Universidad del Azuay

Facultad de Ciencia y Tecnología

Escuela de Ingeniería Electrónica

DISEÑO DE UNA APLICACIÓN PARA iOS DE UN TABLERO DE ARBITRAJE Y MARCADOR DEPORTIVO PARA RACQUETBALL, E IMPLEMENTACIÓN SOBRE UN IPAD

Autor:

José Daniel Alvarez Coello

Director:

Leopoldo Vázquez Rodríguez

Cuenca, Ecuador

2013

DEDICATORIA

A mis padres: Hernán Alvarez Serrano (+) y Eliana Coello Pons, quienes han sido la base fundamental de mi educación a lo largo de la vida. A mi abuela: Ketty Pons Páez, ejemplo de generosidad y entrega incondicional.

AGRADECIMIENTOS

Agradezco a la Universidad del Azuay, a sus directivos y catedráticos, por sus enseñanzas, y por su apoyo para hacer posible que mis responsabilidades académicas fueran de la mano con mis actividades y compromisos deportivos del momento. A la Escuela de Ingeniería de Sistemas y Telemática, por hacer extensivo el acceso a la licencia universitaria para el desarrollo de iOS hacia estudiantes de la Escuela de Ingeniería Electrónica. Finalmente a una mujer que admiro y respeto, Verónica Encalada, por su comprensión, motivación y apoyo incondicional en todo momento.

RESUMEN

DISEÑO DE UNA APLICACIÓN PARA iOS DE UN TABLERO DE ARBITRAJE Y MARCADOR DEPORTIVO PARA RACQUETBALL, E IMPLEMENTACIÓN SOBRE UN iPAD

Resumen

Este trabajo trata del diseño y desarrollo de una aplicación para iPad, que soluciona las informalidades, del arbitraje y de la muestra de puntajes, existentes en el Racquetball, reemplazando las papeletas de arbitraje, por un marcador remoto comandado con una tableta electrónica.

Aborda la programación orientada a objetos del lenguaje de programación Objective-C. El proceso está basado en el IDE Xcode de Mac OSX, y la aplicación funciona sobre el sistema operativo iOS de Apple.

Diferencia las capas de: modelo, vista, y controlador, responsables de cada actividad dentro del código. Finalmente, explica el proceso de sincronización con un dispositivo real.

Palabras clave

Apple, Xcode, iOS, iPad, Objective-C, Storyboard, Racquetball.

Jose Daniel Alvarez	Lcdo. Leopoldo Vásquez R.	Ing.Francisco Vásquez Calero
ESTUDIANTE	DIRECTOR	DIRECTOR DE CARRERA

ABSTRACT

ABSTRACT

DESIGN OF AN iOS APPLICATION FOR A RACQUETBALL REFEREE AND SCOREBOARD, AND ITS IMPLEMENTATION IN AN iPAD

This work deals with the design and development of an iPad application that can help solve the current informalities during umpiring and scoring in racquetball. It will replace the referee paper slips with a remote indicator controlled by an electronic tablet.

The study addresses programming focused on Objective-C programming language. The process is based on the IDE Xcode from Mac and it operates with Apple's iOS operative system.

The research also differentiates the components: model, view, and controller, which are responsible for each activity within the code. Finally, we explain the synchronizing process with an actual device.

Key words: Apple, Xcode, iOS, iPad, Objective-C, Storyboard, Racquetball.

José Daniel Álvarez
STUDENT

Lcdo. Leopoldo Vásquez R.
DIRECTOR

Ing. Francisco Vasquez Calero
SCHOOL DERECTOR

UNIVERSIDAD DEL AZUAY
DPTO. IDIOMAS

Translated by,
Diana Lee Rodas

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE SEGMENTOS DE CÓDIGO

ÍNDICE DE ANEXOS

José Daniel Alvarez Coello

Trabajo de graduación

Lcdo. Leopoldo Vázquez Rodríguez

Abril de 2013

DISEÑO DE UN APLICACIÓN PARA iOS DE UN TABLERO DE ARBITRAJE Y MARCADOR DEPORTIVO PARA RACQUETBALL, E IMPLEMENTACIÓN SOBRE UN IPAD

INTRODUCCIÓN

La tecnología ha superado grandes barreras a través de los años, su inclusión dentro de los infinitos campos a los cuales se aplican sus avances han aportado comodidad y profesionalismo. El ámbito deportivo es uno de los favorecidos por la tecnología, las nuevas herramientas permiten un desempeño más eficiente del deportista en procesos preparativos y competitivos.

Aunque algunos deportes poseen sistemas electrónicamente avanzados, los restantes (mayoría) no tienen variedad en los elementos tecnológicos de apoyo que se utilizan, y en general presentan dos falencias en común:

- Ausencia de marcadores y resultados. Lo que causa situaciones incómodas, tales como interrupción del juego, confusión y equivocaciones.
- Manejo de los parámetros del juego mediante papeletas de arbitraje, que representa una notable informalidad, y un aspecto visual nada atractivo.

Estos dos aspectos, aunque elementales, constituyen la carta de presentación del deporte hacia nuevas personas que lo miran por primera vez. Varios deportes en algunas ocasiones han usado sistemas de puntuación simples como carteles manuales rotativos de números, u otros electrónicos más completos pero a la vez complejos en los requerimientos que demandan; es decir no portables, costosos y con flexibilidad

de aplicaciones notablemente limitada. Implementar sistemas de puntuación o cualquier otra aplicación basándose en una plataforma móvil genérica existente, abre la oportunidad de crecer con un dispositivo, en este caso sería la opción de expandir el producto para aplicarlo en otros deportes, o simplemente incrementar funciones para mejorar la aplicación y venderla por versiones.

En el mercado no existe una solución para Racquetball que cubra los requerimientos citados de una manera totalmente flexible, es decir, no hay un dispositivo para puntuación electrónica que permita ser fácilmente comandado por cualquier persona de manera remota sin necesidad de mayor explicación o práctica, y que además sea fácil de almacenar y transportar hacia los escenarios en los cuales demande su utilización, añadiendo también que la plataforma del marcador serviría para cualquier deporte o indicativo que se desee con sólo algunas modificaciones en los programas, estaríamos hablando entonces no sólo de una solución a la muestra de datos, sino también de una solución a la informalidad del manejo de parámetros de arbitraje, pues el dispositivo suplantaría una simple papeleta de arbitraje, por un tablero electrónico de primer nivel con opción a crecer en funciones tanto como se desee. Los objetivos de este trabajo se enfocan en diseñar una aplicación para iOS que sirva como tablero de arbitraje y marcador deportivo electrónico para Racquetball, con el fin de suplir esas necesidades mencionadas.

CAPÍTULO 1

GENERALIDADES CONCEPTUALES

1.1. Programación orientada a objetos

La programación orientada a objetos facilita notablemente el desarrollo de aplicaciones, pues aporta la flexibilidad necesaria para que el programador plasme sus ideas iniciales en un resultado de primera línea de manera más rápida y genérica, basándose en la idea de que cada estructura de código cumple una tarea específica, por lo tanto mediante la creación de objetos de dicha estructura se toma ventaja al utilizar instrucciones ya desarrolladas para implementar su funcionalidad en conjunto con otras estructuras, permitiendo obtener resultados con mayor eficiencia.

Aunque cada lenguaje de programación orientada a objetos tiene su propia perspectiva, generalmente en ellos se distinguen tres partes: Un lenguaje orientado a objetos, herramientas de desarrollo, y una librería de objetos. En el sistema operativo de móviles iOS de la compañía *Apple Incorporated*, se utiliza este tipo de programación, y el lenguaje con el que se maneja es una extensión del estándar ANSI C, llamado Objective-C.

Este trabajo, estará enfocado sobre el lenguaje de programación Objective-C que será utilizado para el propósito de este proyecto.

Nota: Un lenguaje de programación contiene gran cantidad de funcionalidades y estructuras, por esa razón este documento no contempla la descripción del lenguaje en sí, sino su enfoque y conceptualización general. Si desea conocer más acerca del lenguaje Objective-C puede acceder en línea al material que *Apple Inc.* ofrece en la siguiente dirección virtual:

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf

Objetos dentro de la programación

Lo especial de la programación orientada a objetos es que agrupa los datos y las funciones en unidades llamadas precisamente objetos, esto permite combinarlos con estructuras de código que ya se encuentren desarrollas con el fin de formar programas completos. A diferencia de la programación convencional en la cual los elementos principales son los datos y las funciones (operaciones sobre los datos), en este tipo de programación los elementos principales son los objetos y la interacción entre los objetos.

Los objetos en programación al igual que los objetos físicos de la vida diaria, combinan estado y comportamiento, por ejemplo: una guitarra combina estado (cuantas cuerdas tiene, que tan afinada se encuentra, que tan golpeada está, etc.) y comportamiento (habilidad para generar sonidos en distintas tonalidades), entonces, la composición de cada objeto en programación sería como se muestra en la *Figura 1*.

Figura 1: Composición de un objeto en programación.

Implementación e interfaz [1]

Para hacer programas se necesita expresar en código las ideas abstractas que se conciben en la mente. Para ello sirven los elementos de un lenguaje, para facilitar la transformación de una idea en una realidad. Al interactuar con la programación nos encontramos con dos aspectos:

- La implementación, que responde a ¿Cómo funciona?
- La interfaz, que responde a ¿Qué es y para qué sirve?

---

[1] APPLE INC. (2010). *Object-Oriented Programming with Objective-C*. Pág. 9 – 11.

El lenguaje de programación ya trae implementado colecciones que ofrecen diversas funcionalidades, tomando en cuenta que no tendría sentido tratar de inventar algo que ya fue inventado por alguien más, un programador debe directamente tratar a estas colecciones como herramientas para el desarrollo y verlas desde afuera como una interfaz, es decir, nada más preguntarse acerca de ellas ¿Qué son y que hacen?

Tanto las estructuras de datos como las funciones contienen elementos de implementación que dan las siguientes características:

Para estructuras de datos

- Una estructura puede estar formada por varias estructuras.
- Información compleja puede estar compuesta de capas simples.
- Nombres de las variables dentro de una estructura no dan conflictos con nombres iguales dentro de otras estructuras.

Para funciones o métodos

- Son reutilizables y pueden ser llamadas las veces que se deseen.
- Cada función debe tener un nombre único dentro de la estructura.

¿Qué hacen las funciones incluidas en el lenguaje de programación? Esta es la única pregunta que un programador debe formularse para empezar a trabajar en el desarrollo de su aplicación.

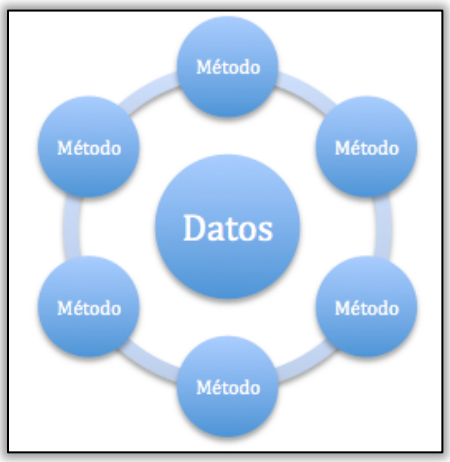
¡Las estructuras de datos NO deben ir dentro de las funciones! Pues cumplen tareas distintas. Los objetos encapsulan sus datos para que los usuarios se fijen sólo en su comportamiento, dando como resultado una interfaz más simple.

Para usar una función que pertenece a un objeto se debe pensar primero en lo que hace el objeto antes de pensar en lo que hace la función, de esa manera el programador se puede mover con facilidad dentro del lenguaje.

1.2. Perspectiva de Objective-C [2]

Como se mencionó anteriormente, el fundamento de este tipo de programación es combinar estado y comportamiento dentro de una unidad llamada objeto, su estructura se puede representar como datos encerrados por sus operaciones *(ver Figura 2)*.

Figura 2: Estructura interna de un objeto.



Fuente: APPLE INC. (2010). *Object-Oriented Programming with Objective-C*. Pág. 12.

Objetos

Son unidades de alto nivel que combinan datos y operaciones sobre datos, es decir, que contienen un grupo de funciones y a su vez datos que se utilizan en ellas. Cada objeto cumple un papel específico, y no puede realizar otra tarea diferente a la tarea para la cual fue creado. Cada objeto viene a ser entonces una instancia de alguna clase en particular.

Métodos

También conocidos como funciones, son estructuras definidas de instrucciones que sirven para realizar operaciones sobre los datos. Son el único camino de acceso hacia los datos de un objeto. Para invocar métodos de otros objetos, es necesario nombrar como receptor al objeto portador del método. Dentro de la memoria, sólo se almacena una copia de los métodos, sin importar cuantos objetos de una misma clase se creen.

---

[2] APPLE INC. (2010). *Object-Oriented Programming with Objective-C*. Pág. 12 – 23.

Programa

Es una red de objetos interconectados. Puede contener más de un objeto del mismo tipo o clase con variaciones infinitas. Un programa se puede componer de varios archivos independientes, cada archivo puede a su vez compilarse por separado. Esto hace a los programas fácilmente manejables incluso cuando son muy grandes, esta característica se la conoce como modularidad.

Clases

Cuando los objetos son de un mismo tipo, entonces nos referimos a que ellos son miembros de una misma clase, es decir que, todos los objetos que pertenecen a una misma clase tienen un mismo grupo correspondiente de variables de instancia y pueden realizar los mismos métodos. Una clase sólo puede ser definida una vez, por lo tanto, los objetos que tengan su tipo, compartirán una misma definición.

Una característica importante de las clases es la reusabilidad, con ella se logra evitar la re implementación de código, de esta manera se puede empezar desde una base ya desarrollada. Aunque las funciones también pueden ser reusables, ellas tienen restricciones por el hecho de que esconden sus datos, en cambio los objetos no presentan este problema porque sus datos están protegidos y no pueden ser alterados por otra parte del programa. Por lo tanto, lo correcto es acceder a los métodos por medio de los objetos (instancias de las clases), pues de esa manera los métodos podrán confiar que los datos con los que los invocan son compatibles.

Encapsulación

La interface encapsula la implementación, esto quiere decir que, esconde la implementación y la vuelve no accesible desde otras partes de un programa. Esta característica brinda protección frente a acciones no deseadas. En Objective-C no sólo hablamos de una encapsulación de los métodos, sino también de las variables de instancia, de esta forma estamos escondiendo la información. Contrario a lo que podría parecernos, la encapsulación brinda libertad al programador evitando que la implementación del objeto y el usuario de ese objeto estén ligados de alguna manera.

Polimorfismo

Los nombres que se asignan a variables de instancia o métodos dentro de una clase son propios de esa clase, y no producen conflictos con nombres iguales que estén dentro de otra clase. De esta manera un mismo mensaje enviado a dos objetos distintos puede invocar dos métodos diferentes. Esta característica, de que varios objetos puedan responder a un mismo mensaje, cada uno a su manera, se la conoce como Polimorfismo. Resulta muy útil porque simplifica la interface programable, y se puede usar nombres comunes para acciones comunes, sin embargo el resultado dependerá de la implementación que contenga cada clase.

Herencia

Resulta más fácil aprender algo cuando se tienen las bases anteriores, por esa razón, se incluye en el lenguaje de programación la capacidad de heredar características de un objeto hasta cierto punto genérico, es decir que, para definir un nuevo objeto se puede partir desde una definición ya existente. Cuando existe herencia, se distinguen dos tipos de clases: la superclase (clase de quién se hereda) y la subclase (clase que hereda de la superclase). El objetivo de heredar es crear algo por lo menos un poco diferente a la superclase.

Jerarquías de las clases

Lo llamativo de esta programación es que cualquier clase puede ser utilizada como superclase para crear una nueva definición. Además, dicha superclase puede ser una definición que ha heredado de otra aun más general, de esta manera se habla de una estructura ramificada que comparte su contenido sin necesidad de copiarlo. Entonces, las sucesivas subclases tendrán acceso a mayor cantidad de funcionalidades.

Capacidades de una subclase

Una subclase se puede diferenciar de su superclase cuando:

- Añade nuevos métodos y variables.
- Remplaza métodos y variables existentes.
- Modifica, perfecciona o extiende métodos y variables existentes.
- Esto hace que las subclases sean más especializadas.

Usos de la herencia

La herencia entre otras cosas es útil para:

- Reutilizar código.
- Establecer un protocolo.
- Aportar funcionalidad genérica que permita una fácil implementación.
- Hacer modificaciones pequeñas.
- Probar posibilidades de diseño en un proyecto.

1.3. Dinamismo en la programación [3]

En un principio la programación tenía muchas limitaciones de memoria, debido principalmente a que los equipos tenían también sus capacidades muy reducidas. De tal manera que la memoria de un programa no podía aumentar o disminuir. La inclusión del dinamismo en la asignación de memoria aporta notablemente al diseño de un programa, liberándolo de dichas limitaciones que han quedado en el pasado. Adicionalmente, Objective-C posee tres clases de funcionalidades dinámicas que hacen que los programas sean más fluidos, evitando otras limitaciones del mismo tipo. Estas funcionalidades son las que se resumen brevemente a continuación:

- **Escritura dinámica.-** Muestra advertencias de posibles errores en el código, pero también permite la asociación del código con objetos que se determinan en el tiempo de ejecución. Esta función da lugar al enlace dinámico.
- **Enlace dinámico.-** Consiste básicamente en esperar para decidir, durante el tiempo de ejecución de un programa, que método se debe realizar. Para este fin, no es necesario utilizar un puntero y asignarle valores, o dar un nombre diferente a cada alternativa; en Objective-C los mensajes invocan a los métodos indirectamente, pues, durante la ejecución se comprueba la clase deseada y se enlaza al método requerido dentro de esa clase.

La escritura dinámica y el enlace dinámico, permiten que un código pueda enviar mensajes a otro código que aún no ha sido inventado. De esta manera, en este lenguaje se necesita estar de acuerdo en los mensajes, más no en los tipos de datos.

- **Carga dinámica.-** Permite que diferentes partes de un programa ejecutable estén contenidas en diferentes archivos. Así, un programa puede ser usado uniendo las piezas que se necesiten. Sólo el programa principal debe ser cargado antes de iniciar, el resto se añade a la memoria volátil según se vaya necesitando, lo que significa que la memoria no se carga de programas que no se estén usando.

---

[3] APPLE INC. (2010). *Object-Oriented Programming with Objective-C*. Pág. 23 – 28.

1.4. Estructuras en un programa [4]

Se distingue una estructura tanto en la definición de un programa como en su actividad:

- La estructura dada por la jerarquía de la herencia, se refiere a cómo los objetos se relacionan por su tipo.
- La estructura dada por los mensajes (red de conexiones) entre objetos, se refiere a cómo trabaja el programa.

Conexiones mediante Outlets

Varias conexiones pueden ser temporales, aún así, algunas necesitan ser grabadas en las estructuras de datos. La manera más simple de hacerlo es que cada objeto tenga variables de instancia que sigan el rastro de otros objetos con los cuales se debe comunicar. Estas variables de instancia en particular se las conoce como *Outlets*, porque graban las salidas para los mensajes, y constituyen las conexiones principales entre los objetos dentro de un programa. El nombre de una variable Outlet debe relacionarse con el papel que los objetos Outlet desempeñan.

Outlets intrínsecos

Cuando un objeto es liberado o archivado en el disco, los objetos a los que su Outlet intrínseco apunta tienen que ser liberados o archivados con el.

Outlets extrínsecos

Capturan la organización del programa a un nivel más general. Estos Outlets graban conexiones entre programas independientes.

---

[4] APPLE INC. (2010). *Object-Oriented Programming with Objective-C*. Pág. 29 – 37.

Frameworks

Los lenguajes de programación orientada a objetos, vienen con librerías de clases, algunas de esas clases brindan funcionalidades de servicios básicos como: almacenamiento de datos, mensajería, etc. Así como otras aportan funciones más específicas como el manejo de interfaces y complementos multimedia. Una estructura parcial de un programa está definida generalmente por un grupo de estas librerías cuando se las pone a trabajar en conjunto. Estas clases pertenecientes a las librerías que incluye el lenguaje constituyen lo que se conoce como framework, que es el esqueleto de software incluido en el lenguaje que puede usarse para construir aplicaciones. Cuando se utiliza un framework, el usuario acepta el modelo de este “esqueleto” y se acopla a el.

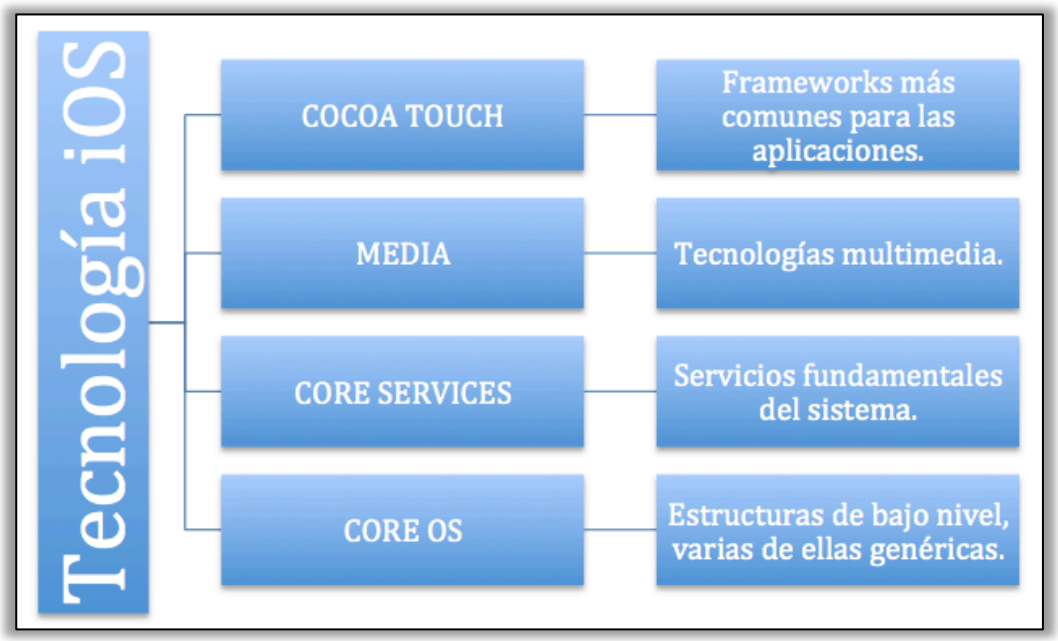
1.5. Tecnología iOS [5]

¿Qué es iOS?

iOS es la abreviatura que se utiliza para referirse al sistema operativo de los dispositivos móviles de Apple (actualmente iPhone, iPod Touch y iPad). Aunque para programar aplicaciones para iOS se necesita tener una computadora Mac de Apple, al momento de programar no se requiere tener experiencia programando aplicaciones para Mac OS X (Sistema Operativo de Mac), pues iOS posee su propio kit de desarrollo (*SDK, Software Development Kit*).

Su tecnología está dividida en cuatro capas principales, las mismas que, junto con su descripción a muy breves rasgos, se aprecian en la siguiente figura:

Figura 3: Esquema de la tecnología iOS por capas.



Lo que se busca es en lo posible utilizar las dos capas superiores para obtener resultados de calidad en poco tiempo, sin embargo, el programador posee acceso a todos los niveles de la tecnología, pudiendo utilizar todas sus herramientas a conveniencia.

---

[5] APPLE INC. (2011). *iOS Technology Overview*. Pág. 6 – 48.

El SDK de iOS

Un kit de desarrollo de software (SDK) posee todas las herramientas que se necesitan para elaborar aplicaciones sobre un determinado sistema operativo. El SDK de iOS proporciona, entre otras cosas, los siguientes recursos: frameworks, un entorno de desarrollo llamado *Xcode*, y un simulador de los dispositivos móviles llamado *iOS Simulator*.

Tipos de aplicaciones para iOS

- Aplicaciones nativas.- Son aquellas que se instalan directamente en el dispositivo móvil y se pueden usar sin necesidad de estar conectado a una red de internet.
- Aplicaciones Web.- Son las que se ejecutan por intermedio del navegador Web, están alojadas en un servidor y se transmiten a través de una red de internet.

1.5.1. Cocoa Touch

Esta es la capa más superficial de la tecnología iOS, en ella encontramos los frameworks orientados a las funcionalidades más utilizadas en una aplicación, constituye el punto inicial para desarrollar un programa, pues muchas de las veces la aplicación puede necesitar sólo de esta capa para estar completa. Entre las funcionalidades que aporta Cocoa Touch tenemos: programación gráfica de vistas mediante Storyboards, soporte de documentos, multitarea, impresión inalámbrica, encriptación de información, notificaciones remotas, notificaciones locales, reconocimiento de acciones de los dedos, compartición de archivos con iTunes, conectividad vía Bluetooth con otros dispositivos y uso de controladores de vistas estándares ya desarrollados por Apple.

Frameworks de Cocoa Touch

Las funciones antes enunciadas, son posibles gracias a todas las clases que están contenidas en los frameworks o paquetes de librerías que pertenecen a Cocoa Touch. Los frameworks que encontramos en este nivel de la tecnología iOS se los resume en la *Tabla 1*.

Tabla 1.- Frameworks presentes en la capa de Cocoa Touch.

Framework	Servicios
AddressBookUI.Framework	Interfaces visuales estándares para gestionar información de contactos.
EventKitUI.Framework	Interfaces visuales estándares para gestionar eventos relacionados con calendarios.
GameKit.Framework	Emparejamiento de dispositivos para utilizar una misma aplicación en un mismo tiempo. Desde iOS 4 se incorpora Game Center que añade funciones prácticas para gestionar el desempeño de los usuarios en un juego o una aplicación de otro tipo.
iAd.Framework	Inclusión de anuncios o publicidad dentro de la aplicación.
MapKit.Framework	Colocación de un mapa para dar direcciones o resaltar puntos de interés.
MessageUI.Framework	Controladores de vistas para introducir parámetros de un mensaje.
Twitter.Framework	Referente a la red social Twitter, envío de solicitudes de suscripción y publicación de mensajes “tweets”.
UIKit.Framework	Entre otras cosas permite usar, adaptar o crear a nivel de código lo siguiente: ▪ Manejo de aplicaciones e interfaces de usuarios. ▪ Pantallas múltiples. ▪ Multitarea. ▪ Impresión inalámbrica. ▪ Contenido de texto y contenido web. ▪ Edición de texto (cortar, copiar, y pegar) ▪ Contenido animado de interfaces de usuario. ▪ Integración con otras aplicaciones usando esquemas URL. ▪ Notificaciones. ▪ Accesibilidad para usuarios con discapacidad. ▪ Creación de archivos PDF. ▪ Gestiona la utilización de la información que viene del hardware del dispositivo como: o Datos del acelerómetro. o Cámara digital. o Librería fotográfica. o Nombre del dispositivo e información del modelo. o Información del estado de la batería. o Información del sensor de proximidad.

1.5.2. Media

En este nivel encontramos tres tecnologías que hacen posible la inclusión de variedades multimedia en las aplicaciones, estas tecnologías permiten manejar respectivamente: gráficos, audio, y video. En la actualidad, los dispositivos móviles son equi-

pados con hardware sofisticado, lo cual incrementa el rango de aprovechamiento que se le puede dar a estos equipos. A continuación se menciona el uso general de cada tecnología disponible, en cada tabla las tecnologías de más alto nivel se ubican en la parte superior.

Gráficos

Aunque lo más común es trabajar con imágenes prediseñadas usándolas a través de las diferentes vistas, iOS adiciona herramientas para manejar gráficos de manera más avanzada, ellas son:

Tabla 2.- Tecnologías para gráficos.

Tecnología para gráficos	Uso general
Core Graphics	Graficas de vectores e imágenes de 2D.
Core Animation	Vistas animadas avanzadas.
Core Image	Manejo avanzado de video e imágenes fijas.
OpenGL ES y GLKit	2D y 3D con interfaces de aceleración.
Core Text	Disposiciones sofisticadas de textos.
Image I/O	Lectura y escritura de imágenes
AssetsLibrary.framework	Acceso a la librería del usuario.

Audio

Las tecnologías utilizadas, permiten trabajar con sonidos de varios formatos a alto o bajo nivel.

Tabla 3.- Tecnologías para audio.

Tecnología para audio	Uso general
MediaPlayer.framework	Acceso a librería de iTunes, reproducción de pistas y listas.
AVFoundation.framework	Interfaces para reproducir y grabar audio.
OpenAL	Interfaces multiplataforma de audio.
Core Audio	Reproducción y grabación de audio, reproducción de sonidos de alerta y otras funciones avanzadas.

Video

iOS incorpora algunas maneras de reproducir y grabar videos dependiendo de las necesidades de la aplicación.

Tabla 4.- Tecnologías para video.

Tecnología para video	Uso general
UIImagePickerController	Clase del framework UIKit para la grabación de video por medio de la cámara incorporada en el dispositivo.
MediaPlayer.framework	Reproducción de videos a pantalla completa o parcial.
AVFoundation.framework	Manejo de grabación y reproducción de videos.
Core Media	Manipulación de los datos multimedia.

Frameworks de Media

Las clases que permiten la gestión multimedia están en los siguientes paquetes:

Tabla 5.- Frameworks presentes en la capa de Media.

Framework	Servicios
AssetsLibrary.framework	Acceso a la librería multimedia del usuario.
AVFoundation.framework	Reproducción y grabación de audio y video.
CoreAudio.framework	Define los tipos de datos de audio.
AudioToolbox.framework	Reproducción, grabación y compartición de audio.
AudioUnit.framework	Acceso al hardware de audio.
CoreGraphics.framework	Dibujar gráficas en 2D.
CoreImage.framework	Manejo de video e imágenes fijas. Corrección de fotos o detección de rostros.
CoreMIDI.framework	Comunicación con dispositivos MIDI como sintetizadores.
CoreText.framework	Procesamiento de palabras.
ImageIO.framework	Importación y exportación de imágenes.
GLKit.framework	Simplifica la creación de una aplicación OpenGL ES 2.0.
MediaPLayer.framework	Reproducción a un alto nivel de audio y video desde la aplicación.
OpenAL.framework	Salida de audio posicional.
OpenGLES.framework	Dibujar contenido en 2D y 3D. Siempre va de la mano con las interfaces EAGL.
QuartzCore.framework	Vistas animadas avanzadas “Core Animation”.

1.5.3. Core Services

En este nivel de la tecnología se alojan los servicios fundamentales usados por todas las aplicaciones de manera directa o indirecta como: almacenamiento en iCloud, posibilidad de vender contenido o servicios desde la propia aplicación, base de datos SQLite, soporte para XML, etc.

Frameworks de Core Services

Al igual que en las capas anteriores, ésta posee frameworks que permiten el acceso a los servicios que se alojan a este nivel:

Tabla 6.- Frameworks presentes en la capa de Core Services

Framework	Servicios
Accounts.framework	Inclusión de un modelo sencillo para iniciar sesión en una cuenta de usuario.
AddressBook.framework	Acceso a los contactos guardados en el dispositivo.
CFNetwork.framework	Trabajo con protocolos de red.
CoreData.framework	Manejo del modelo de estructura de datos.
CoreFoundation.framework	Manejo de datos.
CoreLocation.framework	Involucrar la localización actual del dispositivo con la aplicación.
CoreMedia.framework	Audio y video avanzados.
CoreTelephony.framework	Acceso a la información celular del teléfono.
EventKit.framework	Acceso a eventos del calendario.
Foundation.framework	Complementos para CoreFoundation.framework.
MobileCoreServices.framework	Relacionado con los UTIs (Identificadores uniformes de tipo).
NewsstandKit.framework	Distribución del contenido de revistas o diarios.
QuickLook.framework	Vista previa del contenido de archivos.
StoreKit.framework	Vender contenido extra para la aplicación.
SystemConfiguration.framework	Configurar la red del dispositivo.

1.5.4. Core OS

Es la capa de más bajo nivel dentro de la tecnología iOS, ella contiene características que también están presentes en otras tecnologías, generalmente esta capa se usa indi-

rectamente, pero si se desea trabajar sobre seguridad o comunicación con un dispositivo externo al móvil, sus frameworks aportan con las clases necesarias para llevar a cabo esas tareas.

Frameworks de Core OS

Tabla 7.- Frameworks presentes en la capa de Core OS.

Framework	Servicios
Accelerate.framework	Realizar DSP, algebra lineal y cálculos de procesamiento de imágenes.
CoreBluetooth.framework	Interacción con dispositivos Bluetooth de bajo consumo de energía.
ExternalAccesory.framework	Manejo de hardware externo mediante Bluetooth o conectado al dock de 30 pines.
GSS.framework	Servicios de seguridad genérica.
Security.framework	Garantizar la seguridad de los datos. Manejo de certificados, llaves públicas y privadas, y políticas de seguridad.

1.6. Xcode [6]

Es una aplicación para MAC OS X que constituye el IDE (Ambiente de Desarrollo Integrado) de los dispositivos Apple. Proporciona todo el conjunto de herramientas que un desarrollador necesita para crear y mejorar sus aplicaciones. Dentro de Xcode se puede gestionar los proyectos iOS y archivos fuente, montar una interface de usuario, crear ejecutables, compilar código, ejecutar aplicaciones en el simulador, y sincronizar con los dispositivos móviles.

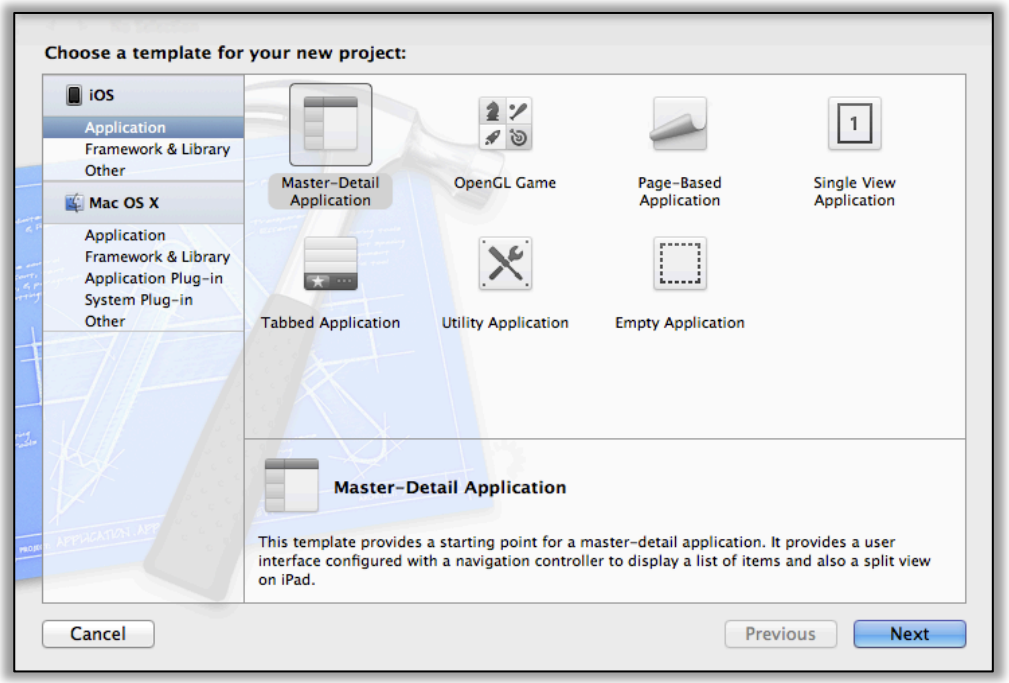
Figura 4: Ícono gráfico de Xcode.

Fuente: APPLE INC. (1999 – 2012). *Xcode 4.5.1*.

Para crear aplicaciones, es necesario empezar creando un proyecto, en Xcode se presentan varias opciones de tipos de proyectos entre los cuales se pueden seleccionar, sin embargo, se puede elegir un proyecto vacío si se lo desea, pues los tipos ofertados no son más que plantillas de trabajo que facilitan el desarrollo de código y ahorran tiempo de programación. A continuación en la *Figura 5* se muestra un ejemplo de la pantalla de selección del tipo de proyecto dentro de Xcode.

[6] APPLE INC. (2011). *iOS Technology Overview*. Pág. 58 – 61.

Figura 5: Ventana de selección de plantilla para el proyecto dentro de Xcode.



Desde la versión 4 de Xcode, el programador únicamente se concentra en la utilización de una sola ventana *(ver Figura 6)*, es decir que, todas las herramientas que brinda, están disponibles en un solo espacio, facilitando la experiencia del usuario. En ella se puede:

- Administrar todos los archivos que intervienen en el proyecto.
- Acceder fácilmente a las herramientas básicas y comandos más utilizados.
- Configurar el espacio de trabajo para mostrar u ocultar paneles de:
- Edición.
- Navegación.
- Compilación.
- Información.
- Programar gráficamente las vistas mediante Storyboards *(ver Figura 7)*.

Figura 6: Ventana de trabajo dentro de Xcode versión 4.3.2.

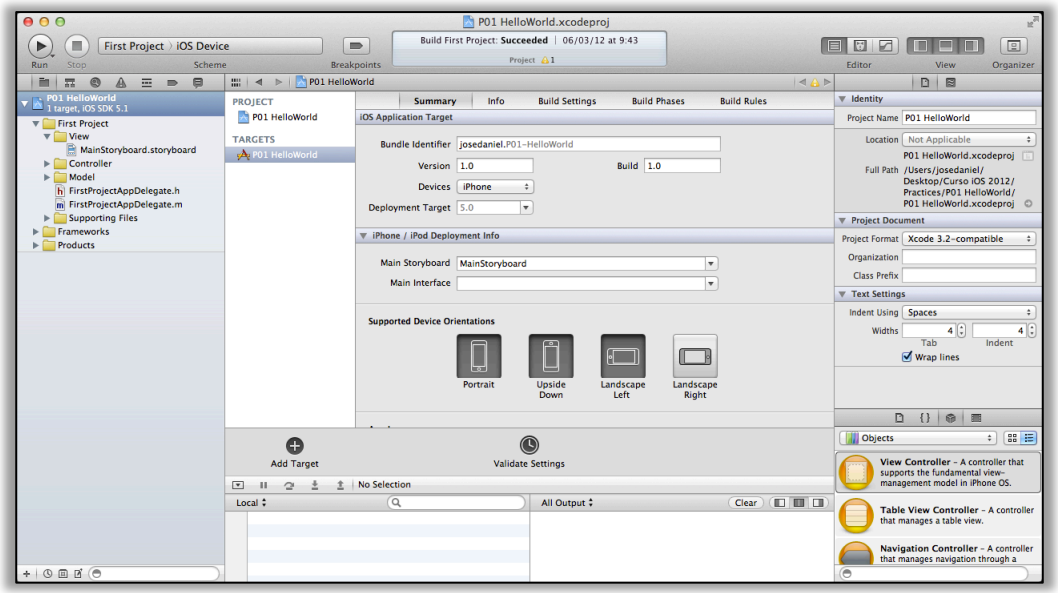
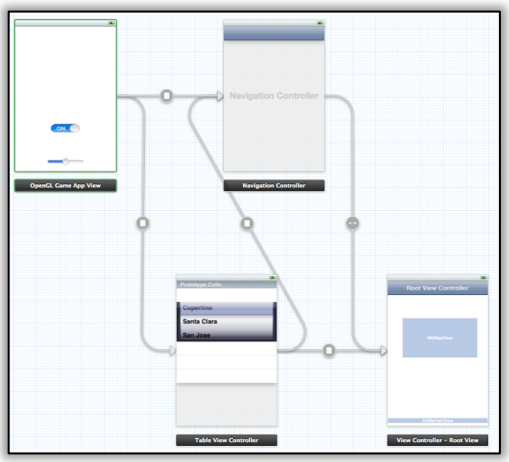
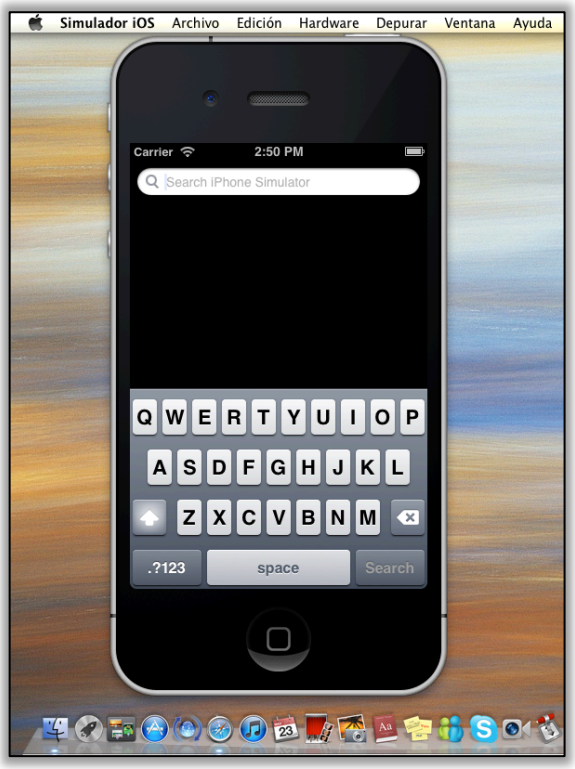


Figura 7: Ejemplo de vistas mediante Storyboards de Xcode.



Como se mencionó anteriormente, Xcode dispone de un simulador de los dispositivos móviles, el mismo que permite realizar pruebas frecuentes del avance de la aplicación sin necesidad de sincronizar con un dispositivo constantemente *(ver Figura 8)*.

Figura 8: Simulador iOS versión 5.1. Dentro de una Macintosh.



1.7. Racquetball

El Racquetball es un deporte de raqueta que se juega, con una pelota redonda de caucho, en un cuarto completamente cerrado; se puede practicarlo en las modalidades de sencillos *(Figura 9)* y dobles.

Figura 9.- Modalidad sencillos durante un partido del US Open de Racquetball.

El objetivo del juego es golpear la pelota con las cuerdas de la raqueta, de manera que ésta llegue a tocar la pared frontal sin antes haber tocado el piso, para este fin, puede utilizarse la combinación de rebotes de cualquiera de las tres paredes restantes y el techo; una vez que la bola impacta la pared frontal, el oponente debe intentar la misma acción antes de que la pelota rebote dos veces en el suelo; cada punto repetirá esto infinitas veces hasta que uno de los jugadores falle su tiro (dando al piso primero), o hasta que no llegase a responder antes de los dos botes; acorde a las reglas de la Federación Internacional de Racquetball, un partido es ganado por el jugador o equipo que consiga adjudicarse dos sets a su favor, los dos primeros sets se juegan a 15 puntos con cambios, y en caso de requerirse se juega un tercer set a 11 puntos.

CAPÍTULO 2

DISEÑO Y DESARROLLO DE LA APLICACIÓN

2.1. Esquema general

Antes de iniciar directamente con la programación, resulta muy útil trazar a breves rasgos un esquema gráfico general sobre el funcionamiento y la apariencia deseada para la aplicación. No es necesario hondar en detalles en el esquema pues únicamente representará una guía para encaminarse durante la programación.

Orientación

Se considera conveniente soportar únicamente la orientación horizontal del dispositivo, evitando cambios durante la experiencia de usuario, además para mostrar un marcador deportivo es mas agradable mostrar amplitud horizontal, sobre todo si es que se desea realizar la transmisión hacia una pantalla de un televisor o un proyector, de manera que se aproveche la mayor área posible.

Características

La aplicación tendrá básicamente las siguientes capacidades y características:

- Selección de una nueva partida, entre la modalidad sencillos (uno contra uno) y dobles (dos contra dos).
- Ingreso y almacenamiento de datos de los jugadores.
- Asignación ó sorteo del servicio inicial.
- Cambios de servidor, manejo de puntuación, asignación de tiempos fuera y asignación de apelaciones mediante gestos táctiles.
- Guardar, ver o borrar los resultados.
- Mostrar duración total del partido.
- Temporizador para mostrar el tiempo restante en un tiempo fuera.
- Compartir los resultados por correo electrónico, redes sociales o impresión inalámbrica.

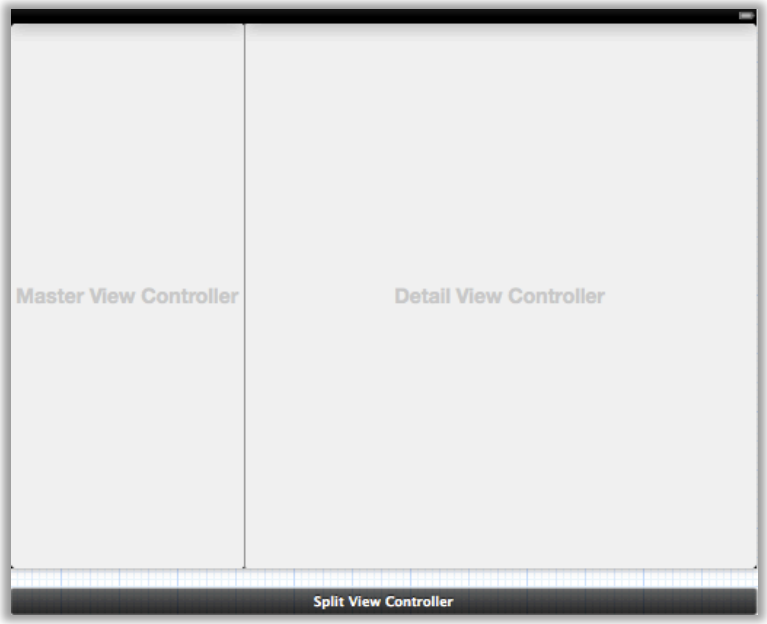
Pantalla dividida

La experiencia del usuario de un iPad se enriquece cuando se evita cambiar constantemente las vistas por completo. Las necesidades de la aplicación se acoplan bastante bien al uso de una pantalla dividida *(ver Figura 10)* mientras se encuentre fuera de la instancia del marcador propiamente, esto permitirá navegar entre listas y detalles de manera fluida aprovechando el tamaño de la pantalla.

Figura 10.- Esquema general para pantalla dividida.

Para este fin se puede usar lo que en el SDK de iOS se llama “SplitView”, que permite manejar dos vistas al mismo tiempo: una llamada “Master” que generalmente es utilizada para ubicar en ella varios objetos de un mismo tipo y otra “Detail” que suele ser controlada para mostrar los detalles del objeto seleccionado en la parte del “Master”. Esta configuración sólo está disponible para el iPad debido a que tiene una pantalla de gran tamaño, el SDK incluye además un tipo de controlador de vista llamado propiamente UISplitViewController *(ver Figura 11)*.

Figura 11.- Controlador de vista dividida en Xcode.



Puesto que la aplicación está pensada para cubrir aspectos de un juego deportivo, se plantean cuatro secciones principales dentro de la vista inicial que pueden ser manejadas por medio de pestañas de navegación: juego nuevo, jugadores, resultados y estrategias.

Juego nuevo

Contempla todo el proceso de configuración para empezar un nuevo partido: selección de modalidad entre sencillos y dobles *(ver Figura 12)*, selección de jugadores y asignación o sorteo del primer servicio.

En el lado del "Master" se puede navegar entre las tablas correspondientes a la instancia actual, mientras que se sacaría provecho la vista de detalle para mostrar en pantalla las configuraciones y selecciones que se vayan realizando *(ver Figura 13)*. Las dos figuras que se presentan a continuación son el esquema para esta sección:

Figura 12.- Esquema general para la pestaña de juego nuevo.

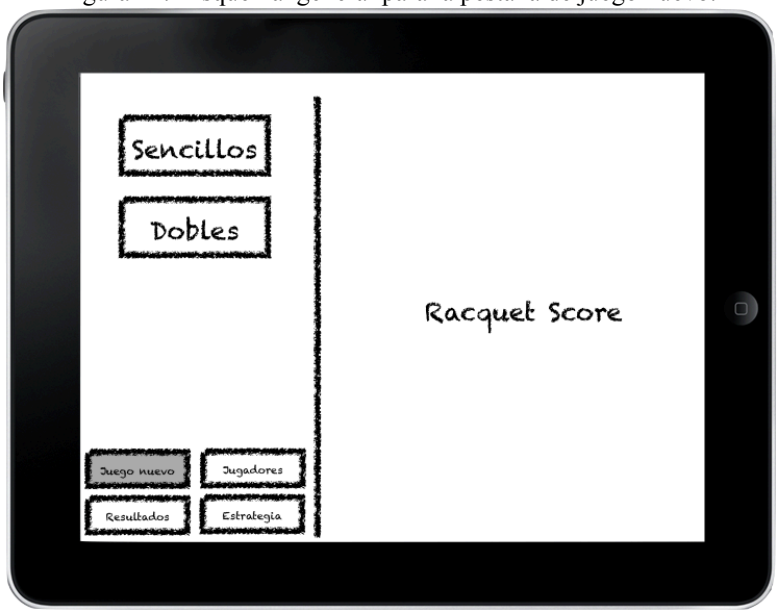
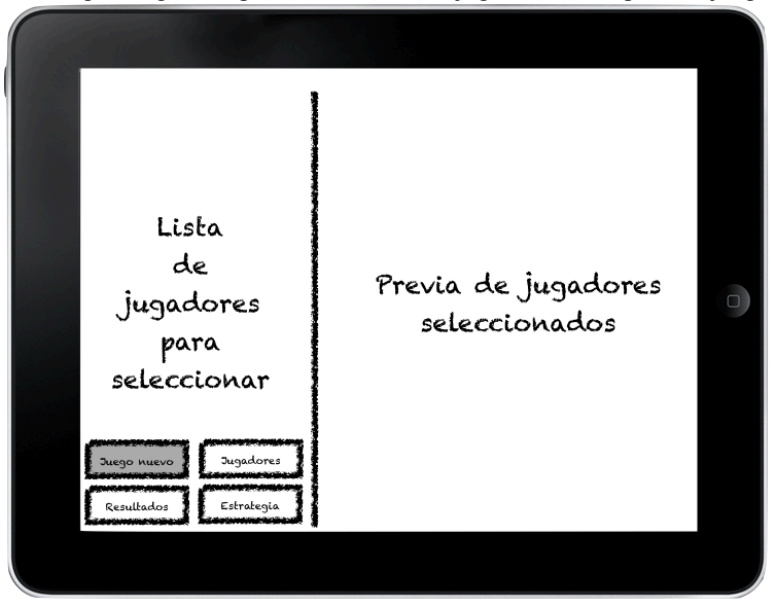


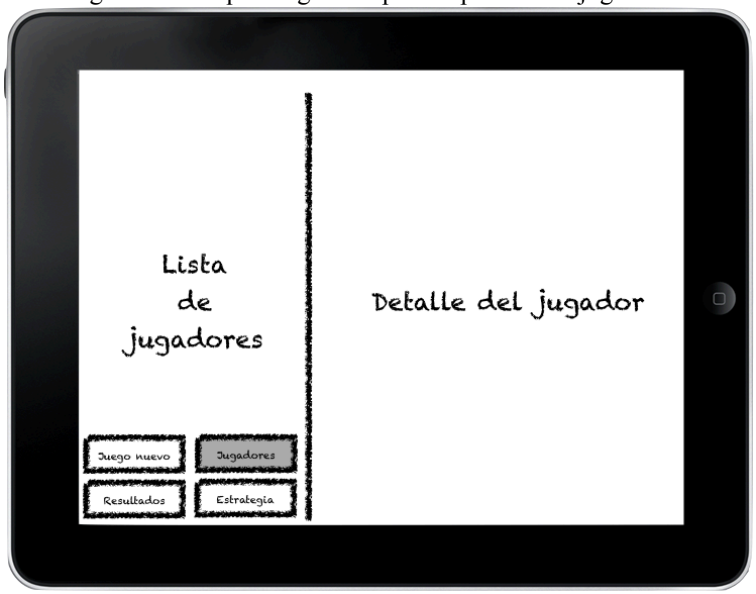
Figura 13.- Esquema general para la selección de jugadores en la pestaña juego nuevo.



Jugadores

Referente al entorno que maneja las acciones sobre los jugadores de la aplicación. Ubicándolo en la vista dividida, nos encontraríamos con algo como la *Figura 14*.

Figura 14.- Esquema general para la pestaña de jugadores.



Resultados

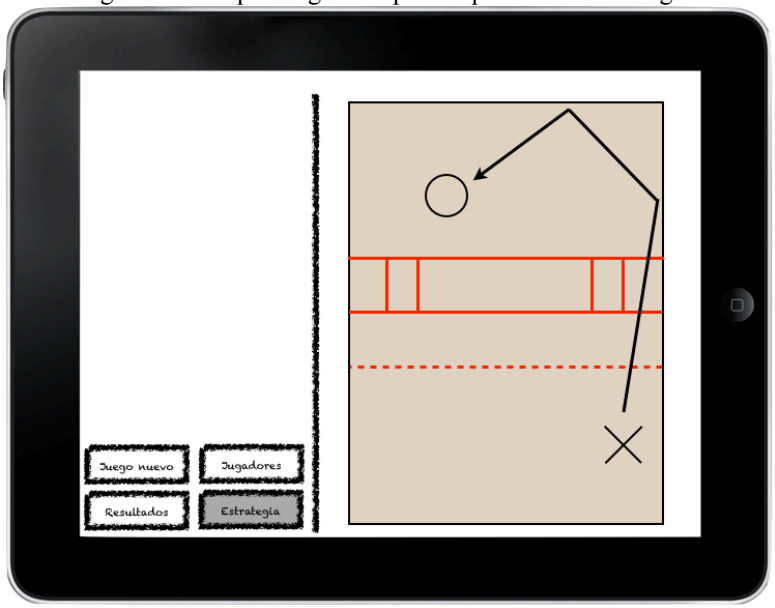
La pestaña de resultados, engloba los objetos necesarios para mostrar una tabla con los resultados almacenados en el dispositivo, y además las opciones para compartir los mismos un una red social, correo electrónico o en una impresión local inalámbrica *(ver Figura 15)*.

Figura 15.- Esquema general para la pestaña de resultados.

Estrategia

Como función adicional se contempla la incorporación de una pestaña denominada estrategia, en ese entorno la idea fundamental es que, usando un fondo de una cancha de Racquetball, se pueda dibujar sobre él estrategias de juego que sirvan para instruir a un jugador sobre un plan de acción ante su oponente. Ilustrando la idea mencionada se lo puede imaginar algo como la *Figura 16*.

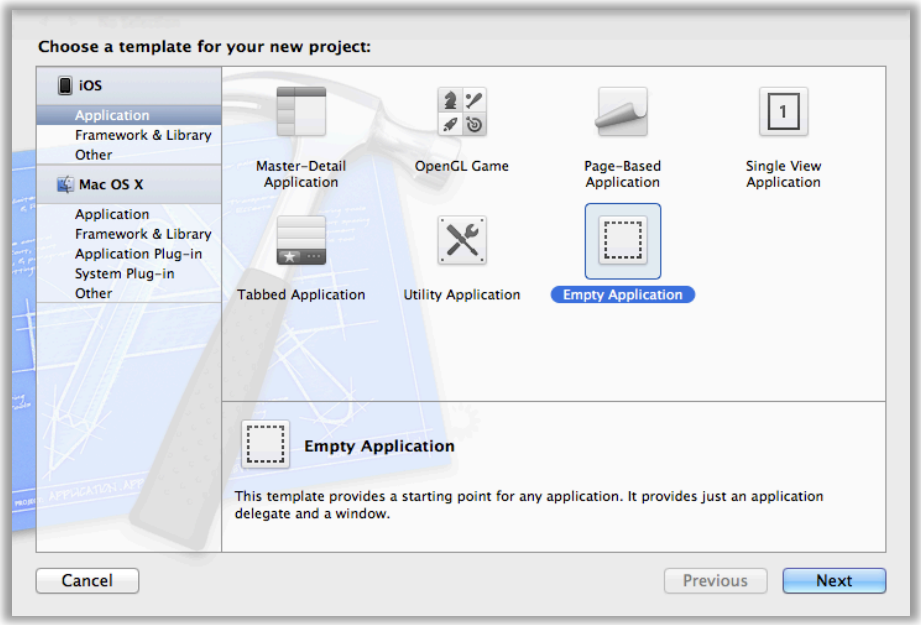
Figura 16.- Esquema general para la pestaña de estrategia.



2.2. Creación y configuración inicial del proyecto

Una vez que se tiene una idea general de lo que se busca conseguir con la aplicación, se puede empezar a trabajar propiamente en su desarrollo, así pues se da inicio a la sección de programación. Se empieza por abrir Xcode y seleccionar la creación de un nuevo proyecto desde cero, tal como se ilustra en la *Figura 17*.

Figura 17.- Creación de un nuevo proyecto en Xcode.



Se continúa con el ayudante de creación de proyectos, proporcionamos la información necesaria para construir el entorno Racquet Score *(ver Figura 18)*.

- Tipo de dispositivo sobre el cual se quiere trabajar: iPad.
- Uso de la base de datos "Core Data": No (Se usará base de datos SQLite).
- Uso de "Automatic Reference Counting" (Gestor automático de memoria): Sí.
- Uso de "Include Unit Tests": No.

Cuando se finaliza el proceso de creación, el resultado que se obtiene es una ventana de trabajo como la que se muestra en la *Figura 19*, Xcode crea para nosotros unos archivos base que se los aprecia al lado izquierdo en el navegador del proyecto.

Figura 18.- Configuración del proyecto nuevo.

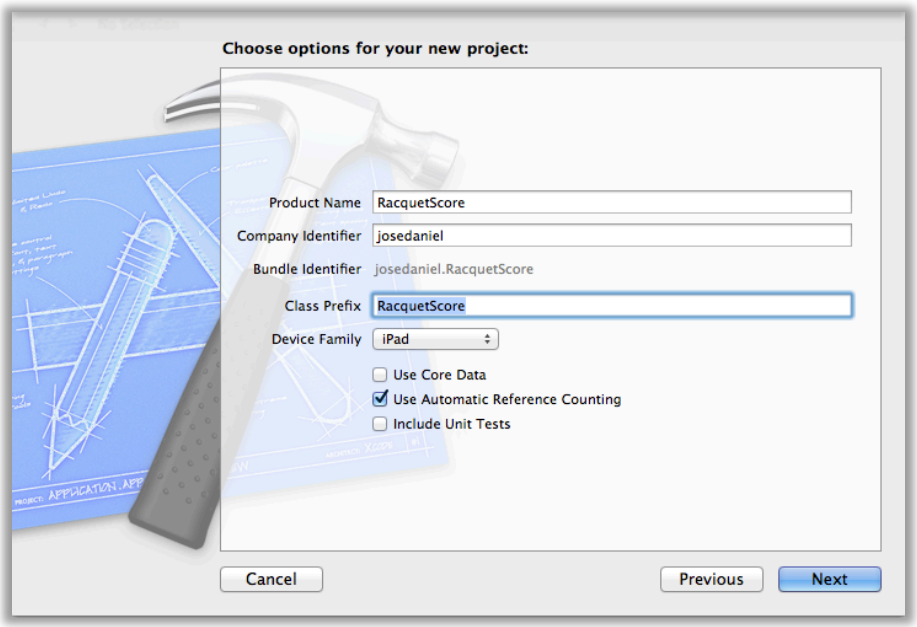
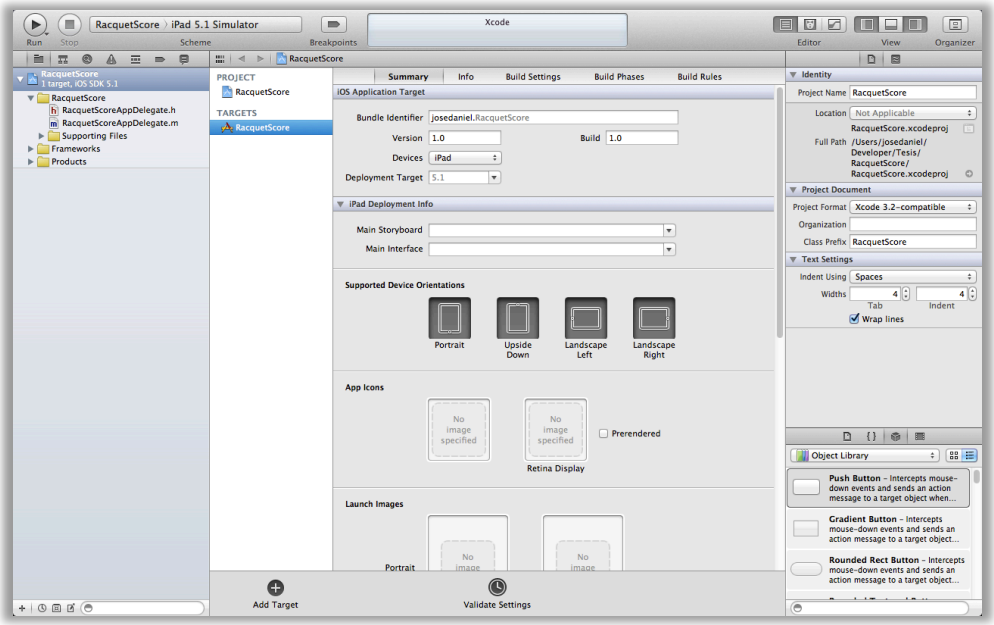


Figura 19.- Entorno de trabajo en Xcode luego de la creación del proyecto.

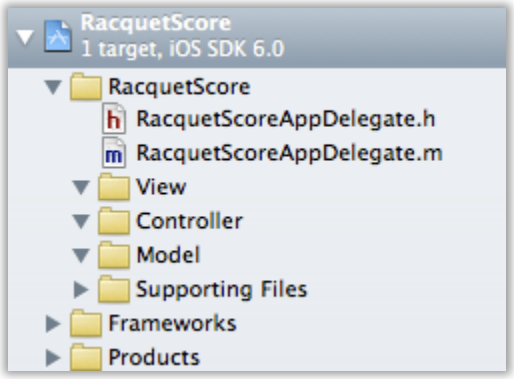


Modelo MVC en el navegador de archivos

Ya que la programación de aplicaciones iOS sigue el modelo MVC (Modelo, vista y controlador), es conveniente crear tres carpetas para ir agrupando los archivos en la capa correspondiente, estas carpetas serían: "Model", "View" y "Controller" *(ver Figura 20)*.

- Modelo MVC distinguible en el navegador de archivos es útil para ubicarse.

Figura 20.- Navegador de archivos con carpetas para distinguir modelo MVC.



Orientaciones soportadas

De acuerdo al esquema planteado, la aplicación deberá funcionar únicamente en una orientación horizontal. Para ello se hace click sobre RacquetScore en el navegador de archivos y entre las opciones de configuración que aparecen, se modifica aquella referente a la orientación *(ver Figura 21)*.

- Orientaciones horizontales para toda la aplicación.

Figura 21.- Configuración de orientación soportada.

Archivo Storyboard para la capa de Vista

Desde la versión 4 de Xcode, se incorpora lo que se denomina Storyboard, que es una opción para desarrollar la capa visual sobre un solo espacio, en donde se puede realizar la creación o modificación de manera gráfica de: controladores de vistas, vistas, barras, botones, etc. Así como también se permite interconectar objetos entre sí a manera de un diagrama de flujo. Es importante mencionar que, cuando se desea un comportamiento más personalizado o una configuración avanzada sobre ciertos objetos se la tiene que realizar mediante código. Por lo tanto, el uso de Storyboard no implica que no se puedan usar comandos sobre las vistas desde el código de la capa controladora. Entonces, la capa de vista contendrá un único archivo que manejará todas las vistas en un mismo espacio.

- Todas las vistas y objetos de interfaz de usuario se engloban en el storyboard.

Figura 22.- Creación de archivo Storyboard.

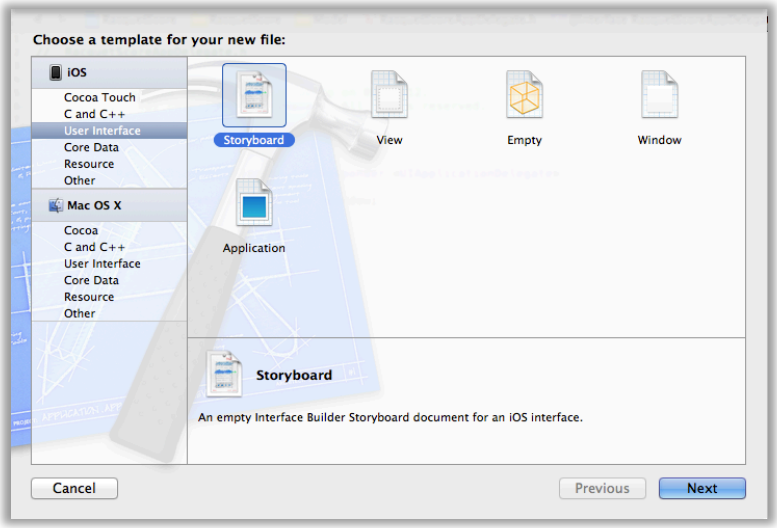
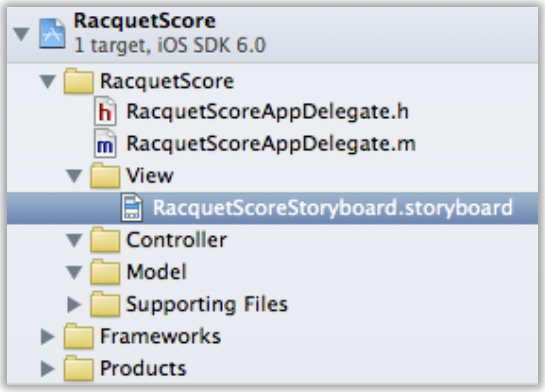
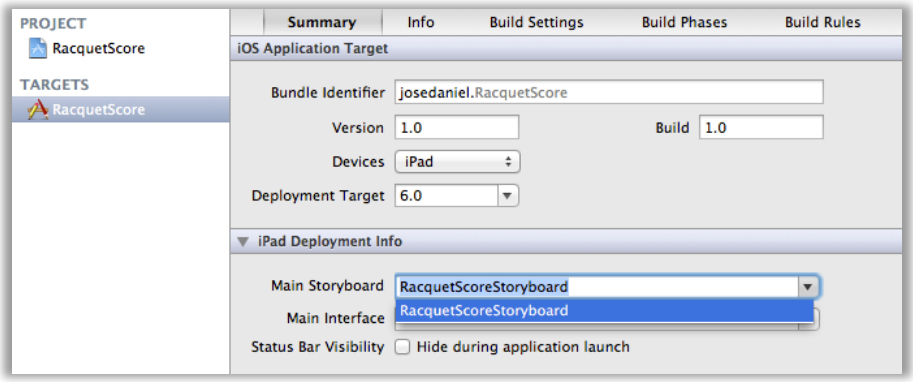


Figura 23.- Capa de Vista dentro del navegador de archivos.



- Archivo storyboard como gestor principal de la interfaz gráfica.

Figura 24.- Configuración del Storyboard principal.



Xcode añade automáticamente la aparición de la ventana en color blanco al correr el programa. Por lo tanto, se quita del archivo "RacquetScoreAppDelegate.m" las instrucciones que inicializan la ventana con un fondo blanco *(Segmento de código 1)*.

- Códigos no deseados que son autogenerados por el IDE pueden estar presentes.

Segmento de código 1.- Código autogenerado por Xcode en un proyecto vacío.

```
- (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;
}
```

Vista inicial

El esquema general contempla la utilización de la pantalla dividida para las instancias en que la aplicación no se encuentre en el marcador propiamente, la vista inicial consta en este grupo, por ello se crea la primera vista de la aplicación del tipo "Split View". En el storyboard se arrastra al espacio de trabajo un controlador de esta clase (*ver Figura 25 y Figura 26)*.

- UISplitViewController es el controlador inicial y raíz de la aplicación.

Figura 25.- Ítem "Split View Controller" añadible por medio de arrastre.

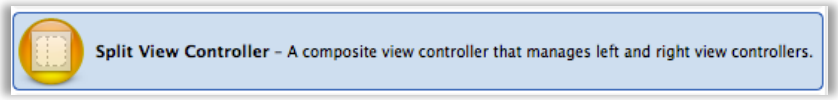
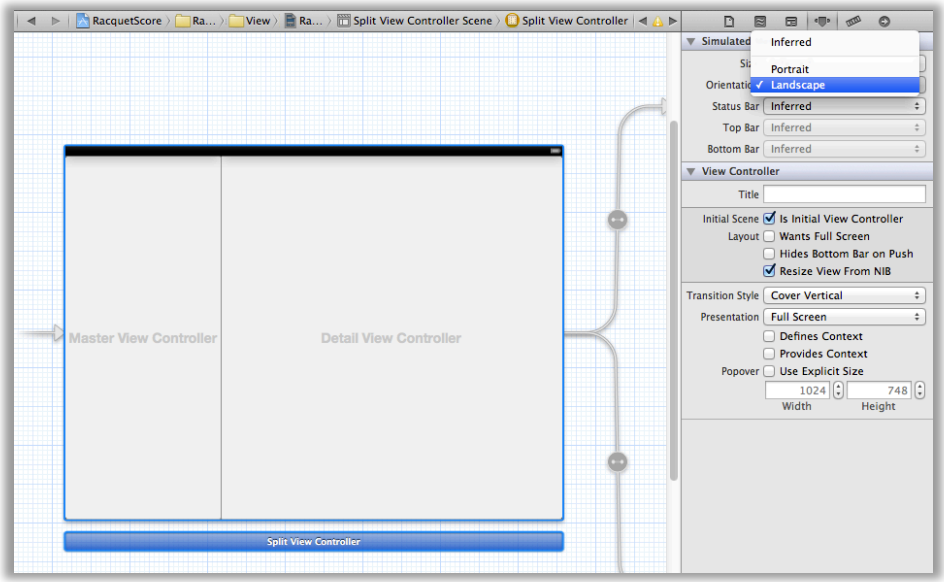


Figura 26.- "Split View Controller" para orientación horizontal.



2.3. Capa Modelo

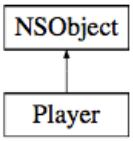
La lógica esencial del funcionamiento de la aplicación se lleva a cabo en esta capa, pues contempla las clases genéricas para definir un jugador y un resultado, el conector con la base de datos que gestionará la información y la lógica del juego para el propio marcador. A continuación se irán presentando las clases más representativas en orden cronológico junto con su descripción, herencia, propiedades, métodos y segmentos de código más relevantes.

2.3.1. Clase "Player"

Descripción

Maneja las propiedades básicas sobre la información de un jugador. Esta Clase representa la interfaz genérica para referirse a un jugador dentro del código, con ella se puede acceder a su información principal: nombre, apellido, género, país y fecha de nacimiento.

Herencia



Propiedades

- NSString ∗ name
- NSString ∗ lastName
- NSString ∗ gender
- NSString ∗ country
- NSString ∗ birthdate

Métodos de Clase

Cálculo de la edad en base a una fecha de nacimiento dada.

```
+ (NSString *)calculateAgeWithBirthdateString:(NSString *)birthdateString;
```

Métodos de Instancia

Inicializar un jugador.

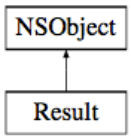
```
- (Player *) initWhitName:(NSString *)_name
                lastName:(NSString *)_lastName
                  gender:(NSString *)_gender
                 country:(NSString *)_country
            andBirthdate:(NSString *)_birthdate;
```

2.3.2. Clase "Result"

Descripción

Maneja las propiedades básicas sobre la información de un resultado de un partido. Esta Clase representa la interfaz genérica para referirse a un resultado dentro del código, con ella se puede acceder a su información principal: ganador, perdedor, marcador, duración del partido y fecha en la que se jugó.

Herencia



Propiedades

- NSString * winner
- NSString * wCountry
- NSString * loser
- NSString * lCountry
- NSString * score
- NSString * matchTime
- NSString * matchDate

Métodos de Instancia

Inicializar un resultado.

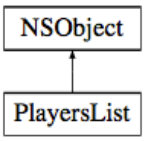
```
- (Result *) initWithWinner:(NSString *)_winner
                   wCountry:(NSString *)_wCountry
                      loser:(NSString *)_loser
                   lCountry:(NSString *)_lCountry
                      score:(NSString *)_score
                  matchTime:(NSString *)_matchTime
                  matchDate:(NSString *)_matchDate;
```

2.3.3. Clase "PlayersList"

Descripción

Maneja un arreglo correspondiente a la lista de jugadores dentro de la aplicación. Esta clase contiene los métodos para efectuar acciones sobre la lista de jugadores.

Herencia



Propiedades

- NSMutableArray * playersList

Métodos de Instancia

Añadir nuevo jugador a la lista de jugadores.

```
- (BOOL)addPlayer:(Player *)player;
```

Retirar un jugador de la lista de jugadores.

```
- (BOOL)removePlayer:(Player *)player;
```

Obtener el jugador que está en el índice dado.

```
- (Player *)playerAtIndex:(int)playerPosition;
```

Número de jugadores en la lista.

```
- (NSUInteger)count;
```

Ordenar la lista de jugadores.

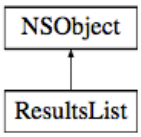
```
- (void)sortPlayersList;
```

2.3.4. Clase “ResultsList”

Descripción

Maneja un arreglo correspondiente a la lista de resultados dentro de la aplicación. Esta clase contiene los métodos para efectuar acciones sobre la lista de resultados.

Herencia



Propiedades

- NSMutableArray ∗ resultsList

Métodos de Instancia

Añadir nuevo resultado a la lista de resultados.

```
- (BOOL)addResult:(Result *)result;
```

Retirar un resultado de la lista de resultados.

```
- (BOOL)removeResult:(Result *)result;
```

Obtener el resultado que está en el índice dado.

```
- (Result *)resultAtIndex:(int)resultPosition;
```

Número de resultados en la lista.

```
- (NSUInteger)count;
```

Ordenar la lista de resultados.

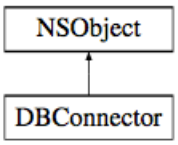
```
- (void)sortResultsList;
```

2.3.5. Clase "DBConnector"

Descripción

La clase DBConnector administra las conexiones con la base de datos para jugadores y resultados. Esta clase contiene los métodos para efectuar acciones y consultas a la base de datos sqlite3. Utiliza el framework libsqlite3.dylib necesario para el uso de instrucciones sqlite3 en Xcode.

Herencia



Métodos de Clase

Inicializar la base de datos.

```
+ (void) initDBConnector;
```

Leer todos los jugadores de la base de datos.

```
+ (PlayersList *) readAllPlayers;
```

Grabar un nuevo jugador en la base de datos.

```
+ (void)savePlayer:(Player *)player
         withPhoto:(UIImage *)photo;
```

Obtener identificación del jugador.

```
+ (int)getIdOfPlayer:(Player *)player;
```

Borrar un jugador de la base de datos.

```
+ (void)deletePlayerWithId:(int)playerId;
```

Obtener la fotografía de un jugador.

```
+ (UIImage *)getImageOfPlayerWithId:(int)playerId;
```

Obtener información adicional sobre el jugador.

```
+ (NSString *)getExtraInfoOfPlayerWithId:(int)playerId;
```

Editar fotografía e información adicional.

```
+ (void)editPhoto:(UIImage *)photo
     andExtraInfo:(NSString *)extraInfo
```

Leer todos los resultados de la base de datos.

```
+ (ResultsList *) readAllResults;
```

Grabar un nuevo resultado en la base de datos.

```
+ (void)saveResult:(Result *)result;
```

Borrar un resultado de la base de datos.

```
+ (void)deleteResult:(Result *)result;
```

Obtener cadena de resultados en formato sencillo para Facebook.

```
+ (NSString *)getFacebookStringToShare;
```

Código relevante

Creación de base de datos.

Segmento de código 2.- Creación de la base de datos.

```
// Establecer el nombre de la base de datos.
[self setDataBaseName];

// Obtener el directorio de documentos
NSArray *dirPaths =
        NSSearchPathForDirectoriesInDomains
                        (NSDocumentDirectory, NSUserDomainMask, YES);
NSString *documentsDir = [dirPaths objectAtIndex:0];

databasePath = [documentsDir stringByAppendingPathComponent:databaseName];

if ([self isDatabaseValid])
{
  // Crear un administrador para comprobar si existe la base de datos.
  NSFileManager *fileManager = [NSFileManager defaultManager];
  if ([fileManager fileExistsAtPath:databasePath] == NO)
  {
      // DB no existe, crear una.
      const char *dbPath = [databasePath UTF8String];
      sqlite3 *database;

      // la función sqlite3_open abre o crea base de datos.
      if (sqlite3_open(dbPath, &database) == SQLITE_OK)
      {
          . . . . .
          . . . . .
          sqlite3_close(database);
      } else
      {
         [RacquetScoreBrain showAlertViewMessage:
                @"Failed to open/create database" withTittle:@"Error"];
      }

  } else
  {
      // DB ya existe, salir sin hacer nada.
      return;
  }
}
```

Creación de tablas de los jugadores.

Segmento de código 3.- Creación de tablas para jugadores.

```
// Tabla PLAYERS_BASIC
NSString *createTableSQL =
@"CREATE TABLE IF NOT EXISTS players_basic "
@"(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, last_name TEXT NOT
NULL, gender TEXT NOT NULL, country TEXT NOT NULL, birthdate TEXT NOT NULL);";
const char *createTableSQL_stmt = [createTableSQL UTF8String];

if (sqlite3_exec(database, createTableSQL_stmt, NULL, NULL, &errMsg) == SQLITE_OK)
{
    [RacquetScoreBrain showAlertViewMessage:@"PLAYER_BASIC table created"
                                withTittle:@"Success"];
} else
{
    [RacquetScoreBrain showAlertViewMessage:@"Failed to create players_basic ta-
ble."
                                withTittle:@"Error"];
}

// Tabla PLAYERS_EXTRA
createTableSQL =
@"CREATE TABLE IF NOT EXISTS players_extra "
@"(extra_info, photo BLOB NOT NULL, id INTEGER REFERENCES players_basic(id) ON
DELETE CASCADE);";
createTableSQL_stmt = [createTableSQL UTF8String];

if (sqlite3_exec(database, createTableSQL_stmt, NULL, NULL, &errMsg) == SQLITE_OK)
{
    [RacquetScoreBrain showAlertViewMessage:@"PLAYERS_EXTRA table created"
                                withTittle:@"Success"];
} else
{
    [RacquetScoreBrain showAlertViewMessage:@"Failed to create players_extra ta-
ble."
                                withTittle:@"Error"];
}
```

Creación de tabla de resultados.

Segmento de código 4.- Creación de tabla para resultados.

```
// Tabla RESULTS
createTableSQL =
@"CREATE TABLE IF NOT EXISTS results "
@"(winner_name TEXT, winner_country TEXT, loser_name TEXT, loser_country TEXT,
score TEXT, match_time, match_date TEXT);";
createTableSQL_stmt = [createTableSQL UTF8String];

if (sqlite3_exec(database, createTableSQL_stmt, NULL, NULL, &errMsg) == SQLITE_OK)
{
    [RacquetScoreBrain showAlertViewMessage:@"RESULTS table created"
                                withTittle:@"Success"];
} else
{
    [RacquetScoreBrain showAlertViewMessage:@"Failed to create results table."
                                withTittle:@"Error"];
}
```

Grabar un jugador con foto en SQLite.

Segmento de código 5.- Grabar una imagen en la base de datos.

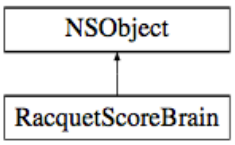
```
+ (void) savePlayer:(Player *)player withPhoto:(UIImage *)photo
{
      sqlite3 *database;
      sqlite3_stmt *statement;
      // Abrir la base de datos.
      const char *dbPath = [databasePath UTF8String];
      if(sqlite3_open(dbPath, &database) == SQLITE_OK)
      {
       NSString *insertSQL = [NSString stringWithFormat:
       @"INSERT INTO players_basic (name, last_name, gender, country, birthdate) "
       @"VALUES ('%@','%@','%@','%@','%@');",
       player.name, player.lastName, player.gender,
       player.country, player.birthdate];

       const char *insert_stmt = [insertSQL UTF8String];
       int statementResult =
              sqlite3_prepare_v2(database, insert_stmt, -1, &statement, NULL);
        if (statementResult == SQLITE_OK)
        {
            if (sqlite3_step(statement) != SQLITE_DONE)
                [RacquetScoreBrain showAlertViewMessage:
                      @"Failed to save player" withTittle:@"Error"];
        } else
        {
            [RacquetScoreBrain showAlertViewMessage:
                      @"Database can't be loaded" withTittle:@"Error"];
        }
        // Liberar de la memoria la instrucción compilada.
        sqlite3_finalize(statement);
        // Grabar en la tabla players_extra.
        insertSQL = [NSString stringWithFormat:
        @"INSERT INTO players_extra (photo, id) "
        @"VALUES (?, (SELECT MAX(id) FROM players_basic));"];
        insert_stmt = [insertSQL UTF8String];
        statementResult =
              sqlite3_prepare_v2(database, insert_stmt, -1, &statement, NULL);
        if (statementResult == SQLITE_OK)
        {
            // Grabar fotografía.
            if (photo == nil)
                photo = [UIImage imageNamed:@"unknown.png"];

            NSData *photoData = UIImagePNGRepresentation(photo);
               if ((sqlite3_bind_blob(statement, 1, [photoData bytes], [photoData
               length], NULL)) != SQLITE_OK)
                NSLog(@"No se puede grabar foto");

            if (sqlite3_step(statement) != SQLITE_DONE)
            {
                [RacquetScoreBrain showAlertViewMessage:
                      @"Failed to save photo" withTittle:@"Error"];
            }

        } else
        {
            [RacquetScoreBrain showAlertViewMessage:
                      @"Database can't be loaded" withTittle:@"Error"];
        }

        // Liberar de la memoria la instrucción compilada.
        sqlite3_finalize(statement);
    }
      sqlite3_close(database);

}
```

2.3.6. Clase “RacquetScoreBrain”

Descripción

La clase RacquetScoreBrain contiene toda la lógica del marcador, además cuenta con métodos de uso general para los controladores de las vistas. Esta clase maneja los parámetros de juego de acuerdo a las reglas; realizando las acciones sobre el puntaje, tiempos fuera, apelaciones, juegos ganados y lógica de los servicios. Incluye también funciones para realizar acciones sobre las características de objetos visuales, como: redondear las esquinas, aplicar un fondo desde un patrón, entre otros.

Herencia



Métodos de Clase

Configurar la modalidad y reglas para el marcador.

```
+ (void)configureScoreWithModality:(NSString *)_modality
                         andRules:(NSString *)_rules;
```

Obtener la modalidad actual del marcador.

```
+ (NSString *)getScoreModality;
```

Aumentar en uno (+1) un número “String”.

```
+ (NSString *)addOneToStringNumber:(NSString *)stringNumber;
```

Reducir en uno (-1) un número “String”.

```
+ (NSString *)subtractOneFromStringNumber:(NSString *)stringNumber;
```

Obtener el número de “game” actual.

```
+ (int)wichGameIsItWithGamesA:(NSString *)gamesA
                      gamesB:(NSString *)gamesB;
```

Comprobar la finalización de un “game”.

```
+ (BOOL)didWinGameWithPoints:(NSString *)points
                      gamesA:(NSString *)gamesA
                      gamesB:(NSString *)gamesB;
```

Comprobar la finalización del partido.

```
+ (BOOL)didFinishTheMatchWithGames:(NSString *)games;
```

Obtener la cantidad de segundos correspondientes para el tiempo fuera.

```
+ (int)correspondingTimeOutSecondsForActualGame:(int)actualGame;
```

Obtener la cantidad de tiempos fuera disponibles.

```
+ (NSString *)correspondingTimeOutsForGameNumber:(int)gameNumber;
```

Obtener la cantidad de apelaciones disponibles.

```
+ (NSString *)correspondingAppealsForGameNumber:(int)gameNumber;
```

Inicializar arreglo de posibles servidores.

```
+ (void)initServingPlayerArrayWithArray:(NSMutableArray *)playersArray;
```

Obtener el jugador que está sirviendo.

```
+ (Player *)whoIsServing;
```

Verificar si un jugador dado está sirviendo.

```
+ (BOOL)isThisPlayerServing:(Player *)player;
```

Asignar el servicio a un jugador dado.

```
+ (void)setThisPlayerAsServer:(Player *)player;
```

Desplazar una casilla en el arreglo de posibles servidores.

```
+ (void)displaceElementsOfServingPlayerArray;
```

Redondear las esquinas de la capa visual dada.

```
+ (void)roundedCornersForLayer:(CALayer *)layer
```

Aplicar un color de fondo especial.

```
+ (void)applySpecialBackgroundColorToImageView:(UIImageView *)imageView;
```

Obtener la imagen de la bandera de un país.

```
+ (UIImage *)getFlagImageOfCountry:(NSString *)country;
```

Escalar una imagen dada.

```
+ (UIImage *)imageWithImage:(UIImage *)image
               scaledToSize:(CGSize)newSize;
```

Convertir una imagen a escala de grises.

```
+ (UIImage *)convertImageToGrayScale:(UIImage *)image;
```

Mostrar un mensaje en una alerta visual.

```
+ (void)showAlertViewMessage:(NSString *)message
                   withTittle:(NSString *)tittle;
```

Escribir un “String” a un archivo “results.txt”.

```
+ (void)writeStringToResultsTXT:(NSString*)string;
```

Leer el archivo “results.txt”.

```
+ (NSString *)readStringFromResultsTXT;
```

Crear un PDF a partir de un “String”.

```
+ (void)writeStringToResultsPDF:(NSString *)string;
```

Código relevante

Redondear las esquinas de la imagen.

Segmento de código 6.- Redondear esquinas de una imagen.

```
// Redondea las esquinas de la capa de un objeto de la interfaz gráfica.
+ (void)roundedCornersForLayer:(CALayer *)layer
               withCornerRadius:(int)radius
                     withBorder:(BOOL)border
{
    layer.cornerRadius = radius;
    layer.masksToBounds = YES;

    if (border == YES)
    {
        layer.borderColor = [UIColor grayColor].CGColor;
        layer.borderWidth = 2;
    }
}
```

Obtener la bandera de un país.

Segmento de código 7.- Obtener bandera de país usando diccionario.

```
// Devuelve la imagen de la bandera asociada al nombre del país dado.
+ (UIImage *)getFlagImageOfCountry:(NSString *)country
{
    NSDictionary *countryCodesByName = [CountryPicker countryCodesByName];
    NSString *countryCode = [countryCodesByName objectForKey:country];
    return [UIImage imageNamed:
                      [countryCode stringByAppendingPathExtension:@"png"]];
}
```

Mostrar un mensaje de alerta en pantalla.

Segmento de código 8.- Mensaje tipo alerta en pantalla.

```
// Muestra una alerta visual en pantalla con el mensaje dado.
+ (void) showAlertViewMessage:(NSString *)message
                   withTittle:(NSString *)tittle
{
    UIAlertView *alert = [[UIAlertView alloc]
                          initWithTitle: tittle
                          message: message
                          delegate: self
                          cancelButtonTitle: @"Ok"
                          otherButtonTitles: nil];
    [alert show];
}
```

Grabar una cadena de caracteres en un archivo “TXT”.

Segmento de código 9.- Escribir un archivo "TXT".

```
// Graba una cadena de caracteres en un archivo TXT llamado "results.txt"
+ (void)writeStringToResultsTXT:(NSString*)string
{
    // Directorio al archivo.
    NSString* filePath =
        [NSSearchPathForDirectoriesInDomains
                (NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];
    NSString* fileName = @"results.txt";
    NSString* fileAtPath = [filePath stringByAppendingPathComponent:fileName];

    if (![[NSFileManager defaultManager] fileExistsAtPath:fileAtPath])
    {
        [[NSFileManager defaultManager]
               createFileAtPath:fileAtPath contents:nil attributes:nil];
    }

    // Escribir al archivo.
    [[string dataUsingEncoding:NSUTF8StringEncoding]
                                            writeToFile:fileAtPath atomically:NO];
}
```

Desplazar arreglo para cambio de servidor.

Segmento de código 10.- Desplazar elementos de un arreglo.

```
// Desplaza una vez el arreglo de posibles servidores.
+ (void) displaceElementsOfServingPlayerArray
{
    Player *aux = [playersArray objectAtIndex:0];

    for (int i = 0; i <= (playersArray.count - 2) ; ++i)
    {
        [playersArray replaceObjectAtIndex:i
                                withObject:[playersArray objectAtIndex:(i+1)]];
    }

    [playersArray replaceObjectAtIndex:(playersArray.count - 1) withObject:aux];
}
```

2.4. Capa Controladora

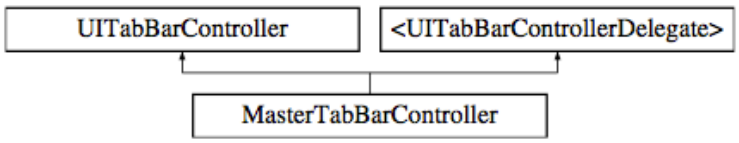
Las clases que corresponden a controladores de vistas son las que se encargan de establecer la conexión con la interfaz gráfica por medio de propiedades tipo “Outlet”. De la misma manera, aquellos métodos que son del tipo “IBAction” están enlazados con elementos de las vistas, y serán dichos objetos quienes llamen la ejecución del método luego de interactuar con ellos. A continuación se irán presentando las clases más representativas de la capa controladora en orden cronológico junto con su descripción, herencia, propiedades, y segmentos de código más relevantes.

2.4.1. Clase “MasterTabBarController”

Descripción

Controla el funcionamiento de la barra de pestañas "TabBar". La barra de pestañas para esta aplicación se encuentra incluida dentro del lado izquierdo "Master" de la vista dividida "Split view". Permite realizar acciones personalizadas sobre otras vistas que se relacionan con la pestaña que se presiona, como por ejemplo influir en el comportamiento de las vistas de detalle "Detail" de la vista dividida "Split view". Un "Split view" es una vista dividida que esta conformada por dos vistas: Una general llamada "Master" y una específica "Detail" para mostrar detalles de acuerdo a lo que se seleccione en la vista general.

Herencia



Propiedades

- NSArray ∗ modalityLabelsArray
- NSArray ∗ modalityImagesArray
- NSArray ∗ infoLabelsArray

Objeto visual relacionado

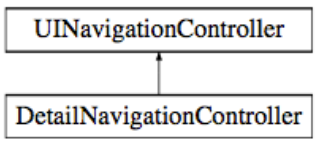
Figura 27.- Controlador de barra de pestañas.

2.4.2. Clase “DetailNavigationController”

Descripción

Controla la navegación de las ventanas de detalle del "Split view". Un "Split view" es una vista dividida que esta conformada por dos vistas: Una general llamada "Master" y una específica "Detail", este controlador se dedica a manejar esta última mencionada.

Herencia



Métodos de Clase

Establecer vista raíz como vista principal de detalle.

```
+ (void)rootDetailAsRoot;
```

Establecer vista de detalle de la partida de sencillos como vista principal.

```
+ (void)singlesDetailAsRoot;
```

Establecer vista de detalle de la partida de dobles como vista principal.

```
+ (void)doublesDetailAsRoot;
```

Establecer vista de detalle de la jugadores como vista principal.

```
+ (void)playersDetailAsRoot;
```

Establecer vista de detalle de resultados como vista principal.

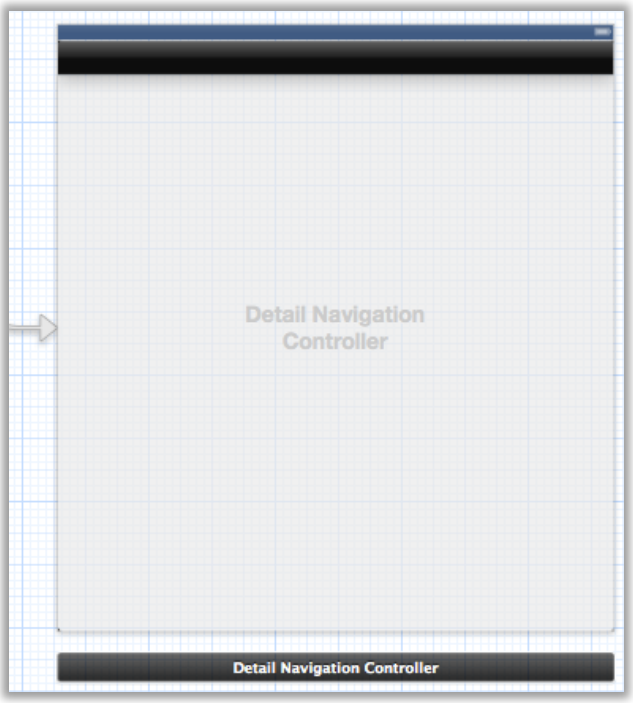
```
+ (void)resultsDetailAsRoot;
```

Establecer vista de detalle de estrategia como vista principal.

```
+ (void)strategyDetailAsRoot;
```

Objeto visual relacionado

Figura 28.- Controlador de navegación de vistas de detalle.

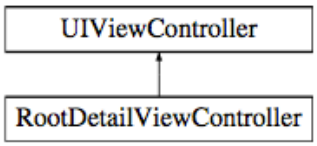


2.4.3. Clase “RootDetailViewController”

Descripción

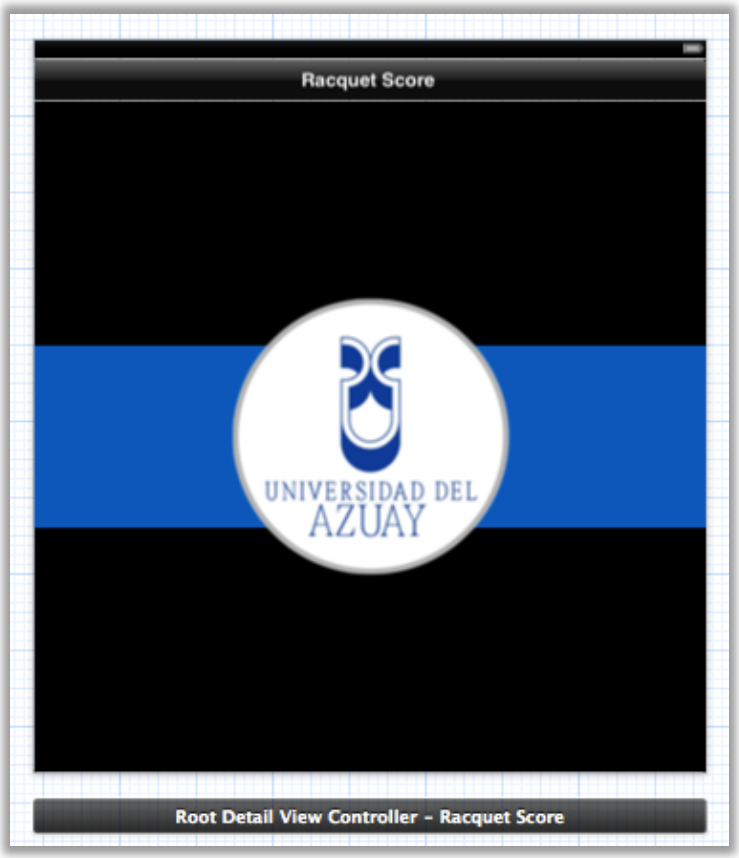
Controla la vista principal para la parte de detalle de la vista dividida "Split view". Con ella se cambia el fondo a un color especial desde un patrón de imagen.

Herencia



Objeto visual relacionado

Figura 29.- Controlador de vista RootViewController".



Código relevante

Establecer un color especial como fondo de la vista.

Segmento de código 11.- Establecer color de fondo especial.

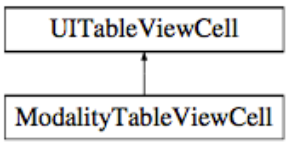
```
// Llamado luego de que la vista se carga en memoria.
- (void)viewDidLoad
{
    [super viewDidLoad];

    // Establecer el color de fondo especial a partir de un patrón.
    self.view.backgroundColor = [[UIColor alloc] initWithPatternImage:[UIImage
imageNamed:@"darkMosaic.png"]];
}
```

2.4.4. Clase "ModalityTableViewCell"

Descripción

Clase que permite comandar una celda personalizada para una tabla. Define tres propiedades tipo Outlet para comunicarse con los elementos de la capa visual.

Herencia

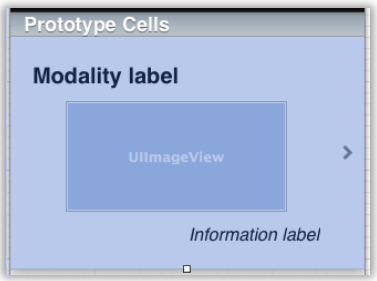


Propiedades

- IBOutlet UIImageView * **modalityImage**
- IBOutlet UILabel * **modalityLabel**
- IBOutlet UILabel * **infoLabel**

Objeto visual relacionado

Figura 30.- Celda prototipo programable para tablas.

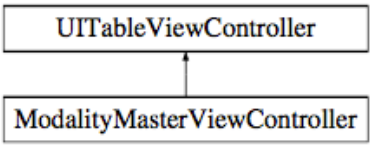


2.4.5. Clase "ModalityMasterViewController"

Descripción

Clase controladora de vista tipo tabla para selección de modalidad. Utiliza la celda personalizada del tipo ModalityTableViewCell para mostrar las opciones de las modalidades de juego para un nuevo partido.

Herencia



Propiedades

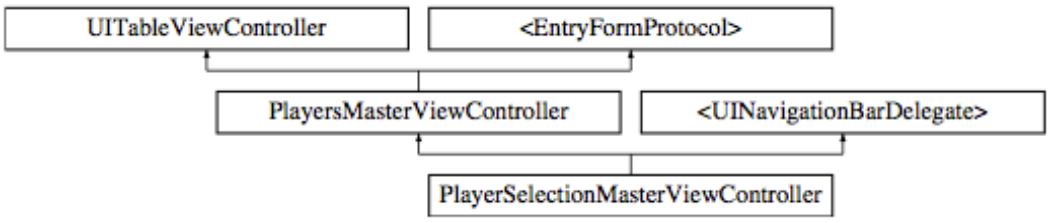
- NSArray * **modalityLabelsArray**
- NSArray * **modalityImagesArray**
- NSArray * **infoLabelsArray**

2.4.6. Clase "PlayerSelectionMasterViewController"

Descripción

Clase controladora de vista tipo tabla que maneja la información de la lista de jugadores.

Herencia



Métodos de Clase

Asignar un valor a la bandera de instancia para reutilizar el controlador.

```
+ (void) setInstanceStringWithString:(NSString *)string;
```

Código relevante

¿Qué hacer al seleccionar un jugador?

Segmento de código 12.- Instrucciones al seleccionar un jugador.

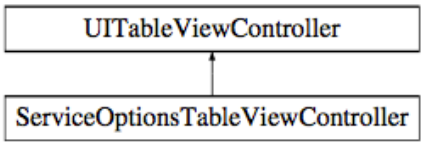
```
// Llamado cuando se selecciona una fila de la tabla.
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath
{
    // Actualizar el receptor del detalle.
    [self updateDetailReceiver];

    // Recuperar el jugador de la lista de jugadores.
    Player *selectedPlayer;
    selectedPlayer = [self.playersList playerAtIndex:indexPath.row];

    // Verificar si no se selecciona el mismo jugador mas de una vez.
    if ([self validateSelectedPlayer:selectedPlayer])
    {
        // Es válido (no repetido), Pasar los datos del jugador a otra clase.
        [self passInfoOfPlayer:selectedPlayer];
    }
    else
    {
        // No es válido (repetido)
        [RacquetScoreBrain showAlertViewMessage:@"That player is already selec-
ted." withTittle:@"Invalid selection"];
    }
}
```

2.4.7. Clase “ServiceOptionsTableViewController”

Descripción

Controlador de tabla que muestra las opciones disponibles para la asignación del primer servidor del partido.

Herencia



Propiedades

- NSArray * **serviceOptionsArray**
- NSArray * **serviceImagesArray**
- UIViewController * **detailVC**

Código relevante

Sorteo del servidor.

Segmento de código 13.- Sortear primer servidor.

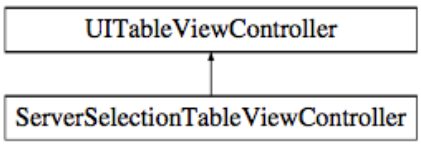
```
NSInteger coinFlipNumber = arc4random() % 2 + 1;

if ([[RacquetScoreBrain getScoreModality] isEqualToString:@"Singles"])
{
    // SENCILLOS
    [RacquetScoreBrain
          initServingPlayerArrayWithArray:
          [SinglesScoreViewController getArrayOfSinglesMatchPlayers]];

    SinglesDetailViewController *singlesDVC =
          (SinglesDetailViewController *) self.detailVC;

    if (coinFlipNumber == 1)
    {
        singlesDVC.playerA_serviceImageview.hidden = NO;
        singlesDVC.playerB_serviceImageview.hidden = YES;
    } else if (coinFlipNumber == 2)
    {
        [RacquetScoreBrain displaceElementsOfServingPlayerArray];
        singlesDVC.playerA_serviceImageview.hidden = YES;
        singlesDVC.playerB_serviceImageview.hidden = NO;
    }
} else
{
    // DOBLES
    [RacquetScoreBrain
          initServingPlayerArrayWithArray:
          [DoublesScoreViewController getArrayOfDoublesMatchPlayers]];

    DoublesDetailViewController *doublesDVC =
          (DoublesDetailViewController *) self.detailVC;

    if (coinFlipNumber == 1)
    {
        doublesDVC.playerA1_serviceImageview.hidden = NO;
        doublesDVC.playerB1_serviceImageview.hidden = YES;
    } else if (coinFlipNumber == 2)
    {
        [RacquetScoreBrain displaceElementsOfServingPlayerArray];
        [RacquetScoreBrain displaceElementsOfServingPlayerArray];
        doublesDVC.playerA1_serviceImageview.hidden = YES;
        doublesDVC.playerB1_serviceImageview.hidden = NO;
    }
}
```

2.4.8. Clase “ServerSelectionTableViewController”

Descripción

Controlador de vista tipo tabla para mostrar los posibles servidores. Muestra en una tabla los jugadores a los que se puede asignar el primer servicio del partido.

Herencia



Propiedades

- NSMutableArray ∗ **possibleServersArray**
- UIViewController ∗ **detailVC**

Código relevante

Asignación manual del primer servidor.

Segmento de código 14.- Asignación manual del servicio.

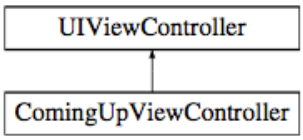
```
// Llamado cuando se selecciona una fila de la tabla.
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath
{
    // Actualizar el receptor del detalle.
    [self updateDetailReceiver];

    // Recuperar el jugador de la lista de jugadores.
    Player *selectedPlayer;
    selectedPlayer = [self.playersList playerAtIndex:indexPath.row];

    // Verificar si no se selecciona el mismo jugador mas de una vez.
    if ([self validateSelectedPlayer:selectedPlayer])
    {
        // Es válido (no repetido), Pasar los datos del jugador a otra clase.
        [self passInfoOfPlayer:selectedPlayer];
    }
    else
    {
        // No es válido (repetido)
        [RacquetScoreBrain showAlertViewMessage:
       @"That player is already selected." withTittle:@"Invalid selection"];
    }
}
```

2.4.9. Clase “ComingUpViewController”

Descripción

Controlador de la vista previa al partido, contiene un botón para dar inicio al juego. Cuando el botón se presiona se da paso a la vista del marcador. Para cuando el marcador se esconde el botón ya no está visible.

Herencia



Propiedades

- IBOutlet UIButton * **startButton**

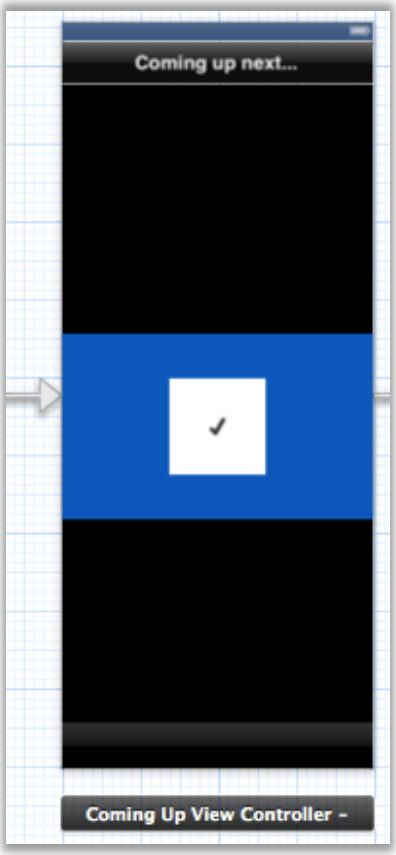
Métodos de Instancia

Iniciar partido.

```
- (IBAction)startMatchButton:(UIButton *)sender;
```

Objeto visual relacionado

Figura 31.- Vista "Master" previa al inicio del juego.

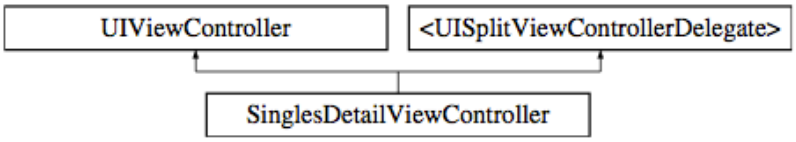


2.4.10. Clase “SinglesDetailViewController”

Descripción

Controla la vista de detalle que muestra los jugadores seleccionados en la modalidad sencillos. Contiene las propiedades tipo "Outlet" referentes a cada jugador seleccionado para mostrar la información en la pantalla. Ellas son: Nombre, fecha de nacimiento, país, género y fotografía.

Herencia



Propiedades

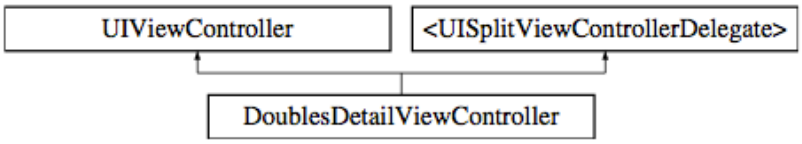
- IBOutlet UILabel * **playerA_nameLabel**
- IBOutlet UILabel * **playerA_birthdateLabel**
- IBOutlet UILabel * **playerA_countryLabel**
- IBOutlet UILabel * **playerA_genderLabel** I
- BOutlet UIImageView * **playerA_genderImageView**
- IBOutlet UIImageView * **playerA_countryFlagImageView**
- IBOutlet UIImageView * **playerA_photoImageView**
- IBOutlet UIImageView * **playerA_serviceImageview**
- IBOutlet UIImageView * **playerA_birthdateImageview**
- IBOutlet UILabel * **playerB_nameLabel**
- IBOutlet UILabel * **playerB_birthdateLabel**
- IBOutlet UILabel * **playerB_countryLabel**
- IBOutlet UILabel * **playerB_genderLabel**
- IBOutlet UIImageView * **playerB_genderImageView**
- IBOutlet UIImageView * **playerB_countryFlagImageView**
- IBOutlet UIImageView * **playerB_photoImageView**
- IBOutlet UIImageView * **playerB_serviceImageview**
- IBOutlet UIImageView * **playerB_birthdateImageview**

2.4.11. Clase “DoublesDetailViewController”

Descripción

Controla la vista de detalle que muestra los jugadores seleccionados en la modalidad dobles. Contiene las propiedades tipo "Outlet" referentes a cada jugador seleccionado para mostrar la información en la pantalla. Ellas son: Nombre, fecha de nacimiento, país, género y fotografía.

Herencia



Propiedades

- IBOutlet UILabel * **playerA1_nameLabel**
- IBOutlet UILabel * **playerA1_birthdateLabel**
- IBOutlet UILabel * **playerA1_countryLabel**
- IBOutlet UILabel * **playerA1_genderLabel**
- IBOutlet UIImageView * **playerA1_genderImageView**
- IBOutlet UIImageView * **playerA1_countryFlagImageView**
- IBOutlet UIImageView * **playerA1_photoImageView**
- IBOutlet UIImageView * **playerA1_serviceImageview**
- IBOutlet UIImageView * **playerA1_birthdateImageview**
- IBOutlet UILabel * **playerA2_nameLabel**
- IBOutlet UILabel * **playerA2_birthdateLabel**
- IBOutlet UILabel * **playerA2_countryLabel**
- IBOutlet UILabel * **playerA2_genderLabel**
- IBOutlet UIImageView * **playerA2_genderImageView**
- IBOutlet UIImageView * **playerA2_countryFlagImageView**
- IBOutlet UIImageView * **playerA2_photoImageView**
- IBOutlet UIImageView * **playerA2_birthdateImageview**
- IBOutlet UILabel * **playerB1_nameLabel**
- IBOutlet UILabel * **playerB1_birthdateLabel**
- IBOutlet UILabel * **playerB1_countryLabel**

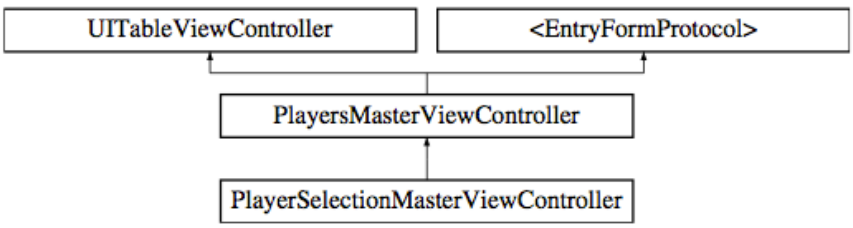
- IBOutlet UILabel * **playerB1_genderLabel**
- IBOutlet UIImageView * **playerB1_genderImageView**
- IBOutlet UIImageView * **playerB1_countryFlagImageView**
- IBOutlet UIImageView * **playerB1_photoImageView**
- IBOutlet UIImageView * **playerB1_serviceImageview**
- IBOutlet UIImageView * **playerB1_birthdateImageview**
- IBOutlet UILabel * **playerB2_nameLabel**
- IBOutlet UILabel * **playerB2_birthdateLabel**
- IBOutlet UILabel * **playerB2_countryLabel**
- IBOutlet UILabel * **playerB2_genderLabel**
- IBOutlet UIImageView * **playerB2_genderImageView**
- IBOutlet UIImageView * **playerB2_countryFlagImageView**
- IBOutlet UIImageView * **playerB2_photoImageView**
- IBOutlet UIImageView * **playerB2_birthdateImageview**

2.4.12. Clase "PlayersMasterViewController"

Descripción

Controlador de vista tipo tabla que gestiona todos los jugadores dentro de la aplicación. Incluye botones en la barra de navegación para realizar el refrescamiento de la tabla, adición de un jugador y edición.

Herencia



Propiedades

- PlayersList * **playersList**
- PlayersDetailViewController * **detailVC**

Métodos de Instancia

Refrescar la tabla de jugadores.

```
- (void)refreshPlayersTable;
```

Actualizar el receptor de detalle.

```
- (void)updateDetailReceiver;
```

Código relevante

Abrir formulario para ingreso de jugador nuevo.

Segmento de código 15.- Abrir formulario para jugador nuevo.

```
- (void)openPlayerEntryForm
{
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(refreshPlayersTable)
                                                 name:@"PossibleNewPlayer"
                                               object:nil];

    EntryFormViewController *entryForm =
      [self.storyboard instantiateViewControllerWithIdentifier:@"EntryFormID"];
    // Establecer deledado de EntryFormProtocol
    entryForm.delegate = self;
    [self presentViewController:entryForm animated:YES completion:NULL];
}
```

Borrar jugador de la tabla.

Segmento de código 16.- Borrar jugador de la tabla de jugadores.

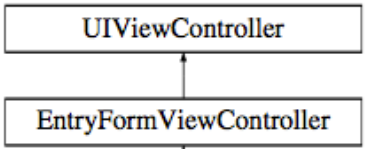
```
- (void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete)
    {
        // Borrar un jugador
        Player *selectedPlayer;
        selectedPlayer = [self.playersList playerAtIndex:indexPath.row];

        [DBConnector deletePlayerWithId:
                            [DBConnector getIdOfPlayer:selectedPlayer]];

        // Update the players list
        self.playersList = [DBConnector readAllPlayers];

        [tableView deleteRowsAtIndexPaths:@[indexPath]
                         withRowAnimation:UITableViewRowAnimationFade];
    }
}
```

2.4.13. Clase “EntryFormViewController”

Descripción

Controlador de la vista que representa el formulario de creación de un jugador nuevo. Es delegado de varias clases que permiten entre otras cosas: seleccionar país, interactuar con las alertas visuales, seleccionar imágenes guardadas, usar la cámara.

Herencia



Propiedades

- BOOL **newMedia** IBOutlet
- UINavigationBar * **navBar**
- IBOutlet UITextField * **nameTextField**
- IBOutlet UITextField * **lastNameTextField**
- IBOutlet UISegmentedControl * **genderSegmentedControl**
- IBOutlet UIDatePicker * **birthdatePicker**
- IBOutlet UIImageView * **photoImageView**

Métodos de Instancia

Ocultar el teclado al presionar la tecla “Aceptar”.

```
- (IBAction) textFieldDone:(id)sender;
```

Ocultar el teclado al presionar el fondo.

```
- (IBAction) backgroundTouched:(id)sender;
```

Usar la cámara.

```
-(IBAction)useCamera:(id)sender;
```

Usar las imágenes guardadas en el dispositivo.

```
-(IBAction)useCameraRoll:(id)sender;
```

Código relevante

Instrucción que oculta el teclado.

Segmento de código 17.- Instrucción para ocultar el teclado.

```
[sender resignFirstResponder];
```

Fuente: APPLE INC. (1999 – 2012). *Xcode 4.5.1*.
Autor: José Daniel Alvarez Coello.

Usar la cámara del dispositivo

Segmento de código 18.- Usar la cámara.

```
-(IBAction)useCamera:(UIButton *)sender
{
    if ([UIImagePickerController
              isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera])
    {
        UIImagePickerController *imagePicker =
                                        [[UIImagePickerController alloc] init];
        imagePicker.delegate = self;
        imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;
        imagePicker.mediaTypes =
                      [NSArray arrayWithObjects:(NSString *) kUTTypeImage, nil];
        [imagePicker setAllowsEditing:YES];
        [self presentViewController:imagePicker
                           animated:YES
                         completion:nil];
        newMedia = YES;
    }
}
```

Usar el rollo de la cámara

Segmento de código 19.- Usar el rollo de la cámara del dispositivo.

```
-(IBAction)useCameraRoll:(UIButton *)sender
{
    if ([popoverController isPopoverVisible])
    {
        [popoverController dismissPopoverAnimated:YES];
    } else
    {
        if ([UIImagePickerController
        isSourceTypeAvailable:UIImagePickerControllerSourceTypePhotoLibrary])
        {
            UIImagePickerController *imagePicker =
                      [[NonRotatingUIImagePickerController alloc] init];

            imagePicker.delegate = self;
            imagePicker.sourceType =
                              UIImagePickerControllerSourceTypePhotoLibrary;
            imagePicker.mediaTypes =
                      [NSArray arrayWithObjects:(NSString *) kUTTypeImage, nil];
            [imagePicker setAllowsEditing:YES];

            popoverController = [[UIPopoverController alloc]
                                    initWithContentViewController:imagePicker];

            popoverController.delegate = self;
            [popoverController presentPopoverFromRect:sender.frame
                                               inView:self.view
                             permittedArrowDirections:UIPopoverArrowDirectionRight
                                             animated:YES];
            newMedia = NO;

        }
    }
}
```

Objeto visual relacionado

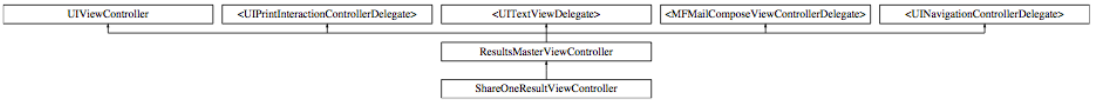
Figura 32.- Controlador del formulario para nuevo jugador.

2.4.14. Clase “ResultsMasterViewController”

Descripción

Controla una vista con botones para compartir todos los resultados. Maneja acciones de botones para compartir o imprimir todos los resultados.

Herencia



Métodos de Instancia

Imprimir.

```
- (IBAction)sharePrintButton:(UIButton *)sender;
```

Enviar resultados por correo electrónico.

```
- (IBAction)shareMailButton:(UIButton *)sender;
```

Publicar resultados en Facebook.

```
- (IBAction)shareFacebookButton:(UIButton *)sender;
```

Compartir en Twitter.

```
- (IBAction)shareTwitterButton:(UIButton *)sender;
```

Código relevante

Imprimir.

Segmento de código 20.- Impresión inalámbrica de un archivo.

```
- (IBAction)sharePrintButton:(UIButton *)sender
{
    // Directorio al archivo deseado.
    NSString* filePath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES) objectAtIndex:0];
    NSString* fileName = @"results.pdf";
    NSString* fileAtPath = [filePath stringByAppendingPathComponent:fileName];

    // Datos desde el archivo.
    NSData *resultsData = [NSData dataWithContentsOfFile:fileAtPath];

    UIPrintInteractionController *pic =
        [UIPrintInteractionController sharedPrintController];

    if(pic && [UIPrintInteractionController canPrintData:resultsData])
    {

        pic.delegate = self;

        UIPrintInfo *printInfo = [UIPrintInfo printInfo];
        printInfo.outputType = UIPrintInfoOutputGeneral;
        printInfo.jobName = [filePath lastPathComponent];
        printInfo.duplex = UIPrintInfoDuplexLongEdge;
        pic.printInfo = printInfo;
        pic.showsPageRange = YES;
        pic.printingItem = resultsData;

       void (^completionHandler)(UIPrintInteractionController *, BOOL, NSError *)
       = ^(UIPrintInteractionController *pic, BOOL completed, NSError *error)
        {
       if (!completed && error)
            {
               NSLog(@"FAILED! due to error in domain %@ with error code %u",
               error.domain, error.code);
            }
        };

        [pic presentFromRect:sender.frame
                      inView:self.view
                      animated:YES completionHandler:completionHandler];
    }
}
```

Enviar resultados por correo electrónico

Segmento de código 21.- Componer un correo electrónico con un archivo.

```
- (IBAction)shareMailButton:(UIButton *)sender
{
    if([MFMailComposeViewController canSendMail])
    {
        // Crear instancia de la clase MFMailComposeViewController.
       MFMailComposeViewController *mailVC =
              [[MFMailComposeViewController alloc]init];

        // Asignar la clase actual como la delegada.
        mailVC.mailComposeDelegate = self;

        // Configurar parámetros del correo electrónico
        [mailVC setSubject:@"Racquet Score results"];
        [mailVC setMessageBody:
              [RacquetScoreBrain readStringFromResultsTXT] isHTML:NO];

        // Adjuntar archivo PDF al correo electrónico.
        NSString* filePath =
        [NSSearchPathForDirectoriesInDomains
              (NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];
        NSString* fileName = @"results.pdf";
        NSString* fileAtPath =
              [filePath stringByAppendingPathComponent:fileName];
        NSData *fileData =
              [NSData dataWithContentsOfFile:fileAtPath];
        [mailVC addAttachmentData:fileData
                     mimeType:@"application/pdf"
                     fileName:@"results"];

        // Presentar la vista correspondiente.
        [self presentViewController:mailVC animated:YES completion:nil];
    }
    else
    {
        NSLog(@"Mail cannot be sent");
    }
}
```

Objeto visual relacionado

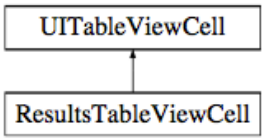
Figura 33.- Controlador de la vista de compartición de resultados.

2.4.15. Clase “ResultsTableViewCell”

Descripción

Permite comandar una celda personalizada para una tabla de resultados. Define varias propiedades tipo Outlet para comunicarse con los elementos de la capa visual con el fin de mostrar los resultados en una vista contenedora de tabla.

Herencia



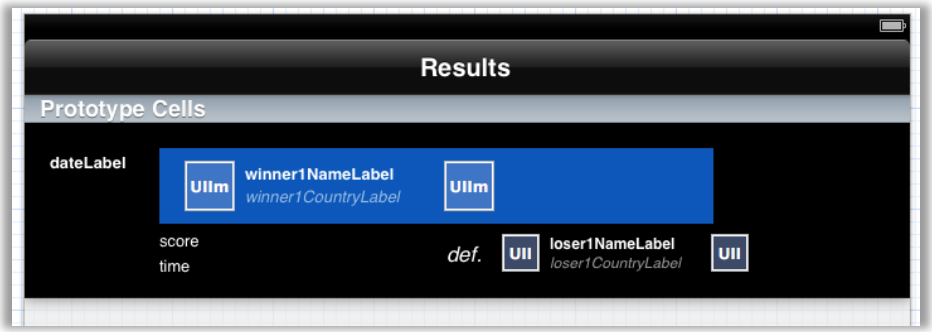
Propiedades

- IBOutlet UILabel * **winnerName1Label**
- IBOutlet UILabel * **winnerCountry1Label**
- IBOutlet UILabel * **winnerName2Label**
- IBOutlet UILabel * **winnerCountry2Label**
- IBOutlet UILabel * **loserName1Label**

- IBOutlet UILabel * **loserCountry1Label**
- IBOutlet UILabel * **loserName2Label**
- IBOutlet UILabel * **loserCountry2Label**
- IBOutlet UILabel * **matchDateLabel**
- IBOutlet UILabel * **matchTimeLabel**
- IBOutlet UILabel * **scoreLabel**
- IBOutlet UIImageView * **winner1CountryFlagImage**
- IBOutlet UIImageView * **winner2CountryFlagImage**
- IBOutlet UIImageView * **loser1CountryFlagImage**
- IBOutlet UIImageView * **loser2CountryFlagImage**
- IBOutlet UIView * **winnersSubView**

Objeto visual relacionado

Figura 34.- Celda genérica para resultados.

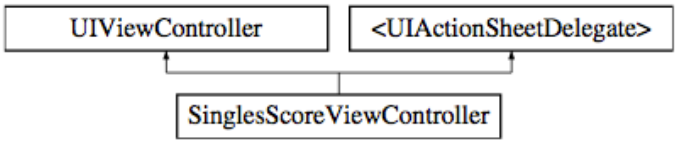


2.4.16. Clase "SinglesScoreViewController"

Descripción

Controla la vista del marcador para un partido de sencillos.

Herencia



Propiedades

- IBOutlet UILabel * **playerA_nameLabel**
- IBOutlet UILabel * **playerA_infoLabel**
- IBOutlet UIImageView * **playerA_countryFlagImageView**
- IBOutlet UIImageView * **playerA_photoImageView**
- IBOutlet UIImageView * **playerA_serviceImageView**
- IBOutlet UILabel * **playerB_nameLabel**
- IBOutlet UILabel * **playerB_infoLabel**
- IBOutlet UIImageView * **playerB_countryFlagImageView**
- IBOutlet UIImageView * **playerB_photoImageView**
- IBOutlet UIImageView * **playerB_serviceImageView**
- IBOutlet UILabel * **scoreLabelA**
- IBOutlet UILabel * **scoreLabelB**
- IBOutlet UILabel * **timeOutLabelA**
- IBOutlet UILabel * **timeOutLabelB**
- IBOutlet UILabel * **appealsLabelA**
- IBOutlet UILabel * **appealsLabelB**
- IBOutlet UILabel * **gamesLabelA**
- IBOutlet UILabel * **gamesLabelB**
- IBOutlet UILabel * **matchTimeLabel**
- IBOutlet UIImageView * **timeImageView**
- IBOutlet UILabel * **pointsGame1ALabel**
- IBOutlet UILabel * **pointsGame1BLabel**
- IBOutlet UILabel * **pointsGame2ALabel**
- IBOutlet UILabel * **pointsGame2BLabel**
- IBOutlet UILabel * **pointsGame3ALabel**
- IBOutlet UILabel * **pointsGame3BLabel**
- IBOutlet UILabel * **pointsGame4ALabel**
- IBOutlet UILabel * **pointsGame4BLabel**

Métodos de Clase

Obtener jugadores que intervienen en la partida.

```
+ (NSMutableArray *)getArrayOfSinglesMatchPlayers;
```

Borrar registro de jugadores que fueron seleccionados en partidas anteriores.

```
+ (void) resetSelectedPlayers;
```

Métodos de Instancia

Cambiar de servidor.

```
- (IBAction)doubleTapTwoFingers:(id)sender;
```

Adicionar punto.

```
- (IBAction)swipeUpScore:(id)sender;
```

Reducir punto.

```
- (IBAction)swipeDownScore:(id)sender;
```

Usar tiempo fuera.

```
- (IBAction)longPressUseTimeOut:(id)sender;
```

Usar apelación.

```
- (IBAction)longPressUseAppeal:(id)sender;
```

Salir del marcador.

```
- (IBAction)quitScoreboard:(UIButton *)sender;
```

2.4.17. Clase “DoublesScoreViewController”

Descripción

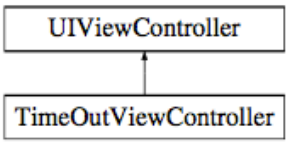
Controla la vista del marcador para un partido de dobles. Cumple la misma función que la clase “SinglesScoreViewController”, a diferencia de que incluye el manejo de dos jugadores mas para representar la modalidad de dobles. Sus métodos y herencia son similares a dicha clase mencionada.

2.4.18. Clase “TimeOutViewController”

Descripción

Controla la vista del tiempo fuera.

Herencia



Propiedades

- IBOutlet UILabel * **countdownTimerLabel**

Métodos de Clase

Configurar duración del tiempo fuera.

```
+ (void)setSecondsForContdownTimer:(int)seconds;
```

Métodos de Instancia

Ocultar vista del tiempo fuera.

```
- (IBAction)dismissTimeOutButton:(UIButton *)sender;
```

Código relevante

Crear un temporizador.

Segmento de código 22.- Creación de un temporizador.

```
    countDownTimer = [NSTimer scheduledTimerWithTimeInterval:1
                                                      target:self
                                                    selector:@selector(timerRun)
                                                    userInfo:nil
                                                     repeats:YES];
```

2.5. Resultados funcionales

El IDE Xcode incorpora una herramienta sumamente útil para el desarrollo de aplicaciones, un simulador de dispositivos móviles en que se pueden probar las aplicaciones sin necesidad de sincronizarlas continuamente con un aparato real. Luego de la implementación del código necesario para conectar funcionalmente las capas de Modelo, Vista, y Controlador, y en base al esquema general planteado se obtiene los resultados visuales que se apreciarán a continuación en las diferentes figuras capturadas a partir del funcionamiento real.

Pantalla de inicio

Al iniciar la aplicación se muestra una pantalla con una imagen que generalmente se refiere al desarrollador, dicha imagen se la carga directamente en Xcode y se puede controlar su duración.

Figura 35.- Imagen de inicio de la aplicación.

Pestaña de jugadores

En esta sección maneja los datos de los jugadores, y permite ver el detalle de cada uno de ellos. Antes de que la lista de jugadores tenga elementos, la instancia que se muestra es la que se aprecia en la *Figura 36.*

Figura 36.- Vista dividida para jugadores.

Claramente se puede observar una tabla vacía del lado izquierdo, y del otro lado, un espacio grande para mostrar en pantalla la información relacionada a un jugador específico. Se incluye un botón que indica la posibilidad de agregar elementos a la tabla. Una vez que se presiona dicho botón, aparece un formulario con campos de información básica para ser llenado.

Formulario para añadir jugadores

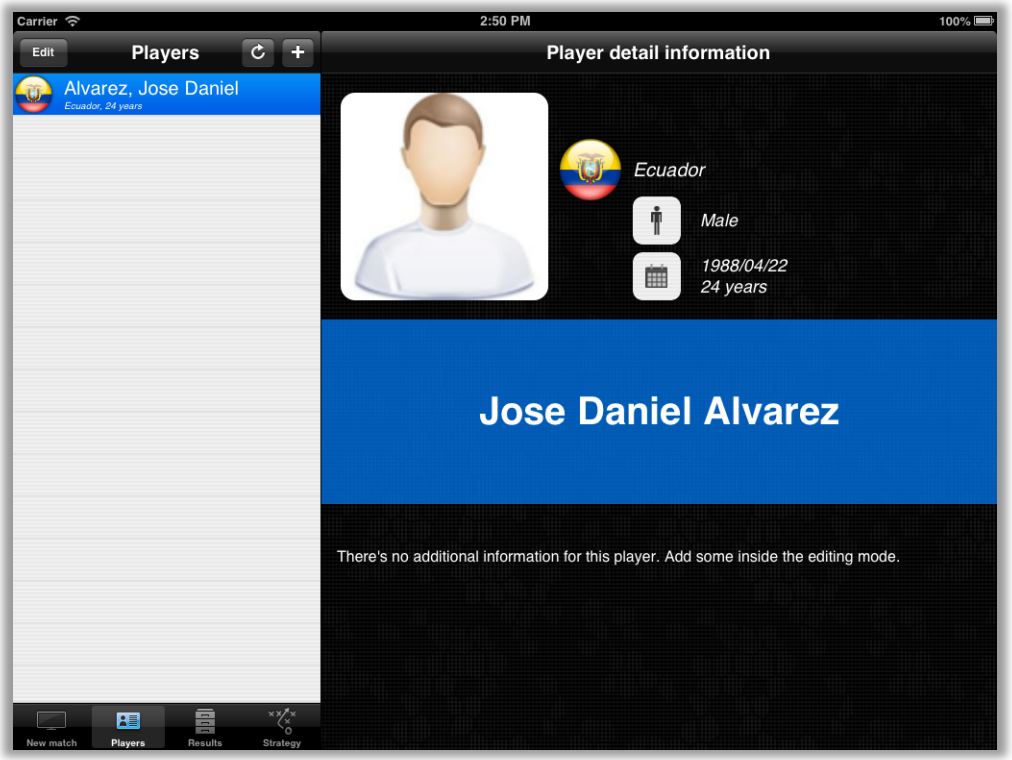
El formulario solicita la información básica: nombre, apellido, género, nacionalidad, fecha de nacimiento, y fotografía; siendo este último elemento de carácter opcional y que puede editarse posteriormente en caso de ser necesario. La *Figura 37* ilustra lo mencionado.

Figura 37.- Formulario para añadir jugador.

Detalle de un jugador

Cuando la lista de jugadores no está vacía, se puede presionar sobre uno de los nombres para ver el detalle correspondiente *(ver Figura 38)*.

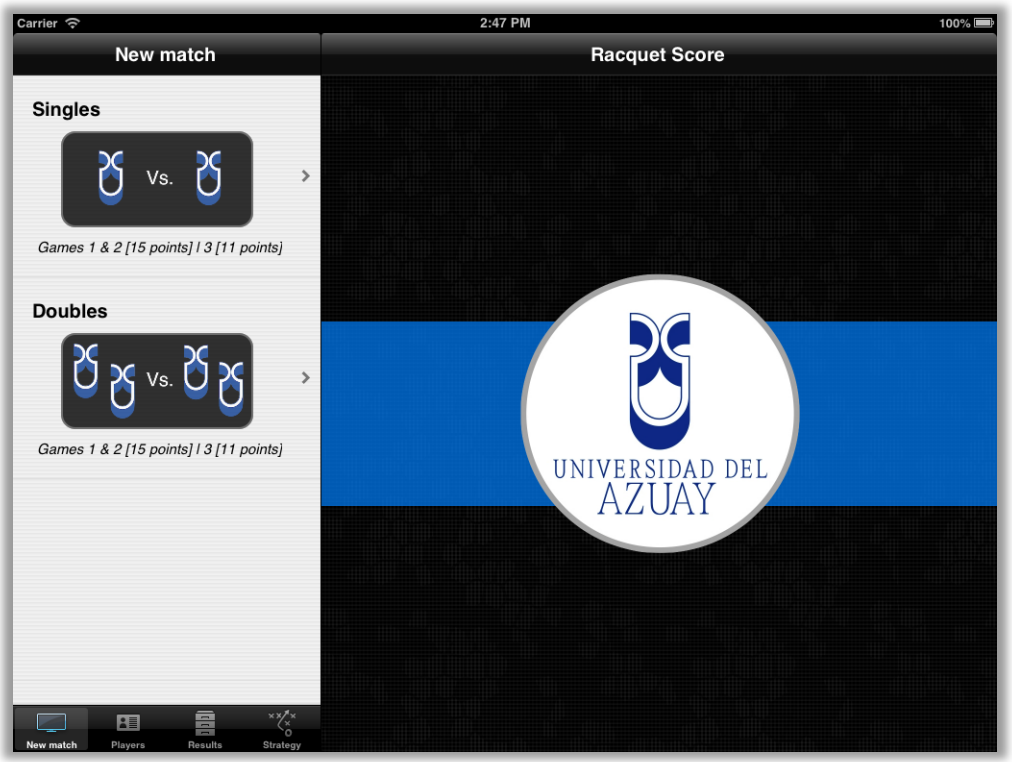
Figura 38.- Detalle de un jugador específico.



Pestaña de juego nuevo

Cuando se desea iniciar una nueva partida, la pestaña para un juego nuevo es la que ofrece esta función. En ella se puede empezar eligiendo la modalidad *(ver Figura 39)*.

Figura 39.- Pestaña de juego nuevo, selección de modalidad.



Selección de jugadores

Dentro de cualquiera de las modalidades de juego, la aplicación solicita la selección de los jugadores que intervendrán en la partida. La tabla de jugadores de la *Figura 38* se reutiliza para cada instancia de la selección. El espacio de detalle por su parte, muestra en pantalla una previa del partido que se está configurando, una vez seleccionado los jugadores se requiere la elección del la opción para asignación del servicio, esta puede ser por medio de asignación manual o por sorteo *(ver Figura 40)*. Finalmente, cuando se ha configurado el juego, queda por última instancia una vista que representa una previa del siguiente partido que estará por comenzar (*ver Figura 41)*, en caso de alguna equivocación, se puede cancelar el proceso en cualquier instancia.

Figura 40.- Selección del modo de asignación del primer servidor del juego.

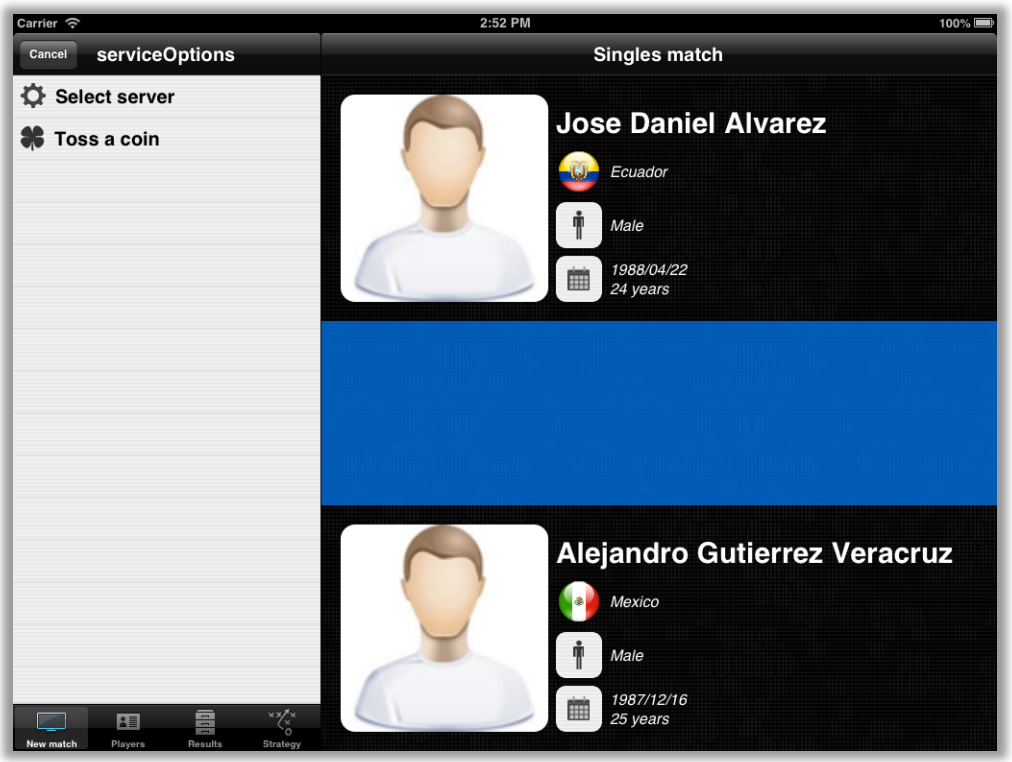
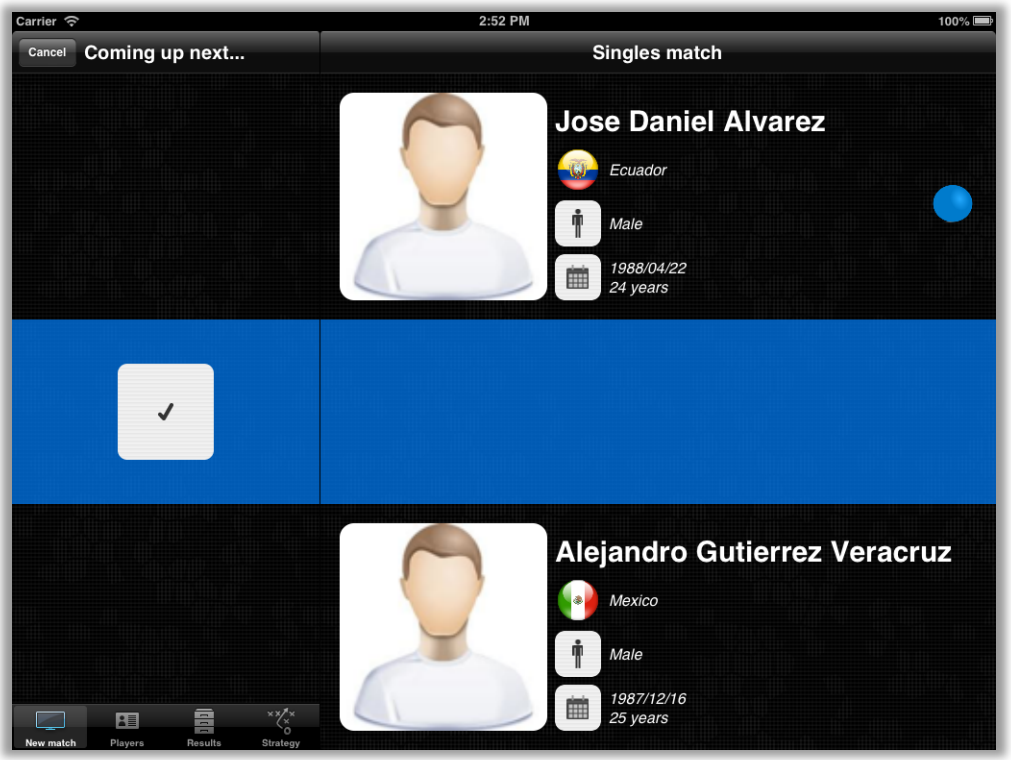


Figura 41.- Previa de un juego que está por empezar.



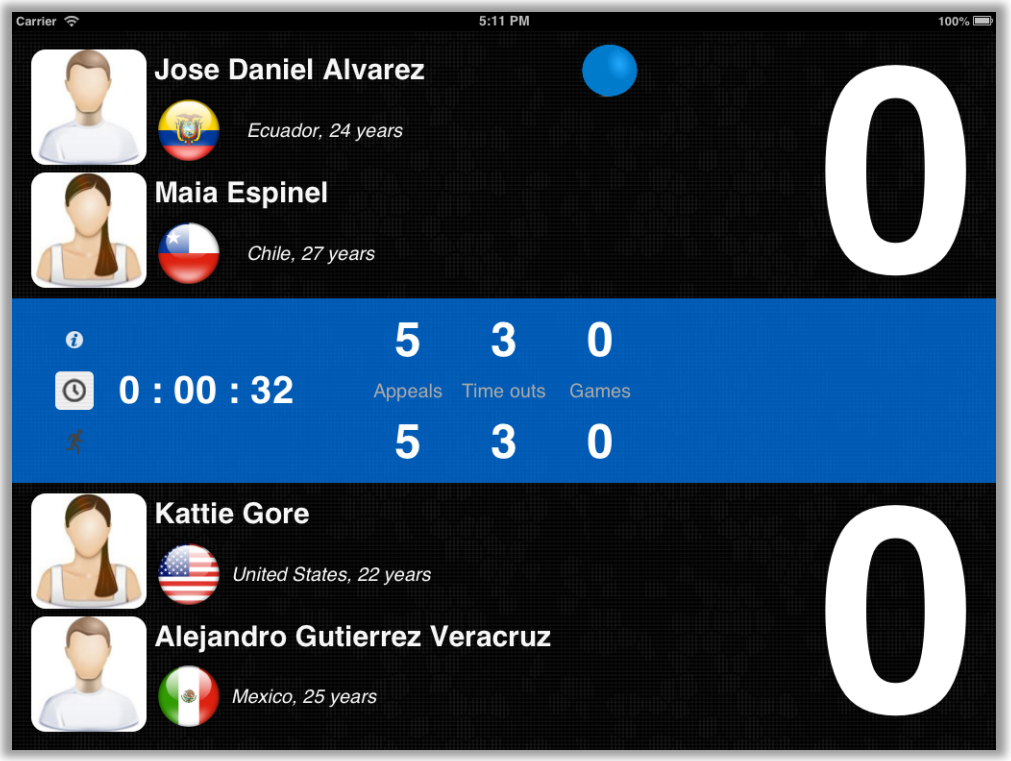
Marcadores

Cuando se da inicio a un juego, dependiendo de la modalidad seleccionada, aparece en toda la pantalla el marcador con los parámetros de arbitraje *(ver Figura 42 y Figura 43)*. En ella el juez del partido podrá tomar directamente acciones sobre el puntaje, tiempos fuera, apelaciones, o si prefiere simplemente salir del marcador para regresar al menú.

Para realizar el cambio de servidor basta con presionar dos veces con dos dedos sobre cualquier lugar en la superficie de la pantalla. De la misma manera para incrementar o corregir puntajes las acciones tienen lugar con un simple deslizamiento de un dedo hacia arriba o hacia abajo del número. Y por último para hacer uso de los tiempos fuera o apelaciones se requiere de que un dedo presione por más de un segundo sobre el número que hace referencia a la cantidad de donde se desea utilizar el parámetro.

Figura 42.- Marcador para la modalidad sencillos.

Figura 43.- Marcador para la modalidad dobles.



Tiempos fuera

Al hacer uso de un tiempo fuera, aparece una vista pequeña en la mitad de la pantalla, que indica un temporizador con los segundos restantes de dicho tiempo fuera *(ver Figura 44)*.

Figura 44.- Vista correspondiente a un tiempo fuera.

Partido finalizado

Cuando un partido termina, el marcador que estaba activo se esconde para regresar al menú principal, además se advierte del resultado por medio de una alerta visual *(ver Figura 45)*.

Figura 45.- Instancia de un partido finalizado.

Pestaña de resultados

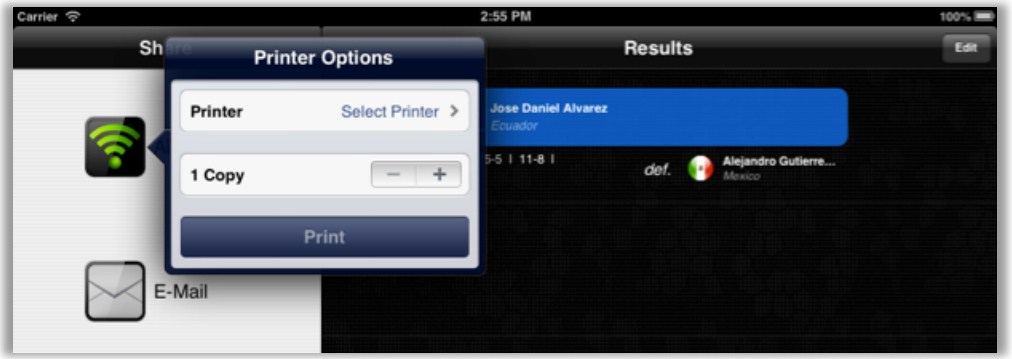
Los partidos que se vayan completando dentro de la aplicación se guardarán en la base de datos. Para visualizar o compartir resultados, la pestaña de resultados es el lugar indicado para esto *(ver Figura 46)*.

Figura 46.- Vista dividida para la instancia de resultados.

Imprimir

La primera acción posible es realizar una impresión inalámbrica de todos los resultados disponibles *(ver Figura 47)*.

Figura 47.- Imprimir todos los resultados.



Enviar un correo electrónico

Se puede enviar un correo electrónico con un texto que redacte los resultados, inclusive se incluye un archivo "PDF" adjunto con todos los resultados disponibles en caso de que se requiera con algún otro propósito *(ver Figura 48)*.

Figura 48.- Correo electrónico con todos los resultados.

Compartir en redes sociales

Las dos acciones restantes de compartición de resultados contemplan las redes sociales Facebook y Twitter, se puede publicar con facilidad contenido sobre los resultados de los partidos jugados en la aplicación, y en el caso de Twitter, al permitir publicar cadenas de hasta 140 caracteres como máximo, se prefiere para incluir un mensaje personal sobre la aplicación *(ver Figura 49 y Figura 50)*.

Figura 49.- Publicar resultados en Facebook.

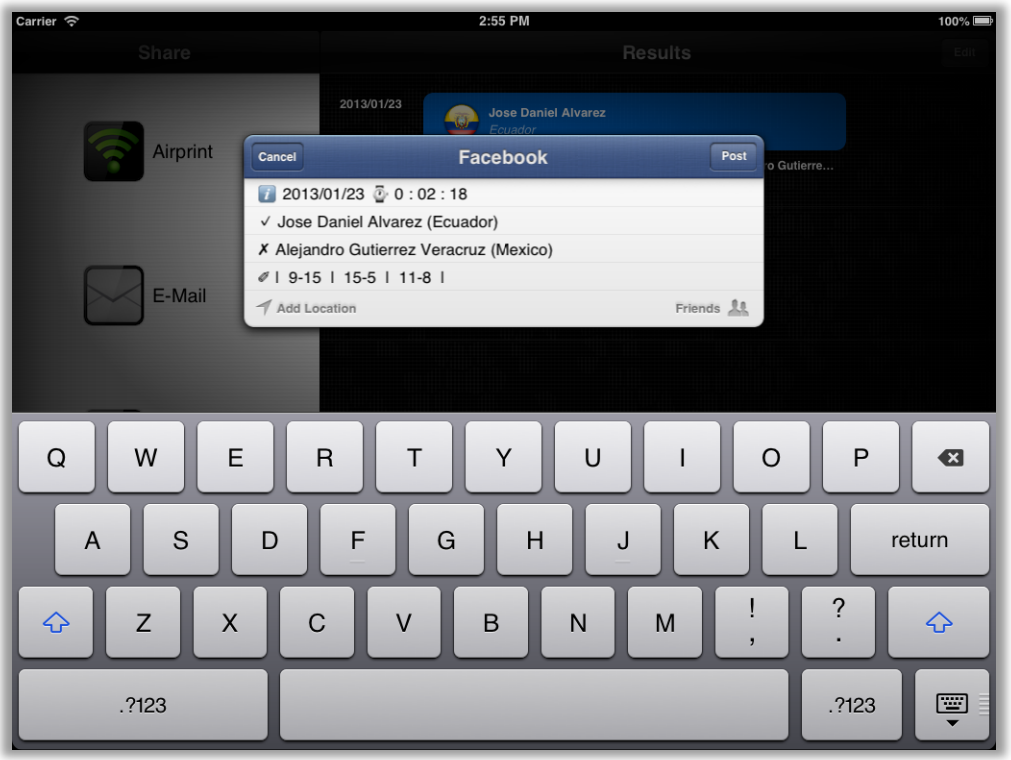
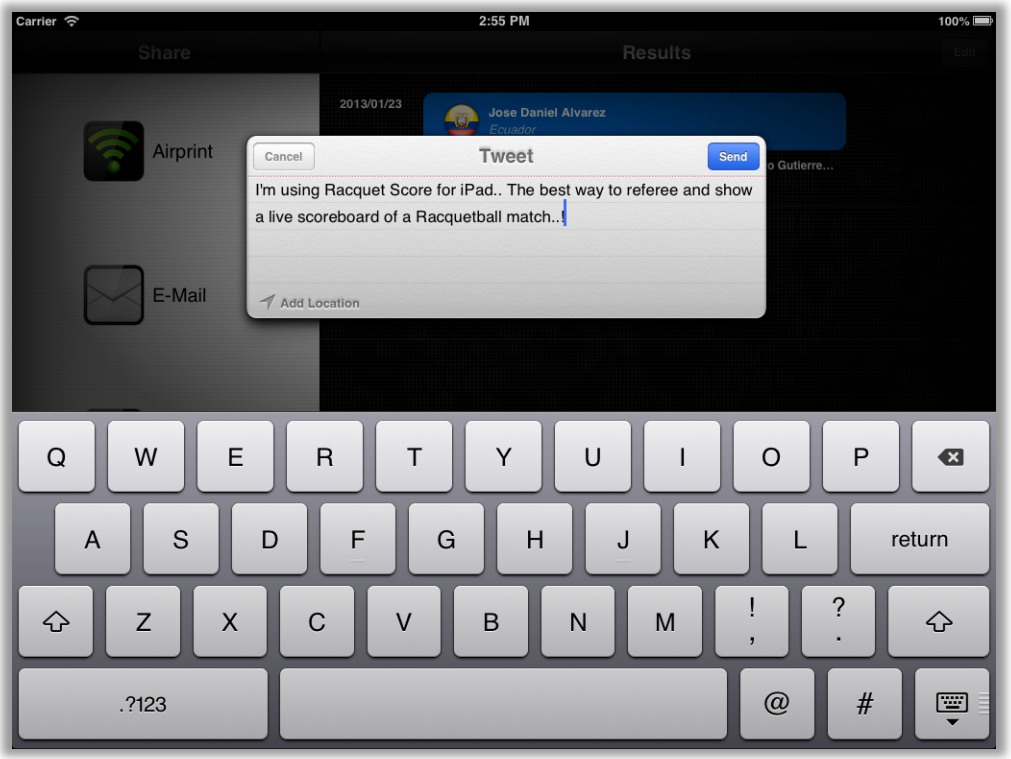


Figura 50.- Compartir mensaje sobre la aplicación en Twitter.



Compartir resultados de forma individual

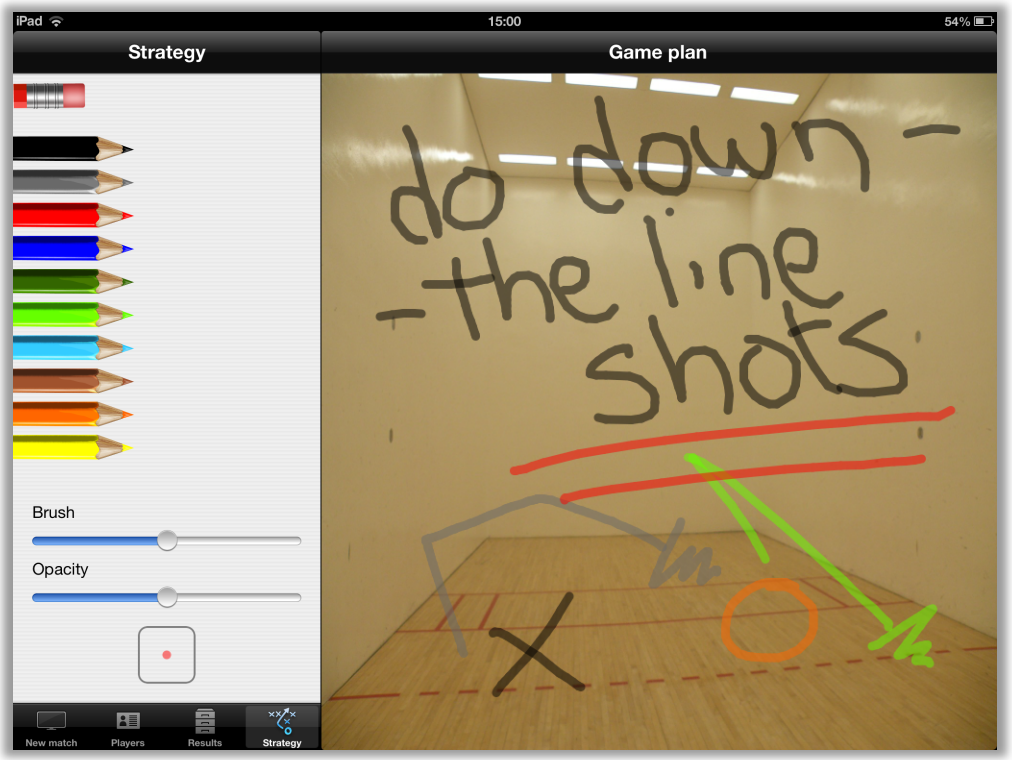
En caso de que existan más resultados, y sólo se desee compartir uno de ellos, simplemense te presiona sobre el resultado deseado y una subvista aparecerá con las opciones de compartición *(ver Figura 51)*.

Figura 51.- Compartición de un resultado específico.

Pestaña de estrategia

Aunque en un inicio estaba planteado como una posible mejora, adicionalmente a las funciones anteriormente citadas, se incluye una última pestaña sobre estrategias, la misma que da acceso a un espacio para dibujar estrategias de juego *(ver Figura 52)*, puede ser aprovechado para dar instrucciones de juego a un jugador, aunque no necesariamente está vinculado con el marcador, constituye una interesante función para expresar fácilmente ideas sobre aspectos diversos del juego. Se puede dibujar con varios colores, además permite cambiar el grosor y la transparencia del puntero.

Figura 52.- Vista para el dibujo de estrategias.



CAPÍTULO 3

IMPLEMENTACIÓN EN EL DISPOSITIVO

3.1. Programas para desarrolladores iOS

La primera consideración a tomar en cuenta antes de desarrollar aplicaciones para iOS, es sin duda alguna el enrolarse en uno de los programas para desarrolladores que Apple ofrece. Los productos de Apple, constituyen un sistema cerrado en el cual su hardware y software están construidos el uno para el otro. Así pues, no existe una manera legal (fuera de los programas de desarrolladores de Apple) en la que se pueda sincronizar las aplicaciones desarrolladas en Xcode con los dispositivos reales, y mucho menos de comercializarlas en la tienda virtual. Se puede desarrollar en tres ramas: Mac, Safari e iOS. La última mencionada corresponde a la categoría a la que está enfocado este trabajo. Para información completa y actualizada sobre todos los programas para desarrolladores que Apple ofrece se encuentra la página web: https://developer.apple.com

Dentro de la rama de desarrollo para iOS existen programas específicos que van de acuerdo a las necesidades de la persona que busca enrolarse, los mismos que se explican brevemente a continuación.

Programa “MFi” para iOS

Este programa hace referencia al desarrollo de dispositivos que sean compatibles con artefactos que utilicen iOS. Comprende la obtención de componentes de hardware, herramientas, documentación, soporte técnico y logotipos certificados necesarios para crear accesorios de audio y video que se puedan conectar con un iPod, iPhone, y iPad.

Figura 53.- Programa "MFi" para iOS.

Fuente: APPLE INC. *iOS Developer Program – Apple Developer*. Disponible en: https://developer.apple.com/programs/ios/

Programa “Enterpise” para iOS

Dirigida para aplicaciones que buscan ser usadas en negocios propios o corporaciones, es decir, todas aquellas en las que se desean sincronizar con tantos dispositivos como se necesiten para un fin propio. Este tipo de aplicaciones no poseen necesidad de ser comercializadas en la tienda virtual, ya que no persigue masificación sino crecimiento autónomo de un negocio o institución.

Figura 54.- Programa "Enterprise" para iOS.

Fuente: APPLE INC. *iOS Developer Program – Apple Developer*. Disponible en: https://developer.apple.com/programs/ios/

Programa general para iOS

Comprende la solución a las necesidades más comunes de los desarrolladores, es decir, permisos para sincronizar con dispositivos reales a manera de prueba y posterior comercialización en la tienda. Básicamente se resume en tres sencillos pasos: desarrollar, probar, y distribuir.

Figura 55.- Programa general para desarrolladores iOS.

Fuente: APPLE INC. *iOS Developer Program – Apple Developer*. Disponible en: https://developer.apple.com/programs/ios/

Programa Universitario para iOS

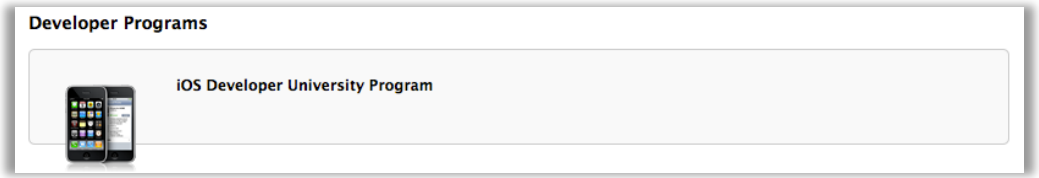
Para un desarrollador que busca iniciarse en este campo, el punto de partida más conveniente es sin duda un programa universitario. A este programa se accede a través de centros educativos. La Universidad del Azuay (a la fecha de desarrollo de este trabajo) cuenta con licencia de este programa, el mismo que permite a sus estudiantes el acceso a todas las funciones de un programa general de desarrollador, exceptuando la comercialización de las aplicaciones en la tienda virtual.

Figura 56.- Programa Universitario para iOS.

Fuente: APPLE INC. *iOS Developer Program – Apple Developer*. Disponible en: https://developer.apple.com/programs/ios/

Para formar parte del programa universitario, se debe contactar con el administrador del grupo, y solicitar la inclusión como miembro. Una vez que el administrador envíe la invitación y se acepte la misma, en el perfil de usuario se apreciará la membresía como lo que se muestra en la *Figura 57*.

Figura 57.- Membresía de programa universitario registrada.



3.2. Sincronizar la aplicación en un dispositivo real

Antes de ver nuestra aplicación ejecutándose en el iPad, es necesario cumplir con tres etapas: configuración de perfil, instalación de certificados, y sincronización desde Xcode. Lo primero que se debe hacer es pedir al administrador del programa, que agregue nuestro dispositivo a la lista de dispositivos registrados para el equipo, esto se hace simplemente proporcionando el ID único del dispositivo, el mismo que se lo puede conseguir en iTunes o en el Organizador de Xcode (con el dispositivo conectado), además se debe activar la opción que cita "Use for Development" *(ver Figura 58)*.

Figura 58.- Detalles del dispositivo en Xcode.

Ahora se debe realizar la configuración del perfil y la instalación de permisos y certificados, lo que antes podía resultar un proceso complicado, ahora Apple incluye dentro de su página de desarrolladores de iOS (https://developer.apple.com/membercenter/) un ayudante que paso a paso sirve de guía para cumplir estas etapas. Las figuras sucesivas que se incluyen a continuación, muestran paso a paso el proceso. Al entrar en el ayudante se empieza con una imagen como se muestra en la *Figura 59* que será el punto de partida.

Figura 59.- Asistente para certificados y firmas.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 60.- Identificación única de la aplicación.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 61.- Selección del dispositivo deseado.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 62.- Solicitar certificado para desarrollador iOS.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer*. Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 63.- Solicitar certificado dentro del computador.

Figura 64.- Seleccionar certificado generado.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 65.- Seleccionar ubicación del certificado.

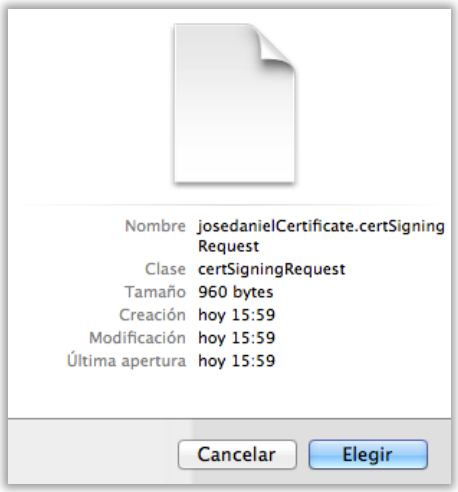
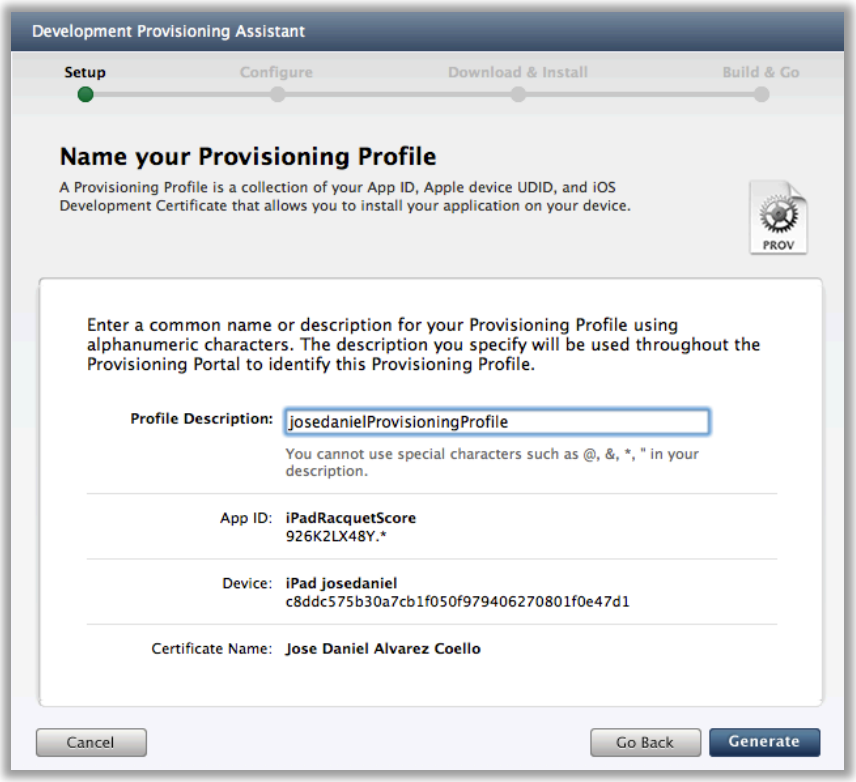
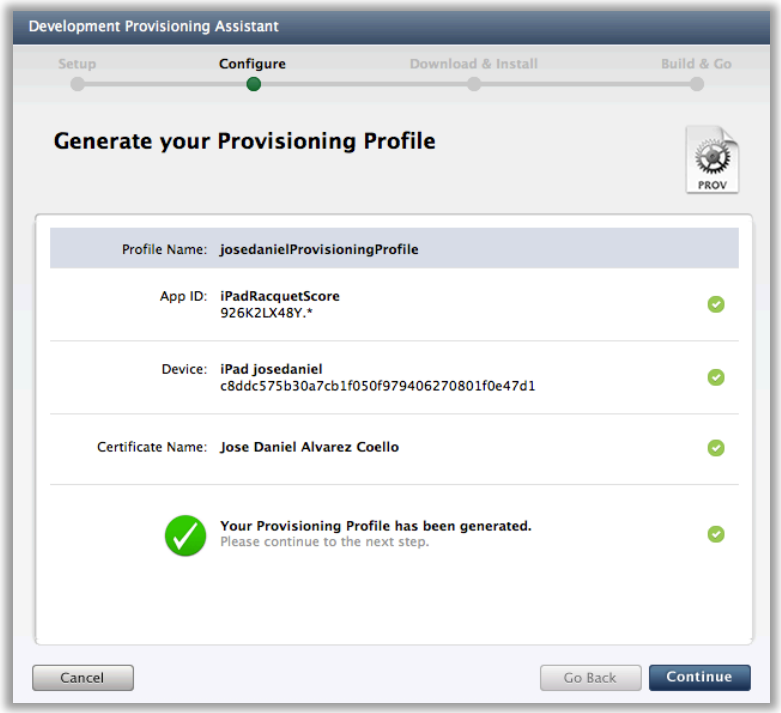


Figura 66.- Generar perfil de provisión.



Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer*. Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 67.- Resultado de la creación del perfil de provisión.

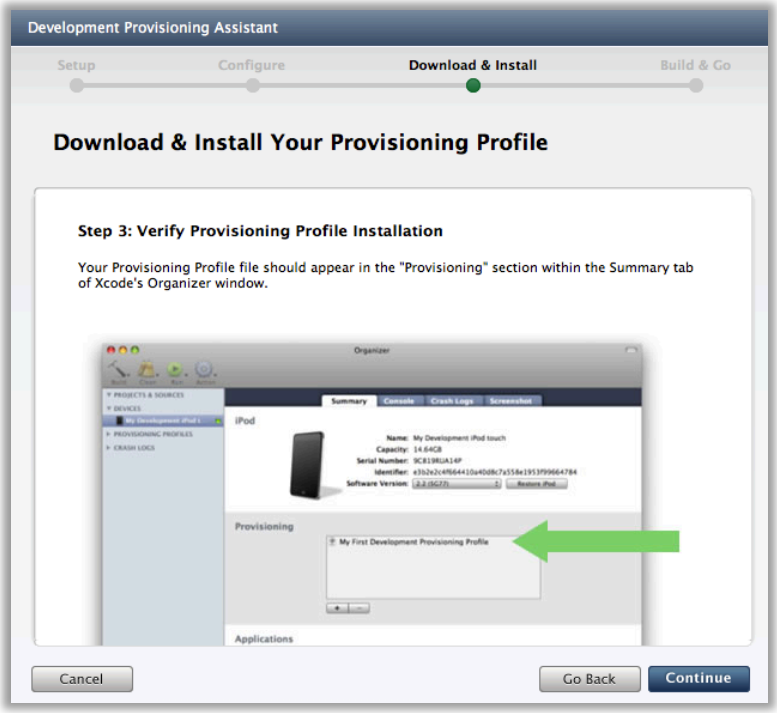


Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer*. Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 68.- Descarga e instalación del perfil de provisión.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 69.- Verificación de la instalación del perfil de provisión.



Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 70.- Perfil de provisión instalado en el dispositivo.

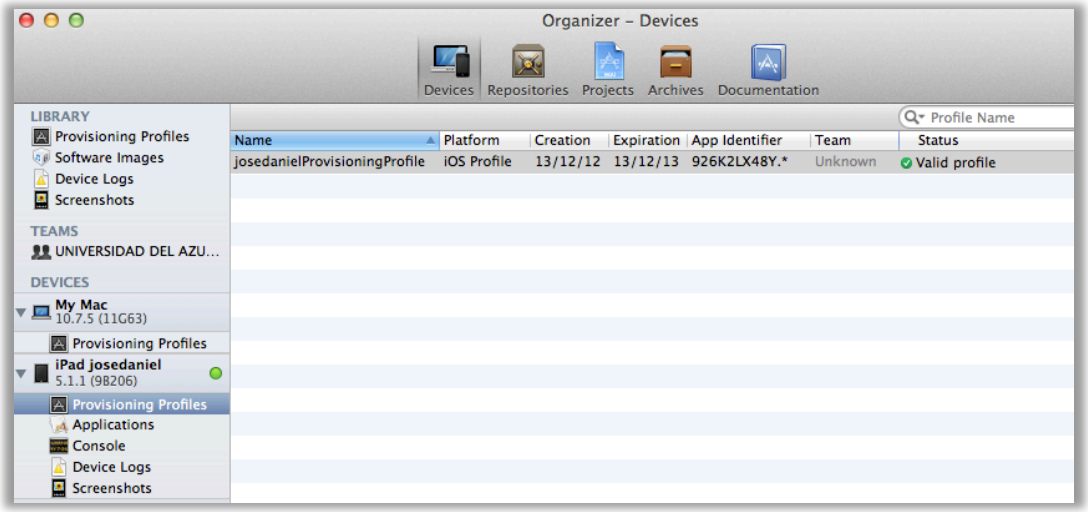


Figura 71.- Instalación del certificado de desarrollador.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 72.- Certificado instalado en el computador.

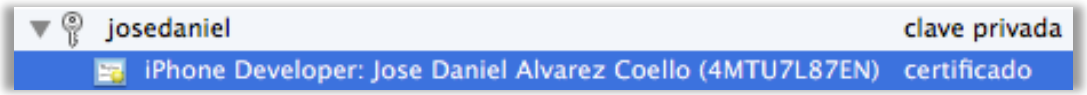


Figura 73.- Sincronizar la aplicación usando Xcode.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Figura 74.- Mensaje final del asistente de creación de certificados y perfil de provisión.

Fuente: APPLE INC. *iOS Provisioning Portal – Apple Developer.* Disponible en: https://developer.apple.com/ios/manage/overview/index.action

Una vez concluido con los pasos para creación del perfil de provisión e instalación de certificados, Xcode se encuentra habilitado para instalar las aplicaciones desarrolladas en el dispositivo registrado. Así pues antes de correr la aplicación Racquet Score desde Xcode deberemos seleccionar el iPad en lugar del simulador iOS, tal como lo muestra la *Figura 75*.

Figura 75.- Selección del dispositivo destino de la aplicación.

CONCLUSIONES Y RECOMENDACIONES

En la última década, los avances tecnológicos han ido evolucionando, centrándose en la integración de varios de componentes de hardware de última generación dentro de un mismo dispositivo, esta tendencia seguirá siendo el motor que impulse a grandes compañías a producir artefactos innovadores durante muchos años más. Las compañías que lideran el mercado tecnológico actual en algún momento apostaron acertadamente a la utilización de la programación orientada a objetos como el núcleo del lenguaje de sus sistemas operativos. Tomando en cuenta aquello, resulta interesante el hecho de que un estudiante de Ingeniería Electrónica pueda tener acceso funcional hacia cada componente de la tecnología móvil existente en el mercado, con el fin de utilizar cada elemento a gusto según el objetivo de la aplicación o proyecto que se desee realizar.

Considero de suma importancia que, asignaturas de desarrollo de software basadas en programación orientada a objetos se deben mantener dentro de la malla curricular e incluso de ser posible profundizarlas empezando desde ciclos inferiores, con el propósito de que, durante los últimos semestres de la carrera el estudiante posea acceso completo a una gran variedad de dispositivos de última tecnología, pero no como consumidor final, sino que dichos dispositivos se empiecen a considerar como una herramienta más para el desarrollo de proyectos electrónicos de gran escala y con una imagen distinguida.

Al mirar los computadores, teléfonos o tabletas electrónicas actuales como una herramienta de trabajo para el Ingeniero Electrónico, se podrá conseguir que dichos artefactos sirvan de interfaz entre el usuario y los circuitos, permitiendo la creación de mandos táctiles remotos o comandos de voz programables para robots, prótesis, alarmas, etc. Además de abrir la oportunidad de conectarse a la red para realizar monitoreo e instrumentación remota de proyectos electrónicos, bioelectrónicos, procesos automáticos, y demás ideas que surjan del ingenio de los estudiantes.

Trabajar sobre un dispositivo móvil ha constituido una experiencia enriquecedora que abre puertas hacia la integración de las ideas con el mundo tecnológico actual, y permite aportar soluciones prácticas y de fácil uso. Al finalizar este trabajo se realizaron pruebas de funcionamiento y promoción de la aplicación en el Campeonato Panamericano de Racquetball realizado en Cali – Colombia en Marzo de 2013 *(Ver*

Figuras 76 – 79), dando como resultado comentarios positivos de parte de los participantes del evento, así como también la Federación Internacional de Racquetball manifestó su felicitación y aceptación de la aplicación propuesta para incluirla posteriormente a un sistema integral para futuros campeonatos oficiales *(ver Anexo 1)*.

Figura 76.- Marcador grande transmitido desde la aplicación Racquet Score.

Figura 77.- Tableta de arbitraje y marcador pequeño con Racquet Score.

Figura 78.- Tableta de arbitraje con Racquet Score.

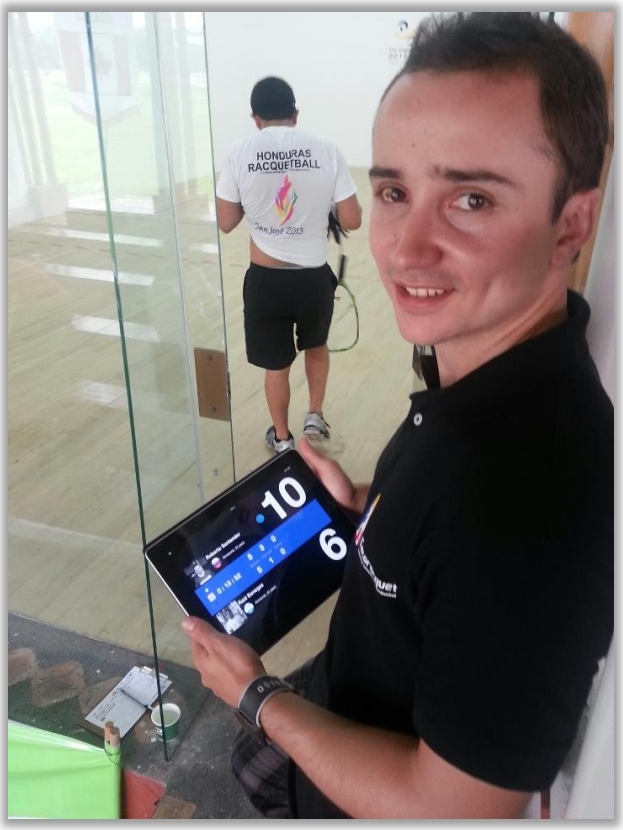


Figura 79.- Final del Campeonato Panamericano 2013 con Racquet Score.

BIBLIOGRAFÍA

- APPLE INC. (2010). *Object-Oriented Programming with Objective-C*. Cupertino, California: Apple Inc. En línea. Citado (Marzo 2012). Disponible en:

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/OOP_ObjC/OOP_ObjC.pdf

- APPLE INC. (2011). *The Objective-C Programming Language*. Cupertino, California: Apple Inc. En línea. Citado (Marzo 2012). Disponible en:

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf

- APPLE INC. (2010). *Cocoa Fundamentals Guide*. Cupertino, California: Apple Inc. En línea. Citado (Marzo 2012). Disponible en:

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf

- APPLE INC. (2011). *iOS Technology Overview*. Cupertino, California: Apple Inc. En línea. Citado (Marzo 2012). Disponible en:

https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf

- APPLE INC. (2012). *iOS App Programming Guide*. Cupertino, California: Apple Inc. En línea. Citado (Marzo 2012). Disponible en:

https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneAppProgrammingGuide.pdf

- APPLE INC. (2011). *iOS Human Interface Guidelines*. Cupertino, California: Apple Inc. En línea. Citado (Marzo 2012). Disponible en:

https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/MobileHIG.pdf

- NAHAVANDIPOOR, Vandad (2012). *iOS 5 Programming Cookbook*. Sebastopol, California: O'REILLY.

- DANIEL, Steven (2011). *Xcode 4 iOS Development*. Mumbai: PACKT.

- GRIMES, Shawn y FRANCIS, Colin (2012). *iOS 5 Recipes*. Estados Unidos: Apress.

- SMYTH, Neil (2012). *iPad iOS 5 Development Essentials*. Estados Unidos: NEIL SMYTH.

- APPLE INC. *iOS Developers Lybrary*. Cupertino, California: Apple Inc. En línea. Citado (Marzo 2012). Disponible en: https://developer.apple.com/library/ios/navigation/

ANEXOS

Anexo 1.- Carta de la Federación Internacional de Racquetball.

International Racquetball Federation (IRF)

Member of the International Olympic Committee

Universidad Del Azuay March 25, 2013

Dear Sirs,

I am Secretary General of the International Racquetball Federation based out of the United States in the state of Colorado. The IRF is one of the recognized sports under the International Olympic Committee for inclusion in future programs of the Olympic Games.

I had to witness the exhibition of Mr. Jose Daniel Alvarez Coello's iPad application for tournament communication during the Pan American Racquetball competition Easter week in Cali, Colombia. Our Ex Committee has been searching for such a application for the past 5 years in order to communicate in real time the activity and results of our many matches.

This application is exactly what we were looking for and we hope to use it extensively in out events in the future.

Regards,

Luke St Onge
Sec General—IRF

Cc President-- Osvaldo Maggi—IRF

IRF Ex Committee

1631 Mesa Avenue, Suite A-1 • Colorado Springs, CO 80906, USA
719-477-6934 • fax 719-634-5198
• All contributions are tax deductible•