrada	Valor	Salida Esperada	Observaciones
	**DISTANCIA:** Campo que contiene un valor numérico que indica la distancia en km o millas hasta el lugar para presentar la notificación			
	Se ingresa una distancia correctamente	0,15	150 mts de distancia	El sistema configura el nuevo parámetro para la distancia
	Si el campo esta vacío	-	Error: Escriba un valor válido para la distancia	El sistema solicita el ingreso de un valor válido
	Se ingresa mas de un punto	0,8,	Error: Escriba máximo un punto	El sistema solicita el ingreso de un valor válido
	Se ingresa un valor inválido	,	Error: Escriba un valor válido para la distancia	El sistema solicita el ingreso de un valor válido

Fig. 3.17: Caso de prueba: Validación de entradas de la vista Distancia

- **Modificación de registros:** Para realizar la modificación de lugares y tareas, se debe presionar el botón de color azul junto a cada una de las mismas. A continuación se presentarán las mismas opciones de ingreso; es posible modificar un lugar o una tarea a la vez y si se ingresa un nombre de lugar o título de tarea existente, la aplicación lo notificará con un mensaje "**El nombre/título del lugar/tarea ya existe**" en ese caso el usuario deberá ingresar un nuevo nombre o título.

- Eliminado de registros: Para eliminar estos registros, se realizará directamente en la vista que presenta el listado de los mismos, siguiendo el clásico método de eliminado de registros que presenta *iOS.* Si se elimina un lugar que tenía asignado una o varias tareas, se realiza el proceso de borrado en cascada, es decir se elimina el lugar junto a todas las tareas pendientes asignadas al mismo. El usuario es notificado sobre esto, pues al querer borrar un lugar que tiene una tarea asignada se presentará el mensaje: **"Advertencia: Existe una tarea activa en este lugar. Si elimina este lugar, la tarea activa será también ELIMINADA. ¿Continuar?"**, y tiene la opción de proseguir o cancelar

3.4 Envío para revisión y aprobación para comercialización en AppStore

Una vez desarrollada la aplicación antes de enviarla para el proceso de revisión y aprobación para comercialización en la tienda *AppStore*, después de haber realizado las pruebas pertinentes en el apartado anterior, se han revisado todos los problemas y advertencias que ha presentado el *IDE*.

A continuación se agregó la aplicación en el portal *iTunes Connect*, en donde se escribió su nombre, descripción, versión, entre otros. También se agregó capturas de pantalla de la aplicación en ejecución en los dispositivos en los que será compatible, apoyándose en el apartado *"2.13 Cómo subir una aplicación a la tienda AppStore"*.

Cuando la aplicación estuvo estado *"Waiting for Upload"* en *iTunes Connect*, antes de subir la aplicación, se consideraron todas las recomendaciones presentadas en el sub apartado *"2.13.1 Consideraciones para evitar el rechazo de aplicaciones"*.

Una vez enviada la aplicación para su revisión, después de los siguientes siete días, se recibió un correo indicando que la aplicación estaba siendo revisada.

Lamentablemente, inicialmente fue rechazada, debido a que al utilizar los mapas ofrecidos por la empresa *Google*, se debe indicar dentro de la aplicación ya sea con un componente *Label* o *Image View*, que los mapas son de la empresa antes mencionada.

Al corregir este problema se envió nuevamente la aplicación para su revisión. Esta vez, después de siete días, la aplicación fue aprobada y se encuentra ahora disponible en la tienda *AppStore* para su descarga a nivel mundial por un valor de $1.99.

Fig. 3.18: Aplicación *iOS GPSReminds* disponible en la tienda *AppStore*

3.5 Aplicación de sugerencias del tutorial

En los siguientes sub apartados, se implementará las sugerencias que se han presentado en el tutorial, como la generación de una versión gratuita de la aplicación, implementación de publicidad en la aplicación gratuita de manera de generar ingresos a través de la misma, y, la incorporación de la clase *AppiRater* a la aplicación de manera de conseguir que los usuarios que compren la versión de pago o descarguen la aplicación gratuita las valoren en la tienda *AppStore* y escriban una buena reseña sobre las mismas.

3.5.1 Generación de una versión gratuita

Para generar una versión gratuita de la aplicación *GPSReminds*, primero se definieron las funcionalidades que tendría la misma: La aplicación permitirá a los usuarios agregar lugares y recordatorios indefinidamente y además el usuario podrá configurar la aplicación a su antojo en cuanto a parámetros de distancia. Esta aplicación gratuita notificará a los usuarios sobre tareas pendientes en lugares cercanos solamente tres veces. Después de la tercera vez, no se notificará más sobre las tareas que haya programado el usuario.

Para evitar que los usuarios desinstalen la aplicación y la vuelvan a instalar con el fin de conseguir tres nuevos recordatorios de forma gratuita, se utilizará un identificador único que será almacenado en una *SimpleDB* de los servicios web de *Amazon* utilizando un *Token Vending Machine* para asegurar la transferencia de información entre la aplicación y la *SimpleDB.* Para esto, se hará referencia a los apartados *"2.15 Identificación única de dispositivos iOS"* y *"2.16.4 Desarrollo de una aplicación iOS comunicada con un Amazon SimpleDB mediante un Token Vending Machine"*

Se creará un nuevo ícono para la aplicación con un banner que indique que es gratuita y se creará también una vista inicial en la cual se controle si la aplicación ha sido instalada previamente en el dispositivo *iOS.*

Se ha desarrollado la aplicación gratuita y actualmente se encuentra disponible para su descarga a nivel mundial en la tienda *AppStore* como se puede observar en la Fig. 3.19.

3.5.2 Implementación de iAds

Para generar ingresos mediante la aplicación gratuita creada en el sub apartado anterior, se implementarán banners publicitarios dentro de la misma.

Para esto, se hará referencia al sub apartado *"2.17.1 Implementación de un banner publicitario iAd en una aplicación"*. Además se tomarán en cuenta las mejores prácticas presentadas en el sub apartado *"2.17.2 Mejores prácticas al implementar publicidad en aplicaciones"*, de forma de maximizar la oportunidad de generar ingresos por publicidad y asegurar una mejor experiencia de usuario con la aplicación.

Lamentablemente estos banners publicitarios no están disponibles a nivel mundial y son exclusivos para algunos países como Estados Unidos y China, pero pronto estarán disponibles a nivel mundial.

Como se indicó en el apartado anterior la aplicación gratuita se encuentra disponible actualmente en la tienda *AppStore*, y al ejecutarla en un país en donde se pueda visualizar la publicidad de *Apple*, se ve como la presentada en la Fig. 3.20.

Fig. 3.19: Aplicación *iOS GPSReminds Free* disponible en la tienda *AppStore*

Fig. 3.20: *GPSReminds* con un banner publicitario

3.5.3 Motivar a usuarios a calificar la aplicación mediante AppiRater

Se implementará la clase *AppiRater* de *Arash Payan* presentada en el apartado *"2.14.2 Implementación de AppiRater"* en la aplicación *GPSReminds* para motivar a los usuarios que compren la aplicación a valorarla y escribir una reseña positiva sobre la misma en la tienda *AppStore*. El evento significativo de la aplicación será una notificación sobre una tarea pendiente en un lugar cercano. El número de eventos significativos para solicitar al usuario la valoración de la aplicación será tres.

Esto quiere decir de que cuando un usuario utilice la aplicación y programe varios recordatorios, sea notificado sobre los mismos cuando se encuentre en los lugares donde fueron programados. Después de la tercera notificación, es decir, cuando el usuario se encuentre en un buen nivel de satisfacción con la aplicación, se presentará una ventana como la indicada en la Fig. 2.377 para solicitar que valore la aplicación en la tienda *AppStore*. Si el usuario accede a valorarla, será llevado directamente a la sección de reseñas de la aplicación en la tienda.

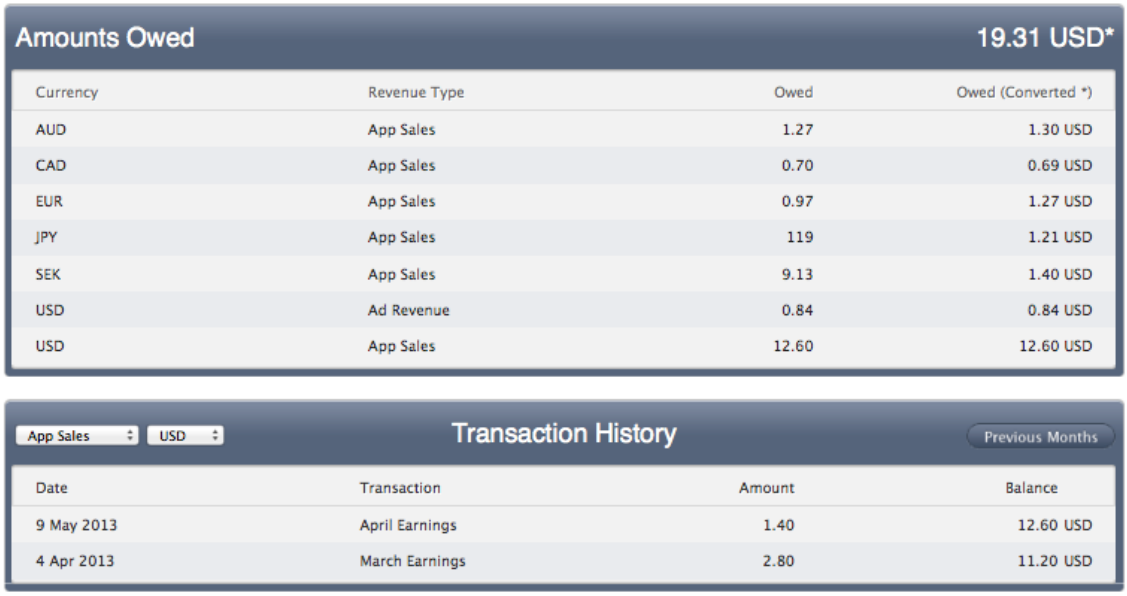
3.6 Estadísticas de descargas y ventas de la aplicación

A continuación se presentarán las estadísticas de descargas y ventas de la aplicación *GPSReminds.* Como se indicó en el sub apartado *"2.13.2 Portal iTunes Connect"*, estas estadísticas se encuentran disponibles en la sección *Payments and Financial Reports* del portal *iTunes Connect*.

Fig. 3.21: Resumen de ventas de la aplicación

Las ganancias generadas por las ventas de la aplicación se presentan en un reporte que puede ser filtrado por meses.

Fig. 3.22: Resumen de ganancias perteneciente al mes de marzo del 2012



Amounts Owed 19.31 USD*

Currency	Revenue Type	Owed	Owed (Converted *)
AUD	App Sales	1.27	1.30 USD
CAD	App Sales	0.70	0.69 USD
EUR	App Sales	0.97	1.27 USD
JPY	App Sales	119	1.21 USD
SEK	App Sales	9.13	1.40 USD
USD	Ad Revenue	0.84	0.84 USD
USD	App Sales	12.60	12.60 USD

Date	Transaction	Amount	Balance
9 May 2013	April Earnings	1.40	12.60 USD
4 Apr 2013	March Earnings	2.80	11.20 USD

Fig. 3.23: Ganancias después de los impuestos

Conclusiones

Al culminar el desarrollo de la aplicación *iOS GPSReminds,* se ha demostrado que se puede utilizar el tutorial del capítulo anterior para desarrollar aplicaciones *iOS.* Lo único que se debe hacer, es conocer los apartados en donde se encuentran los temas necesarios para la aplicación que se desee desarrollar.

Se presentó el análisis y diseño de la aplicación para explicar su funcionalidad. La descripción de cada uno de los casos de uso junto con los diagramas adicionales presentados complementa la comprensión de la funcionalidad de la aplicación desarrollada.

El diseño de interface propuesto, representa las ventanas que componen a la aplicación *iOS.*

Se detalló cada apartado del tutorial al que se hizo referencia tanto para el desarrollo como para la distribución de la aplicación *GPSReminds.*

Para garantizar la correcta funcionalidad de la aplicación se desarrollaron distintos casos de prueba, antes de proceder con la distribución de la misma.

Se tomaron en cuenta las recomendaciones del tutorial desarrollado en el capítulo anterior para generar ingresos adicionales y conseguir valoraciones y reseñas positivas sobre la aplicación en la tienda *AppStore.*

Se presentaron distintas estadísticas de descargas de la aplicación, las cuales se pueden encontrar en el portal de desarrolladores de *Apple iTunes Connect.*

La aplicación *GPSReminds* se encuentra actualmente disponible para su descarga en la tienda *AppStore,* tanto en su versión gratuita como su versión comercial.

Capítulo 4: Manual de usuario de la aplicación

De acuerdo a la Fig. 2.359 del sub apartado *"2.13.6 Administración de aplicaciones en iTunes Connect y distribución"* perteneciente al tutorial desarrollado en el capítulo 2, para publicar una aplicación en la tienda *AppStore*, es necesario que el desarrollador provea una dirección *URL* correspondiente al portal que brinde soporte para la misma.

Para la aplicación *GPSReminds*, se desarrollará un manual de usuario que será alojado en el portal web *chappssupport.blogspot.com*, dirección *URL* que fue proveída en el portal *iTunes Connect* al momento de crear la aplicación.

En el manual de usuario que se desarrollará en este capítulo, se explicarán los principales módulos de la aplicación. Es decir, se indicará como agregar lugares en un mapa y se indicará como crear nuevas tareas o recordatorios y asignarlos a los lugares previamente agregados. De igual manera, se indicará como configurar la aplicación de acuerdo a las necesidades del usuario.

Se indicará también como visualizar los reportes de lugares, tareas y el reporte de tareas de acuerdo a la ubicación actual del usuario.

De esta forma, se espera que el lector se familiarice con el funcionamiento de la aplicación. La misma que puede ser descargada de la tienda *AppStore*.

4.1 Módulo de registro de lugares

Es la pestaña inicial de la aplicación (Fig. 4.1), debido a que es el pilar fundamental de la misma. Sin lugares, no habrá como programar recordatorios. Inicialmente, la aplicación indica que aún no se ha agregado ningún lugar por lo que para añadir uno se debe tocar el botón estándar de iOS: 

Agregar lugares: Al presionar el botón anterior se presenta esta ventana, en donde se pueden agregar lugares de forma interactiva. Se utilizan los mapas de *Google* o *Apple* dependiendo del *iOS* que ejecute el dispositivo. Se puede visualizar este mapa de distintas formas:

Fig. 4.1: Vista de Lugares

Fig. 4.2: Tipos de vistas de mapa

Se debe ingresar el nombre del lugar y colocar un punto en el mapa que indique la localización del mismo. Para colocar un punto en el mapa se lo debe presionar, en el lugar donde se desee colocar el *pin*, durante al menos un segundo.

El *switch* 

indica la posición del usuario sobre el mapa.

Para agregar el lugar, se debe presionar el botón 

Posibles errores al agregar un lugar:

- Error: Coloque un punto en el mapa. Se presenta debido a que se no se ha colocado un *pin* en el mapa que indique la ubicación del lugar que se agregará

- Error: Escriba un nombre para el lugar. Se presenta si se ha colocado un *pin* en el mapa pero no se ha escrito un nombre para el mismo.

Listado de lugares: Después de agregar un lugar, el listado de lugares de la Fig. 4.1 se actualizará (Fig. 4.4). Se puede presionar el botón  para modificar un lugar previamente almacenado.

Fig. 4.4: Listado de lugares

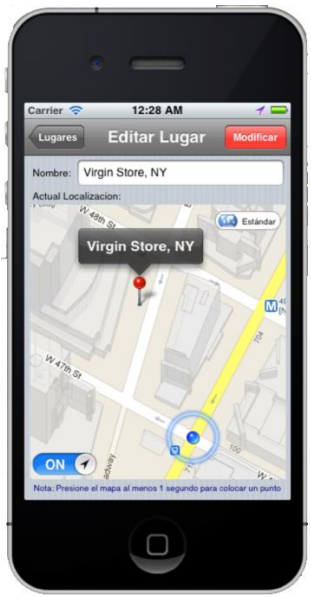


Fig. 4.5: Vista para modificar lugares

Fig. 4.3: Advertencia de modificación tarea asignada a un lugar

Modificar lugares: Al presionar el botón previamente mencionado, se presentará la ventana que permite modificar un lugar (Fig. 4.5). Se puede modificar tanto el *pin* que indica la localización del lugar como su nombre. No se puede agregar ni modificar un lugar con el mismo nombre de uno previamente almacenado.

Para modificar el lugar se debe presionar el botón 

. Si el lugar que se modifica tiene una tarea asignada al mismo, se presentará la ventana de la Fig. 4.3.

Eliminar lugares: Para eliminar un lugar se puede presionar el botón "Editar" de la ventana de listado de lugares y proceder a eliminar el lugar deseado (Fig. 4.6). Se puede también utilizar la característica de *iOS*: *Swipe To Delete* que consiste en deslizar el dedo sobre el lugar que se desee eliminar (Fig. 4.7).

Para confirmar el borrado de un lugar, se debe presionar el botón 

Fig. 4.6: Eliminar un lugar

Fig. 4.7: Eliminar un lugar *(Swipe To Delete)*

4.2 Módulo de registro de tareas

La ventana de tareas indica inicialmente que no existe ninguna tarea almacenada (Fig. 4.10). Para poder agregar una tarea se debe haber agregado previamente al menos un lugar. Para agregar una nueva tarea se debe presionar el botón  .

Si no se ha agregado un lugar previamente se muestra la ventana de la Fig. 4.8, caso contrario se presenta la ventana de ingreso de tareas Fig. 4.11.

Fig. 4.8: Ventana informativa

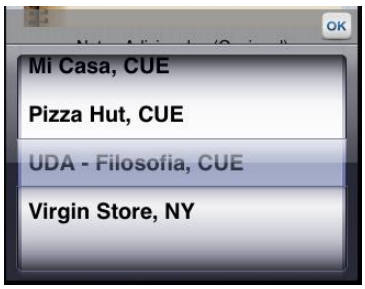


Fig. 4.9: Picker View de lugares

Fig. 4.10: Vista de tareas

Fig. 4.11: Vista para agregar tareas

Agregar Tareas: Para agregar una nueva tarea se debe comenzar por escribir un título para la misma, opcionalmente se pueden escribir notas adicionales para ésta en el recuadro color café como se puede observar en la Fig. 4.11. Finalizar seleccionando el lugar en donde se desea ser recordado sobre esta tarea presionando 

. Este botón presentará un *Picker View* cuyos contenidos serán los lugares previamente agregados, como se puede observar en la Fig. 4.9. Finalizar presionando 

para seleccionar el lugar.

Para grabar la tarea se debe presionar el botón 

.

Si la tarea es grabada correctamente, ha partir de este momento, el dispositivo *iOS* detectará la posición actual del usuario y lo seguirá monitoreando hasta que llegue a alguno de sus destinos y sea notificado sobre sus tareas pendientes. Si no se puede grabar la tarea, se puede presentar alguno de los siguientes errores:

Posibles errores al agregar una tarea:

- Error: Escriba un título para la tarea. Se presenta si no se ha escrito un título para la tarea que se desea agregar.

- Error: Seleccione un lugar donde quiera ser recordado. Se presenta si no se ha seleccionado un lugar de recordatorio en donde se desea ser notificado sobre la tarea pendiente.

Listado de tareas: Cada vez que se agregue una tarea, la vista de listado de tareas se actualizará con todas las tareas almacenadas. En caso de existir mas de una tarea asignada a un mismo lugar, se agruparán las tareas por lugar (Fig. 4.13). En este mismo listado, se puede tocar una tarea para marcarla como realizada como se muestra en la Fig. 4.12.

Fig. 4.13: Listado de tareas

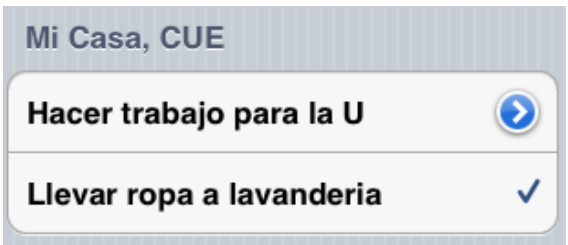


Fig. 4.12: Tocar una tarea para marcarla como realizada

Modificar tareas: Para modificar una tarea se debe presionar el botón correspondiente a la tarea deseada en el listado de tareas. Al presionarlo se presentará una ventana exactamente igual a la de ingreso de tareas, con los datos de la tarea a modificar. Si se la modifica se debe finalizar presionando el botón Modificar.

Eliminar tareas: Para modificar una tarea se puede seguir cualquiera de los dos métodos explicados para eliminar lugares en el apartado anterior, como se indica en las siguientes figuras.

Fig. 4.14: Eliminar una tarea

Fig. 4.15: Eliminar una tarea *(Swipe To Delete)*

4.3 Reportes

La aplicación *GPSReminds* permitirá generar reportes o listados de los lugares agregados, así como de las tareas almacenadas filtradas por lugar.

Adicionalmente permitirá visualizar un reporte de tareas pendientes en lugares cercanos a la ubicación actual del usuario. Estos reportes o listados permitirán visualizar las tareas realizadas, así como las tareas por realizar y sus notas adicionales.

4.3.1 Reporte de lugares

El reporte de lugares es el presentado en la Fig. 4.4, en este reporte se listan todos los lugares agregados por el usuario. En este reporte o listado se puede seleccionar un lugar para visualizar su ubicación en un mapa o modificar los datos del mismo, ya sea su nombre o ubicación. Adicionalmente, en este listado se pueden eliminar los lugares existentes.

4.3.2 Reporte de tareas asociadas a lugares

El reporte de tareas asociadas a lugares es el presentado en la Fig. 4.13, en este reporte se listan todas las tareas agregadas por el usuario. En este reporte o listado se puede seleccionar una tarea para visualizar sus notas adicionales o modificar los datos de la misma, ya sea su título, notas adicionales o lugar asociado a ésta. Adicionalmente, en este listado se pueden eliminar las tareas existentes.

Si existe mas de una tarea asignada a un mismo lugar, se agruparán las tareas de acuerdo al lugar en donde deben ser realizadas.

En este reporte se puede también tocar una tarea para marcarla como realizada. Cuando no existan tareas pendientes o todas estén realizadas, se dejará de utilizar el *GPS* del

dispositivo *iOS*. Una tarea realizada tiene como accesorio un *visto* o *checkmark* ( ). Una tarea por realizar tiene como accesorio un *disclosure button* ( ).

4.3.3 Reporte de tareas de acuerdo a proximidad de lugares

El reporte de tareas de acuerdo a la proximidad del usuario a un lugar específico previamente almacenado se puede visualizar en la tercera pestaña de la aplicación denominada "Aquí".

En esta pestaña aparecerán los lugares cercanos a la ubicación actual del dispositivo *iOS*, en caso de existir algún lugar cercano.

En caso de que el usuario se encuentre cerca de uno o varios lugares, se detallará la distancia hasta los mismos junto a sus tareas pendientes como se indica a continuación:

Fig. 4.16: Reporte de tareas pendientes en un lugar cercano

En este reporte, se pueden marcar las tareas como realizadas (al presionar la tarea deseada) o visualizar sus notas adicionales en caso de existir, al presionar el botón de la tarea, como se indica en la siguiente figura.

Fig. 4.17: Reporte de tareas en lugares cercanos (pestaña "Aquí")

Si se encuentra mas de un lugar cercano a la ubicación actual del usuario, se mostrarán flechas de desplazamiento que permitan mostrar los diferentes lugares cercanos junto a cada una de sus tareas pendientes

Fig. 4.18: Se ha encontrado más de un lugar cercano en el reporte

4.4 Configuraciones o preferencias del sistema

En las configuraciones de la aplicación se encuentran algunos parámetros que podrían ser configurados a conveniencia del usuario. A continuación se presenta la ventana de configuraciones de la aplicación *GPSReminds* (Fig. 4.19).

La primera vista, "Acerca De", permite visualizar la versión de la aplicación y contactar al desarrollador a través de correo electrónico (Fig. 4.20).

Fig. 4.19: Configuraciones de la aplicación *GPSReminds*

Fig. 4.20: Ventana "Acerca De" la aplicación

Configuración de idioma: Vista que permite seleccionar de una lista el idioma de la aplicación. Actualmente la aplicación soporta únicamente el idioma inglés y español como se puede ver en la Fig. 4.21.

Configuración de distancia: Vista que permite ingresar el valor de la distancia hasta un lugar, en donde se desea que se presente la notificación. Por ejemplo se puede escribir el valor "1" para presentar la notificación cuando se encuentre a un kilómetro de distancia del lugar en donde se debe realizar la tarea pendiente. A medida que el

usuario escriba un valor para la distancia, el texto de la parte inferior indicará la distancia en metros correspondiente al valor decimal o entero ingresado, como se puede observar en la Fig. 4.22.

Fig. 4.21: Selección del idioma de la aplicación

Fig. 4.22: Configuración del parámtro distancia de la aplicación

Unidades: Parámetro que permite seleccionar que tipo de unidad de medida se utilizará para la distancia configurada anteriormente. Se puede elegir entre kilómetros y millas. 

.

Vista Previa: Parámetro de la aplicación que permite configurar la privacidad de las notificaciones presentadas por la misma. Si se activa el *switch*, la notificación presentará el título de la tarea pendiente, caso contrario únicamente indicará que se tiene una tarea pendiente en un lugar cercano.

Economizar Batería: Parámetro de la aplicación que permite un ahorro significativo de la batería, debido a que al estar activo, la aplicación realiza un monitoreo de actualizaciones significativas de localización del dispositivo *iOS* y por lo tanto las

notificaciones podrían tardar en presentarse. Si se desactiva el *switch*, el monitoreo de cambios de posición del dispositivo *GPS* se realiza cada segundo, asegurando que la notificación se presentará tan pronto como el usuario se acerque a un lugar con una tarea pendiente, pero implica un consumo superior de batería.

Compartir: Permite indicar en las principales redes sociales (*Facebook* y *Twitter*) que el usuario se encuentra utilizando la aplicación, como se indica a continuación:

Fig. 4.23: Publicar en Facebook indicando que se está utilizando la aplicación

Fig. 4.24: Enviar un *Tweet* indicando que se esta utilizando la aplicación

Conclusiones

Al concluir el desarrollo del manual de usuario de la aplicación *iOS GPSReminds* se ha explicado de manera detallada, como ingresar, modificar y eliminar un lugar. Se ha explicado también como crear recordatorios o tareas y asignarlos a lugares, de manera de que el usuario será notificado tan pronto como se encuentre cerca de un lugar con una tarea pendiente.

Se ha indicado que estas tareas o recordatorios pueden ser también eliminados o modificados.

Como se puede observar, se ha explicado a los usuarios que se encuentran en la capacidad de interactuar con los reportes de tareas y lugares. El reporte de tareas de acuerdo a proximidad de la ubicación actual del usuario ha sido presentado.

Para culminar, se ha explicado la forma en la que se puede configurar a la aplicación de acuerdo a las necesidades del usuario. Se explicó como configurar los parámetros idioma, distancia, unidad de distancia y vista previa en notificaciones.

De esta forma el usuario que descargue de la tienda *AppStore* la aplicación *iOS GPSReminds,* podrá referirse a este manual de usuario si tiene alguna consulta sobre el funcionamiento de la misma.

CONCLUSIONES GENERALES

Al finalizar el desarrollo de este trabajo de graduación, se ha creado un tutorial para explicar el procedimiento de desarrollo de aplicaciones para dispositivos *iOS*. Para garantizar una mejor comprensión de los lectores, no solo se explicaron los pasos a seguir para crear una aplicación determinada, sino que también se ha explicado el funcionamiento del *IDE XCode*, se desarrollaron distintos análisis que fueron añadidos como sub apartados del tutorial y se desarrollaron aplicaciones demostrativas para la mayoría de sub apartados.

Para demostrar que se puede utilizar el tutorial para desarrollar aplicaciones *iOS*, se ha creado una aplicación de recordatorios por *GPS* haciendo referencia a varios de los apartados del tutorial. Esta aplicación se encuentra actualmente disponible para su descarga en la tienda *App Store* con el nombre de *GPSReminds.* Su funcionamiento, se detalló en el cuarto capítulo de éste documento en donde se ha desarrollado el manual de usuario de la misma.

Se ha creado también una versión *iBook* del tutorial, de manera de que los lectores puedan interactuar con las imágenes, abrir directamente los enlaces web, realizar anotaciones, entre otros.

Se desarrolló una guía para distribuir aplicaciones en la tienda *AppStore*, la cual fue agregada dentro del mismo tutorial como apartado *"2.13 Cómo subir una aplicación a la tienda AppStore"* y sus sub apartados.

De igual manera, se generó una recopilación de acontecimientos en el proceso de revisión de una aplicación a ser publicada en *AppStore* y se la presentó en el sub apartado *"2.13.6 Administración de aplicaciones en iTunes Connect y distribución"* del tutorial.

RECOMENDACIONES

La recomendación principal es la suscripción como desarrollador de *Apple*, lo cual tiene un costo anual pero brinda varios beneficios. Al ser un desarrollador, se puede probar las aplicaciones desarrolladas en dispositivos *iOS* físicos, distribuirlas en la tienda *AppStore* y generar ingresos con las ventas de las mismas.

Además, los desarrolladores tienen acceso a los sistemas operativos en versión beta que aún no se encuentran disponibles para todos los usuarios, como es el caso del *iOS* 7 en este momento, el cual estará disponible para todos los usuarios en otoño; pero al ser un desarrollador se lo podría descargar ahora mismo y desarrollar aplicaciones compatibles con el mismo.

Se recomienda también leer el tutorial presentado en el capítulo dos de éste documento para desarrollar aplicaciones *iOS* de la mejor manera.

Otra recomendación es la de trabajar con las versiones mas recientes de *iOS* y *XCode* disponibles exclusivamente para desarrolladores. Así se podrá desarrollar aplicaciones compatibles con los nuevos dispositivos y sistemas operativos de *Apple*.

Los desarrolladores disponen de un número limitado de dispositivos físicos en los que se pueden probar las aplicaciones y los sistemas operativos en versión beta, por lo que se recomienda asignar diferentes tipos de dispositivos *iOS* y de manera cautelosa.

GLOSARIO

Amazon Inc.: Compañía estadounidense de comercio electrónico con sede en *Seattle, Washington*. Fue una de las primeras grandes compañías en vender a través del internet.

Acelerómetro: Dispositivo que permite medir la aceleración y las fuerzas inducidas por la gravedad, es decir, en un móvil, permite detectar el movimiento y giro del mismo.

API (Application Programming Interface): Conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software o programador.

Apple Inc.: Empresa multinacional estadounidense que diseña y produce equipos electrónicos y software. Entre sus productos más conocidos se encuentran el *iPod*, *iPhone* e *iPad*.

Apple ID: Describe una plataforma de seguridad gestionada por *Apple Inc.* que permite acceder a una gran cantidad de productos y servicios de la empresa.

AppStore: Servicio para i*Phone, iPod , iPad* y sistemas operativos de computadores *Mac OS X* que permite buscar y descargar aplicaciones distribuidas por *Apple Inc*.

Assistant: Componente de *XCode* que permite visualizar el archivo con el que se trabaja junto con archivos relacionados que pudieran afectarse, al editar el mismo.

Automatic Reference Counting (ARC): Característica a nivel de compilador que simplifica la administración del tiempo de vida o (administración de memoria) en las aplicaciones.

Búfer: Espacio reservado en memoria para almacenar datos mientras son procesados.

Bug: Error o anomalía en el código de un programa, generalmente un defecto de *software*. Es el resultado de un fallo o deficiencia durante la creación de un programa.

Build: Proceso donde se realiza revisiones de código fuente como la inclusión de clases pertenecientes a *frameworks* y otras acciones que aseguren el correcto funcionamiento de la aplicación. En caso de no poderse realizarse, se presenta el error correspondiente.

Byte: Secuencia de ocho bits contiguos comúnmente utilizado como unidad básica de almacenamiento de datos.

Clase: Construcción que se utiliza como modelo o plantilla para crear objetos. El modelo describe el estado y comportamiento que comparten todos los objetos creados.

Click: Acción de presionar un botón de un ratón sincronizado con un computador.

Cocoa: Es el *framework* para desarrollar aplicaciones para *Mac OS X*.

Cocoa Touch: *Framework* de desarrollo para dispositivos *iOS*, muy similar a *Cocoa.*

Consola (de IDE): Espacio de un IDE dedicado para presentar información durante la ejecución de una aplicación.

Core Data es un *framework* de administración y persistencia de datos manejado mediante un esquema de objetos gráficos. Ayuda a grabar objetos del en un archivo para recuperarlo después.

Delegation (Delegación): Modelo en donde un objeto en un programa actúa en coordinación con otro objeto. El objeto delegante mantiene una referencia hacia el otro objeto (delegado) y en un momento apropiado le envía un mensaje. El mensaje informa al delegado sobre un evento que el objeto delegante manejará. El delegado responde al mensaje actualizando a sí mismo u otros objetos, su apariencia o estado.

Device Family: Familia o grupo de dispositivos *iOS* para los cuales será compatible una aplicación. Puede ser *iPad, iPhone / iPod* o *Universal*

Dispositivo iOS: Dispositivo *iPhone, iPod o iPad* que funciona con sistema operativo *iOS*. Tanto el dispositivo como el sistema operativo, pertenecen a la empresa *Apple.*

Drag&Drop: Expresión informática que se refiere a la acción de arrastrar con el ratón un objeto desde un lugar hasta soltarlo en otro.

Dock de Mac: Elemento de interface gráfica de usuario que permite iniciar, cambiar y monitorizar aplicaciones en un Mac. Es una barra de accesos directos.

Estado Background: Estado de una aplicación *iOS* en el que ésta no se encuentra activa o presente, pero se encuentra ejecutando código.

Estado Foreground: Estado activo de una aplicación *iOS* en el cual ésta se encuentra presente y está en capacidad de recibir y manejar eventos.

Finder: Aplicación ejecutada en el sistema operativo *Mac OS X* responsable de la gestión total de archivos de usuario, discos, red y el lanzamiento de otras aplicaciones.

Firmware: Es el *software* que se encuentra embebido en una pieza de *hardware* para poder controlarlo.

Framework: En el desarrollo de *software* es una estructura de soporte definida, en la cual otro proyecto puede ser organizado y desarrollado. Incluye programas, bibliotecas, entre otros, para unir diferentes componentes de un proyecto y ayudar en su desarrollo.

Graphical User Interface (GUI): Programa informático que actúa como interface de usuario utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en una interface. Su principal uso, consiste en proporcionar un entorno visual para permitir la comunicación con el sistema operativo.

GIT: Software de control de versiones diseñado por *Linus Torvalds* pensado en la eficiencia y confiabilidad de mantenimiento de versiones de aplicaciones cuando tienen un gran número de archivos de código fuente.

Hacker: Persona que modifica fuertemente el *software* o *hardware* de un sistema computacional.

Hardware: Partes físicas y/o tangibles de un dispositivo electrónico.

Header (Archivo): Archivo cabecera de una clase constituido por código fuente. Es incluido por el compilador al procesar algún otro archivo fuente. Contiene normalmente una declaración directa de clases, métodos, variables u otros identificadores.

iCloud: Sistema de almacenamiento en la nube de la empresa *Apple Inc.* El servicio permite a los usuarios almacenar datos como música y fotos en servidores remotos para descargarlos en múltiples dispositivos *iOS* y computadores personales *Mac.*

Id (Tipo de dato): Es un tipo de dato que se utiliza cuando se desconoce el tipo de dato de un objeto dado.

IDE (Integrated Development Environment): Es un entorno de desarrollo integrado empaquetado en una aplicación que integra un conjunto de herramientas de programación. Consiste en: editor de código, compilador, depurador y un constructor de interface gráfica de usuario.

IMEI: Identidad internacional de equipo móvil, pregrabado en los móviles *GSM.*

Instruments: Herramienta incluida en *XCode* que permite analizar diferentes aspectos del comportamiento de procesos en una aplicación *iOS.*

iOS: Sistema operativo de los dispositivos móviles de la empresa *Apple.*

iPad: Dispositivo electrónico tipo *Tablet* desarrollado por la empresa *Apple.*

iPhone: Familia de teléfonos inteligentes diseñado por la compañía *Apple.*

iPod: Reproductor multimedia portátil diseñado y comercializado por la empresa *Apple.*

iTunes: Reproductor de medios desarrollado por la empresa *Apple Inc.* para reproducir, organizar y sincronizar dispositivos *iOS*. Es utilizado también con el fin de comprar música y contenidos en línea.

Jailbreak: Proceso que permite a los usuarios de dispositivos iOS ejecutar aplicaciones distintas a las alojadas en la tienda *AppStore*, es decir aplicaciones de terceros.

Lenguaje de programación C: Creado en 1972 por *Dennis M. Ritchie* como evolución al lenguaje B orientado a implementación de sistemas operativos, concretamente *Unix.*

Lenguaje de Programación C++: Diseñado a mediados de los años 80 por *Bjarne Strostrup,* con el objetivo de extender al exitoso lenguaje de programación C.

Lenguaje de Programación Objective-C: Lenguaje de programación orientado a objetos usado actualmente como lenguaje de programación en *Mac OS X* e *iOS.*

LLVM (Low Level Virtual Machine): Infraestructura para desarrollar compiladores escrita en lenguaje de programación C++, diseñada para optimizar el tiempo de compilación, enlazado y ejecución.

Mac: Nombre abreviado de los computadores *Apple Macintosh*, fabricados y comercializados por la empresa *Apple*.

OpenGL: Especificación estándar que define un lenguaje cruzado, mediante una *API* para escribir aplicaciones que producen gráficos computarizados 2D y 3D.

OS X: Sistema operativo desarrollado y comercializado por la empresa *Apple* que ha sido incluido en su gama de computadores *Macintosh*.

OS X Lion: Versión 10.7 de *Mac OS X*, sistema operativo de *Apple* para sus computadores de escritorio, portátiles y servidores.

OS X Mountain Lion: Versión 10.8 de *Mac OS X,* sistema operativo de *Apple* para sus computadores de escritorio, portátiles y servidores.

Outlet (Variable): En programación es una estructura de datos que puede cambiar de contenido a lo largo de la ejecución de un programa.

PDF: Desarrollado por la empresa *Abobe Systems,* es el acrónico en inglés de portable document format que permite almacenar documentos.

Phishing: Fraude cometido a través de internet, mediante interfaces que referencian a empresas famosas, que pretende conseguir datos confidenciales de usuarios.

Protocolo: Estándar que controla o permite la conexión, comunicación y transferencia de datos entre dos puntos finales.

Run: Consiste en ejecutar una aplicación para verificar su funcionamiento, ya sea en un simulador o en un dispositivo *iOS* físico.

Safari (Aplicación): Navegador web de código cerrado desarrollado por *Apple*. Disponible para *Mac OS X*, *iOS* y *Windows*.

Scheme: Define una colección de objetivos a construir, la configuración a usar para la construcción y una colección de pruebas a ejecutar.

SDK (Software Development Kit): Kit de desarrollo de software o conjunto de herramientas de desarrollo que permiten crear aplicaciones para un sistema concreto.

Servicio Web: Tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Simulador iOS: Aplicación incluida en el *IDE XCode* que permite simular dispositivos *iOS* de manera de ejecutar aplicaciones *iOS* en un computador *Mac*.

Smartphone: Dispositivo electrónico que funciona como teléfono móvil con características similares a las de un computador personal.

Source Control Management (SCM): Administración de cambios en la aplicación *iOS.* Generalmente identificados mediante un número o letra. Cada cambio es asociado con el tiempo en el que se lo realizó y su autor. Los cambios pueden ser comparados, restaurados y en algunos casos revocados.

SQLite: Sistema de gestión de base de datos relacional. Es un proyecto de dominio público creado por *D. Richard Hipp*.

Unicode: Estándar de codificación de caracteres para facilitar el tratamiento informático, así como la transición y visualización de textos de múltiples lenguajes.

Unit Testing: Es una forma para que el código escrito se adhiera a especificaciones de diseño y se mantenga así mientras el desarrollador lo modifica, por ejemplo, al realizar mejoras de rendimiento o corrección de errores. Ayuda a escribir aplicaciones robustas y seguras.

URL: Localizador uniforme de recursos, secuencia de caracteres o dirección que se usa para nombrar recursos de internet e indicar su localización o identificación.

UTF-8: Formato de codificación de caracteres *Unicode* e *ISO 106464* utilizando símbolos de longitud variable. Definido actualmente como una de las posibilidades de codificación reconocida por *Unicode* y lenguajes web.

XIB (Archivo): Los archivos .xib se abren en *Interface Builder* y son los encargados de definir la interface gráfica de usuario y la apariencia de una aplicación *iOS*.

XML: Son las siglas en inglés para Extensible Markup Language o lenguaje de marcas extensible, desarrollado por el *W3C*. Similar a *HTML*, da soporte a bases de datos y es útil cuando varias aplicaciones deben comunicarse entre sí o integrar información.

Workspace Window: Ventana de trabajo del *IDE XCode* donde se agrupan todos los componentes con los que el desarrollador puede interactuar para crear aplicaciones *iOS*.

BIBLIOGRAFÍA

Amazon. (2012). *Amazon SimpleDB*. Recuperado el 27 de Julio de 2012, de Amazon SimpleDB: http://aws.amazon.com/es/simpledb/

Amazon. (2012). *Autenticating Users of AWS Mobile Applications with a Token Vending Machine*. Recuperado el 29 de Abril de 2013, de Autenticating Users of AWS Mobile Applications with a Token Vending Machine: http://aws.amazon.com/articles/4611615499399490

Amazon. (2012). *Token Vending Machine for Anonymous Registration*. Recuperado el 2 de Mayo de 2013, de Token Vending Machine for Anonymous Registration: http://aws.amazon.com/code/8872061742402990

Apple. (2012). *iAd Suite*. Recuperado el 5 de Mayo de 2013, de iAd Suite: https://developer.apple.com/library/ios/#samplecode/iAdSuite/Introduction/Intro.html#//apple_ref/doc/uid/DTS40010198-Intro-DontLinkElementID_2

Apple. (2012). *Getting Started in iOS Simulator*. Recuperado el 25 de Julio de 2012, de Getting Started in iOS Simulator: http://developer.apple.com/library/ios/#documentation/IDEs/Conceptual/iOS_Simulator_Guide/GettingStartedwithiOSStimulator/GettingStartedwithiOSStimulator.html#//apple_ref/doc/uid/TP40012848-CH5-SW1

Apple. (2011). *iBooks Author*. Recuperado el 9 de Junio de 2013, de iBooks Author: http://www.apple.com/es/ibooks-author/

Sitepoint. (2012). *iOS Development Basics With XCode 4*. Recuperado el 20 de Agosto de 2012, de iOS Development Basics With XCode 4: http://www.sitepoint.com/ios-development-basics-with-xcode4/

Apple. (2012). *iAd Implementation Best Practices*. Recuperado el 13 de Mayo de 2013, de iAd Implementation Best Practices: http://developer.apple.com/library/ios/#technotes/tn2264/_index.html

Apple. (2012). *Apple Developer*. Recuperado el 3 de Marzo de 2012, de Member Center: https://developer.apple.com/membercenter/index.action

Apple. (2012). *iTunes Connect*. Recuperado el 9 de Abril de 2012, de Manage Your Apps: https://itunesconnect.apple.com

Apple. (2012). *About XCode*. Recuperado el 28 de Julio de 2012, de About XCode: https://developer.apple.com/library/mac/#documentation/ToolsLanguages/Conceptual/Xcode_User_Guide/000-About_Xcode/about.html

Apple. (2012). *What's new in XCode*. Recuperado el 25 de Julio de 2012, de What's new in XCode: https://developer.apple.com/technologies/tools/whats-new.html

Apple. (2012). *XCode*. Recuperado el 25 de Julio de 2012, de XCode: https://developer.apple.com/xcode/index.php

Apple. (2012). *iOS Developer Program*. Recuperado el 6 de Agosto de 2012, de iOS Developer Program: https://developer.apple.com/programs/ios/

Apple. (2012). *Getting Started*. Recuperado el 14 de Agosto de 2012, de Getting Started: http://developer.apple.com/library/ios/#DOCUMENTATION/iPhone/Conceptual/iPhone101/Articles/01_CreatingProject.html

Apple. (2012). *Core App Objects*. Recuperado el 22 de Agosto de 2012, de Core App Objects: http://developer.apple.com/library/ios/#DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/AppArchitecture/AppArchitecture.html

Apple. (2012). *About Info.plist keys*. Recuperado el 27 de Agosto de 2012, de About Info.plist keys: https://developer.apple.com/library/mac/#documentation/General/Reference/InfoPlistKeyReference/Introduction/Introduction.html

Apple. (2012). *About information Property List Files*. Recuperado el 27 de Agosto de 2012, de About information Property List Files: https://developer.apple.com/library/mac/#documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html#//apple_ref/doc/uid/TP40009254-SW1

Apple. (2012). *What are Frameworks?* Recuperado el 28 de Agosto de 2012, de What are Frameworks?: https://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/BPFrameworks/Concepts/WhatAreFrameworks.html#//apple_ref/doc/uid/20002303-BBCEIJFI

Apple. (2012). *UIKit Framework Reference*. Recuperado el 28 de Agosto de 2012, de UIKit Framework Reference: http://developer.apple.com/library/ios/#documentation/uikit/reference/UIKit_Framework/Introduction/Introduction.html#//apple_ref/doc/uid/TP40006955-CH1-SW2

Apple. (2012). *Foundation Framework Reference*. Recuperado el 28 de Agosto de 2012, de Foundation Framework Reference: https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/ObjC_classic/Intro/IntroFoundation.html#//apple_ref/doc/uid/20000687-BAJDAJAG

Apple. (2012). *Core Graphics Framework Reference*. Recuperado el 28 de Agosto de 2012, de Core Graphics Framework Reference: http://developer.apple.com/library/ios/#DOCUMENTATION/CoreGraphics/Reference/CoreGraphics_Framework/_index.html

Apple. (2012). *Storyboard*. Recuperado el 12 de Septiembre de 2012, de Storyboard: http://developer.apple.com/library/mac/#documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

Apple. (2012). *Core Services Layes*. Recuperado el 25 de Febrero de 2013, de Core Services Layes: http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/CoreServicesLayer/CoreServicesLayer.html#//apple_ref/doc/uid/TP40007898-CH10-SW3

Apple. (2012). *Core Location Framework Reference*. Recuperado el 25 de Febrero de 2013, de Core Location Framework Reference: http://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CoreLocation_Framework/_index.html#//apple_ref/doc/uid/TP40007123

Apple. (2012). *CLLocationManager Class Reference*. Recuperado el 25 de Febrero de 2013, de CLLocationManager Class Reference: http://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CLLocationManager_Class/CLLocationManager/CLLocationManager.html#//apple_ref/doc/uid/TP40007125

Apple. (2012). *CLRegion Class Reference*. Recuperado el 25 de Febrero de 2013, de CLRegion Class Reference: http://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CLRegion_class/Reference/Reference.html#//apple_ref/doc/uid/TP40009575

Apple. (2012). *MKMapView Class Reference*. Recuperado el 25 de Febrero de 2013, de MKMapView Class Reference: http://developer.apple.com/library/ios/#documentation/MapKit/Reference/MKMapView_Class/MKMapView/MKMapView.html#//apple_ref/doc/uid/TP40008205

Apple. (2012). *Introduction to Secure Coding Guide*. Recuperado el 18 de Marzo de 2013, de Introduction to Secure Coding Guide: https://developer.apple.com/library/mac/#documentation/security/Conceptual/SecureCodingGuide/Introduction.html

Apple. (2012). *Types of Security Vulnerabilities*. Recuperado el 18 de Marzo de 2013, de Types of Security Vulnerabilities: https://developer.apple.com/library/mac/#documentation/security/Conceptual/SecureCodingGuide/Articles/TypesSecVuln.html#//apple_ref/doc/uid/TP40002529-SW2

Apple. (2012). *Managing Contracts, Taxes, and Banking*. Recuperado el 30 de Marzo de 2013, de Managing Contracts, Taxes, and Banking: http://developer.apple.com/library/ios/#documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/5_SigningContractsandBanking/SigningContractsandBanking.html#//apple_ref/doc/uid/TP40011225-CH21-SW1

Apple. (2012). *Mac App Store Review Guidelines*. Recuperado el 10 de Abril de 2013, de Mac App Store Review Guidelines: https://developer.apple.com/appstore/mac/resources/approval/guidelines.html

EndoDigital. (2012). *Part Three: Getting Started in XCode*. Recuperado el 21 de Octubre de 2012, de Part Three: Getting Started in XCode:

http://www.endodigital.com/opengl-es-2-0-on-the-iphone/part-three-getting-started-in-xcode/

Facebook. (2012). *Developers Facebook*. Recuperado el 1 de Marzo de 2012, de Developers Facebook: https://developers.facebook.com/ios/

IDownloadBlog. (2012). *State of app security on iOS and Android*. Recuperado el 19 de Marzo de 2013, de State of app security on iOS and Android: http://www.idownloadblog.com/2012/08/20/infographic-pirates-ahoy/

InfoJobs. (2012). *Persistencia de datos en aplicaciones iOS*. Recuperado el 25 de Enero de 2013, de Persistencia de datos en aplicaciones iOS: http://infojobs.hackathome.com/materiales/recursos-tecnologia/persistencia-de-datos-en-aplicaciones-ios/

IPhone Tutorials. (2012). *Recording Audio on an iPhone with AVAudioRecorder (iOS 6)*. Recuperado el 15 de Marzo de 2013, de Recording Audio on an iPhone with AVAudioRecorder (iOS 6): http://prassan-warrior.blogspot.com/2012/11/recording-audio-on-iphone-with.html

IPhone Development Tutorial HQ. (2012). *What is iTunes Connect?* Recuperado el 28 de Marzo de 2013, de What is iTunes Connect?: http://www.iphonedevelopmenttutorialhq.com/?p=261

Joshua, N. (2011). *Mastering XCode 4: Develop & Design.* Berkeley, California, Estados Unidos: Peachpit Press.

Kitz, G. (2012). *UIDevice with Unique Identifier*. Recuperado el 18 de Abril de 2013, de UIDevice with Unique Identifier: https://github.com/gekitz/UIDevice-with-UniqueIdentifier-for-iOS-5

Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development.* Apress.

Microedition Biz. (2012). *Core Data (1) - Conceptos iniciales*. Recuperado el 30 de Enero de 2013, de Core Data (1) - Conceptos iniciales: http://www.microedition.biz/blog/?p=549

Payan, A. (2009). *Presenting, Appirater*. Recuperado el 12 de Abril de 2013, de Presenting, Appirater: http://arashpayan.com/blog/2009/09/07/presenting-appirater/
Ars Technica. (2012). *Whats the bid deal with iPhone UDIDs?* Recuperado el 17 de Abril de 2013, de Whats the bid deal with iPhone UDIDs?: http://arstechnica.com/apple/2012/09/ask-ars-whats-the-big-deal-with-iphone-udids/

Programador PHP. (2012). *Persistencia de datos en iOS y Google Web Toolkit*. Recuperado el 25 de Enero de 2013, de Persistencia de datos en iOS y Google Web Toolkit: http://www.programadorphp.org/blog/cursos/persistencia-de-datos-en-ios-y-google-web-toolkit/

ReadWrite. (2012). *How To Build Secure iOS Apps*. Recuperado el 18 de Marzo de 2013, de How To Build Secure iOS Apps: http://readwrite.com/2011/06/09/How-to-build-secure-ios-apps#awesm=~o8zlHjhSzjKZZX

Wenkt, R. (2011). *XCode 4: Developer Reference.* Indianapolis, Indiana: Wiley Publishing Inc.

DOCTOR ROMEL MACHADO CLAVIJO,
SECRETARIO DE LA FACULTAD DE CIENCIAS DE LA ADMINISTRACION
DE LA UNIVERSIDAD DEL AZUAY,
C E R T I F I C A:

Que, el Consejo de Facultad en sesión realizada el 30 de mayo de 2012 conoció la petición del señor **Fausto Javier Salazar Jara** con código 44169 que denuncia su tema de tesis denominado: **"TUTORIAL DE DESARROLLO Y DISTRIBUCIÓN DE APLICACIONES PARA DISPOSITIVOS iOS utilizando el IDE XCode"** presentado como un requisito previo a la obtención del Grado de Ingeniero de Sistemas.- El Consejo acoge el informe de la Junta Académica y aprueba la denuncia. Designa como Director del trabajo al Ing. Pablo Pintado Zumba León y como miembros del Tribunal Examinador a los Ings. Marcos Orellana Cordero y Luis Calderón Peralta. De conformidad a las disposiciones reglamentarias el peticionario deberá presentar su trabajo en un plazo no mayor a **DIECIOCHO MESES** contados a partir de la fecha de aprobación, esto es hasta el 30 de noviembre de 2013.

Cuenca, junio 4 de 2012

UNIVERSIDAD DEL AZUAY
FACULTAD DE ADMINISTRACION
SECRETARIA

UNIVERSIDAD DEL AZUAY

Cuenca, 16 de Mayo de 2012

Señor Ingeniero

Oswaldo Merchán Manzano

DECANO DE LA FACULTAD DE CIENCIAS DE LA ADMINISTRACION

Ciudad.

De mis consideraciones:

Yo, Fausto Xavier Salazar Jara con código 44169, estudiante de la escuela de Ingeniería de Sistemas de esta prestigiosa universidad, solicito a usted de la forma más comedida y por su intermedio al Honorable Consejo de Facultad, la aprobación del diseño de tesis con el tema "**Tutorial de desarrollo y distribución de aplicaciones para dispositivos iOS utilizando el IDE XCode**", previo a la obtención del título de Ingeniero de Sistemas.

Me permito sugerir el nombre del Ing. Pablo Pintado Zumba como director de tesis, puesto que he recibido su asesoramiento y cuento con su aprobación.

Agradezco de antemano la favorable acogida a la presente.

Atentamente,

Fausto Xavier Salazar Jara

Código 44169

UNIVERSIDAD DEL AZUAY

Cuenca, 16 de Mayo de 2012

Señor Ingeniero
Oswaldo Merchán Manzano
DECANO DE LA FACULTAD DE CIENCIAS DE LA ADMINISTRACION.
Ciudad.

De mis consideraciones:

Por medio de la presente, me permito informar que he procedido a revisar el Diseño de Tesis del estudiante Fausto Xavier Salazar Jara de la Escuela de Ingeniería de Sistemas y Telemática cuyo tema es "**Tutorial de desarrollo y distribución de aplicaciones para dispositivos iOS utilizando el IDE XCode**", el mismo que cumple con todos los requisitos metodológicos y técnicos requeridos, por tal virtud no tengo ningún inconveniente en dirigir la mencionada Tesis.

Por las consideraciones anotadas me permito, salvo mejor criterio, recomendar su aprobación.

Atentamente,

Ing. Pablo Pintado Zumba
Profesor

Universidad del Azuay

Facultad de Ciencias de la Administración
Escuela de Ingeniería de Sistemas

Diseño de Tesis previo a la obtención del título de Ingeniero de Sistemas

Tutorial de desarrollo y distribución de aplicaciones para dispositivos iOS utilizando el IDE XCode

Autor:

Fausto Xavier Salazar Jara

Director:

Ing. Pablo Fernando Pintado Zumba

Cuenca, Ecuador
2012

1. Tema

Tutorial de desarrollo y distribución de aplicaciones para dispositivos iOS utilizando el IDE XCode

2. Antecedentes

En la actualidad existen algunos ejemplos en línea de desarrollo de aplicaciones para dispositivos *iOS*, la mayoría de los mismos, son para usuarios con experiencia con el IDE *XCode*, están en inglés y son casi obsoletos, debido a que fueron desarrollados para versiones anteriores del mismo. Este IDE, ha ido variando en el tiempo, integrando todos sus componentes en una sola ventana en su versión más actual. Se desconoce la existencia de un tutorial para el desarrollo de una aplicación para *iOS* que abarque desde la instalación del IDE, hasta el proceso de aprobación de la aplicación el la tienda *AppStore*.

El proceso de aprobación de aplicaciones para ser comercializadas en la tienda *AppStore* se ha convertido para la mayoría de desarrolladores en una odisea, (KoldoMac, 2011) y no se dispone de un tutorial en línea sobre como se debe realizar este proceso, así como lo que se debe tomar en cuenta antes de enviar una aplicación desarrollada para su evaluación.

3. Selección y delimitación del tema

El presente trabajo de investigación tiene como objetivo la elaboración de un tutorial para el desarrollo y distribución de aplicaciones para dispositivos *iOS* utilizando el IDE XCode.

El tutorial a realizarse, detallará los pasos para el desarrollo de una aplicación para dispositivos *iOS*, abarcando la instalación del entorno de desarrollo integrado *XCode*, seguido de la creación de un nuevo proyecto, el desarrollo de interfaces de usuario, la conexión de componentes con variables y métodos, la navegación por las ventanas usando *StoryBoards*, las pruebas de las aplicaciones tanto en el simulador integrado como en dispositivos físicos, culminando en el proceso de aprobación de la aplicación en el *AppStore*, junto con las consideraciones que se deben tomar en cuenta antes de enviarla al proceso de revisión.

Al tutorial se le añadirán recomendaciones y sugerencias en un apartado adicional en donde se abordará temas como: Incentivar a que los usuarios califiquen la aplicación en *AppStore*, para popularizarla; identificar dispositivos únicos en un servidor Web para publicar versiones gratuitas de una aplicación; cómo generar mayores ganancias a través de la aplicación publicada mediante el uso de *iAds*.

4. Justificación

El desarrollo del presente trabajo se justifica en razón de que se convertirá en una herramienta de apoyo para quienes hayan decidido desarrollar aplicaciones para la plataforma iOS. El presente trabajo facilitará el proceso de desarrollo de tal manera que no hará falta ser un usuario avanzado, será suficiente contar con conocimientos básicos, ya que, en el tutorial se abordará temas que van desde la instalación del IDE hasta el proceso de aprobación de la aplicación en la tienda *AppStore*.

Otro de los motivos por cual se justifica la creación de este tutorial de desarrollo de aplicaciones para *iOS*, es el valor agregado que éste proporciona abordando temas que resultan de interés para el nuevo desarrollador, relacionadas con

estrategias para aprovechar de mejor manera las alternativas que proporciona el modelo de negocio planteado en el *AppStore*. Una vez desarrollada la aplicación y enviada para su respectiva aprobación en el *AppStore*, el lector puede referirse a este tutorial para conocer: cómo generar valor agregado adicional con sus aplicaciones desarrolladas; cómo incentivar a los usuarios a calificar su aplicación para popularizarla en el AppStore; y, cómo identificar dispositivos únicos para publicar versiones gratuitas de prueba de su aplicación y de esta forma ganar mas mercado.

5. Descripción del objeto de estudio

El tutorial de desarrollo y publicación en AppStore de aplicaciones para dispositivos *iOS* utilizando el IDE *XCode*, se diseñará con el propósito de ser una referencia útil para los lectores interesados, pudiendo ser estos: desarrolladores de software, personal docente, estudiantes, entre otros interesados en incursionar en el mundo del desarrollo de aplicaciones para dispositivos *iOS*. Este tutorial contendrá los pasos para el desarrollo de estas aplicaciones, desde la instalación del IDE, hasta la publicación de la aplicación desarrollada en el *AppStore*.

Adicionalmente, el tutorial será detallado con ejemplos desarrollados que le permitirán al lector seguir paso a paso la creación de una nueva aplicación, contará con varias aplicaciones sencillas, que integren los conocimientos adquiridos en cada parte de este tutorial.

Finalmente se adicionará al tutorial un conjunto de buenas prácticas que permitan generar valor agregado adicional mediante el uso de *iAds*, popularizar la aplicación en el AppStore, incentivando a los usuarios a calificar la misma, y,

sugerencias para el desarrollo de versiones gratuitas de prueba que permitan popularizar la aplicación.

El tutorial estará disponible tanto en material impreso para los lectores interesados, así como una versión *iBook*, lo que permitirá a los desarrolladores revisar este tutorial desde sus propios dispositivos *iOS*.

6. Objetivo general

Tutorial de desarrollo y distribución en AppStore de aplicaciones para dispositivos iOS utilizando el IDE XCode

7. Objetivos específicos

- Crear una versión iBook del tutorial
- Crear ejemplos de aplicaciones demostrativas para cada apartado del tutorial
- Crear una aplicación de recordatorios por GPS empleando el tutorial
- Crear un manual de usuario de la aplicación de recordatorios por GPS
- Generar una guía para subir aplicaciones al AppStore
- Presentar una recopilación de acontecimientos en el proceso de aprobación de una aplicación.

8. Metodología

Para la elaboración del tutorial, se considerarán las siguientes etapas:

a) Planeación del trabajo: En este punto se elaborará un programa de trabajo para establecer estimaciones de tiempo en recopilación de información, el orden de las actividades a realizar, entre otras.

b) Aplicar técnicas de investigación: Se utilizará la investigación documental, ya sea de material físico o digital y la observación directa, analizando directamente el IDE y tomando nota de los acontecimientos más importantes.

c) Análisis de la información: Después de recopilar toda la información, se organizará la misma de forma adecuada para su futura estructuración. Se analizará esta información para facilitar este manejo y ordenamiento.

d) Estructuración del tutorial: Se establecerá el diseño y presentación que se usará para elaborar el tutorial considerando la redacción, es decir manejar un vocabulario y formato claro para los lectores, tomando en cuenta que el material sea fácil de leer, consultar y estudiar.

e) Desarrollo de aplicaciones demostrativas: Se desarrollarán aplicaciones demostrativas para los capítulos en donde se referencian aplicaciones, las cuales se anexarán al tutorial.

9. Marco teórico

Un tutorial es un sistema instructivo de autoaprendizaje que pretende simular al maestro y muestra al usuario el desarrollo de un procedimiento o los pasos para realizar una determinada actividad.

Un IDE es un entorno de desarrollo integrado que provee a los programadores facilidades para el desarrollo de software. Soportan, o pueden acoplarse, para trabajar con uno o varios lenguajes de programación. Por ejemplo, algunos IDEs, mediante *pluggins* añaden soporte de varios otros lenguajes adicionales.

El IDE XCode integra: Editor de código, Diseño de interfaz con "*Interface Builder*" (Constructor de Interfaz), pruebas, y depuraciones en una simple ventana, permitiendo incluso realizar las depuraciones en nuestros propios dispositivos iOS.

XCode es el IDE de la compañía *Apple* para crear aplicaciones para dispositivos iOS como. permite desarrollar incluso aplicaciones para iMacs. pero el objetivo de esta investigación es orientado al desarrollo de aplicaciones iOS. *XCode* incluye un "*iOS Simulator*" (Simulador de iOS).

El simulador de iOS incluido en el IDE XCode permite a los desarrolladores realizar depuraciones emulando a todos los dispositivos iOS existentes con pequeñísimas limitaciones, convirtiéndose en una poderosa herramienta.

Amazon Web Services nos brinda servicios que son ideales para el desarrollo de las aplicaciones en cuestión, se utilizará el servicio Amazon Simple DB.

Amazon SimpleDB es un almacén de datos de alta disponibilidad y flexibilidad que minimiza el trabajo de administración de bases de datos. Los desarrolladores simplemente almacenan elementos de datos y los consultan mediante solicitudes de servicios Web. Esta solución se empleará con el objetivo de desarrollar una aplicación gratuita para dispositivos *iOS*, que solo pueda ser probada una vez.

La ventaja de estos servicios web es la versatilidad para acoplarse a las necesidades del usuario. Es un servicio gratuito inicialmente, hasta que se consuma más de un 1 GB, si se sobrepasa este límite, se cobra un valor mínimo por transferencia de datos y se puede incrementar el espacio requerido por los usuarios para almacenar los datos en la Amazon SimpleBD.

10. Esquema tentativo

Introducción

Capítulo 1: Herramientas a utilizar

1.1 IDE XCode

1.2 Interface Builder

1.3 Componente iOS Simulator

1.4 Servicio de Amazon: Simple DB

Capítulo 2: Tutorial de desarrollo y distribución de aplicaciones iOS utilizando el IDE XCode

2.1 Cómo instalar el IDE XCode y sus componentes

2.2 Cómo crear un nuevo proyecto en blanco

2.2.1 Análisis de plantillas prediseñadas para un proyecto

2.3 Cómo agregar ficheros a un proyecto

2.3.1 Análisis de tipos de posibles ficheros de un proyecto

2.4 Cómo crear interfaces gráficas de usuario

2.4.1 Análisis de componentes comunes

2.4.2 Rotación de interfaces y sus componentes

2.5 Asignación de variables y eventos a componentes en Controladores de Vistas y Vistas de Tabla (Controllers)

2.5.1 Asignación de variables a componentes de GUI

2.5.2 Asignación de métodos a componentes de GUI

2.5.3 Eventos comunes de vistas (Views)

2.5.4 Eventos comunes de vistas de tabla (TableViews)

2.6 Conexión de vistas con mediante StoryBoards

2.6.1 Implementación de un control de navegación

2.6.2 Implementación de un control de pestañas

2.6.1 Análisis de tipos de transiciones

2.7 Cómo realizar persistencia de datos

2.7.1 Implementación del framework CoreData

2.8 Cómo conectar una aplicación con redes sociales

2.8.1 Implementación del API de Facebook para iOS

2.8.2 Cómo añadir frameworks

2.8.2.1 Análisis de frameworks más comunes

2.8.2.2 Implementación del framework de Twitter

2.10 Cómo probar y depurar una aplicación en dispositivos físicos.

2.10.1 Registro de dispositivos en el portal de Apple Developer

2.11 Uso de archivos multimedia

2.11.1 Implementación de fotos de la galería y la cámara a nuestra aplicación

2.11.1.1 Uso del componente UIImagePickerController

2.11.2 Acceso al micrófono del dispositivo

2.12 Acoplamiento de la aplicación para que funcione en dispositivos iPad

2.13 Seguridad en aplicaciones iOS.

2.14 Cómo subir una la aplicación al AppStore

2.14.1 Preparación de la aplicación para el envío a revisión

2.14.2 Requerimientos y restricciones para evitar rechazo de aplicaciones

2.15 Cómo obtener calificaciones de usuarios para la aplicación.

2.15.1 Motivar a usuarios satisfechos a que califiquen la aplicación de forma positiva mediante AppiRater

2.16 Cómo identificar dispositivos únicos

2.16.1 Amazon Web Service: Simple DB

2.16.2 Conexión de la aplicación con los servicios web de Amazon

2.16.3 Seguridad a nivel de base de datos del servicio web

2.17 Cómo generar valor agregado a través de la aplicación

2.17.1 Implementación de iAds a la aplicación

Capítulo 3: Desarrollo de una aplicación iOS de recordatorios por GPS mediante el tutorial.

3.1 Análisis y diseño de la aplicación

3.1.1 Levantamiento de requisitos

3.1.2 Funcionalidad y ámbito de la aplicación

3.1.3 Diagramas de funcionalidad de la aplicación

3.1.4 Diseño de Base de Datos

3.1.5 Diseño de Interfaz

3.2 Desarrollo de la aplicación

3.3 Pruebas de la aplicación

3.4 Envío para aprobación en AppStore

3.5 Aplicación de sugerencias del tutorial

3.5.1 Generación de una versión gratuita

3.5.2 Implementación de iAds

3.5.3 Motivar a usuarios a calificar la aplicación mediante AppiRatter

3.6 Estadísticas de descargas y ventas de la aplicación.

Capítulo 4: Manual de usuario de la aplicación

4.1 Modulo de registro de lugares

4.2 Modulo de registro de tareas

4.3 Reportes

4.3.1 Reporte de lugares

4.3.2 Reporte de tareas asociadas a lugares

4.3.3 Reporte de tareas de acuerdo a proximidad de lugares

4.5.4 Configuraciones o preferencias del sistema

Conclusiones

Recomendaciones

Glosarios

Anexos

Bibliografía

11. Cronograma

Actividad	Mes1				Mes2				Mes3				Mes4				Mes5				Mes6				Mes7				Mes8				Mes9			
	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
Herramientas a utilizar	X																																			
Tutorial de desarrollo y distribucion iOS App	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X								
Instalacion IDE y componentes		X																																		
Creacion Nuevo Proyecto			X	X																																
Agregar Ficheros a Poryecto					X																															
Creación de GUIs						X	X	X																													
Controladore de Vistas									X	X																											
Storyboards											X	X																									
Persistencia de datos													X	X																							
Conexión con Redes Sociales															X	X																					
Probando en dispositivos físicos																	X																				
Uso de archivos multimendia																		X	X	X																	
Acoplar para iPad																					X	X															
Distribución en AppStore																							X	X	X												
Cosechando calificaciones																										X											
Identificar dispositivos unicos																											X									
Generar ganancias con iAds																												X								
Desarrollo de aplicación GPSReminds																													X	X	X	X	X			
Manual de Usuario de GPSReminds																																		X	X	X

12. Bibliografia

UNIVERSIDAD DEL AZUAY

Amazon. (2012). *Amazon Simple DB*. Recuperado el 12 de 02 de 2012, de https://aws.amazon.com/es/simpledb/

Amazon. (2012). *Amazon Web Services*. Recuperado el 12 de 02 de 2012, de http://aws.amazon.com/es/

AngerRising. (2009). *Verbos para la formulación de objetivos (tesis)*. Recuperado el 18 de 04 de 2012, de http://angerrising.wordpress.com/2009/10/05/verbos-para-la-formulacion-de-los-objetivos-tesis/

Apple Inc. (2011). *iOS Developer Library*. Recuperado el 07 de 12 de 2011, de https://developer.apple.com/library/ios/navigation/index.html

Apple Inc. (2012). *iOS Developer Center*. Recuperado el 09 de 02 de 2012, de https://developer.apple.com/devcenter/ios/index.action

Apple Inc. (2012). *XCode 4 - Apple Developer*. Recuperado el 03 de 12 de 2011, de https://developer.apple.com/xcode/

Joshua, N. (2011). *Mastering XCode 4: Develop & Desing*. Berkeley, California, Estados Unidos: Peachpit Press.

KoldoMac. (05 de 02 de 2011). *La Odisea De Publicar En El AppStore*. Recuperado el 02 de 05 de 2012, de http://koldomac.wordpress.com/2011/02/05/la-odisea-de-publicar-en-el-app-store/

Mark, D., Nutting, J., & LaMarche, J. (2011). *Beginning iOS 5 Development*. Apress.

Uribe, J. C. (01 de 11 de 2010). *Guia iPhone*. Recuperado el 02 de 05 de 2012, de Maestros del Web: http://www.google.com.ec/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCUQFjAA&url=http%3A%2F%2Fwww.maestrosdelweb.com%2Fimages%2F2010%2F12%2Fmaestrosdelweb-guia-iphone.pdf&ei=kB6iT-H3N4bTgQfhpdTRCA&usg=AFQjCNFBXp2-XlKHORQ0kwlX1n1WtGGZVQ&sig2=CBkJYIkKFEhlJ7iMZ7OZOA

Wenkt, R. *XCode 4: Developer Reference*. Indianapolis, Indiana: Wiley Publishing Inc.

Wikipedia. (23 de 04 de 2012). *Definición de Tutorial*. Recuperado el 02 de 05 de 2012, de Wikipedia: http://es.wikipedia.org/wiki/Tutorial