UNIVERSIDAD DEL AZUAY

Facultad de Ciencias de la Administración

Escuela de Ingeniería de Sistemas y Telemática

Trabajo de Titulación: Prototipo de un Sistema Domótico para controlar Iluminación y Cámaras de Seguridad mediante Dispositivos Móviles.

Trabajo de Titulación, previo a la obtención del título de Ingeniero de Sistemas y Telemática.

Autores:

Pedro Andrés Cárdenas Vásquez

Pedro Sebastián Corral Jaramillo

Director de tesis:

Ing. Esteban Crespo

Cuenca – Ecuador

2015

Dedicatoria

A todos nuestros Maestros, quienes a lo largo de toda la carrera nos han apoyado para culminar tanto este trabajo de titulación como la formación profesional, a nuestros Amigos y Familiares quienes con su magia, cariño y amor hicieron que todo esfuerzo se sienta recompensado.

_____ _____

Pedro Cárdenas V. **Pedro Corral J.**

Agradecimiento

A la Universidad del Azuay, a todos sus profesores y personal administrativo que a lo largo de toda la carrera, compartieron conmigo sus conocimientos, sus historias y sus experiencias tanto dentro como fuera de clases, por despertar en mí la sed del conocimiento y la investigación, pero principalmente por permitirme llamarles amigos en lugar de ingenieros.

A mis Maestros, quienes de noche y de día me llenaron de su luz, de su amor y su sabiduría para seguir adelante y culminar la carrera con éxito, sin importar el esfuerzo que tuve que realizar para cumplir mis tareas y obligaciones, sabiendo que todo esto servirá en un futuro para crear un mundo mejor ayudando a los demás.

A mi familia, quienes me llenaron de infinito amor cada día para crecer no académica ni intelectualmente sino mental y sobre todo interiormente.

A José Torres, quien me ayudó a ver y entender a las personas de otra manera, compartiendo conmigo una herramienta tan eficaz y poderosa para mis estudios a lo largo de la carrera, la PNL: Programación Neuro Lingüística.

A la magia, que me permitió compartir una sonrisa con mis compañeros, amigos, niños, profesores y toda la universidad.

A Fabián Carvajal, Esteban Crespo, Paola Merchán, Oswaldo Merchán, y Marcos Orellana por ayudarnos directamente en la elaboración de este trabajo de titulación, pero principalmente por compartir su tiempo conmigo y brindarme su amistad.

A Kenneth Palacio, por su confianza y cariño hacia mí, por sus inagotables palabras llenas de motivación, experiencia, enseñanza y felicidad, por estar ahí en todo momento para apoyar mi crecimiento académico e interior y por su gran amistad.

Principalmente, a mi amigo, colega y compañero, Pedro Corral, quien en incontables ocasiones ha estado ahí para ayudarme, por su forma de pensar, por compartir, discutir y crear magia en cada proyecto realizado, además de haber hecho que toda la carrera universitaria sea más sencilla, divertida y feliz.

Pedro Andrés Cárdenas Vásquez

Agradecimiento

A la Universidad del Azuay, que sin sus docentes y equipo administrativo que trabajan arduamente todos los días, ya que sin su trabajo y esfuerzo por crear una mejor universidad, esto no sería posible.

A todos los los profesores, quienes supieron transmitir sus conocimientos en el aula de clase llenándonos de nuevas ideas y sembrando en nosotros la necesidad de investigación y superación, agradezco cada experiencia que compartieron, tanto dentro como fuera del aula de clase, de seguro todo esto quedará plasmado dentro de nuestras vidas.

A la escalada, que además de brindarme las mejores experiencias de mi vida, ha sabido formarme como una mejor persona, a ponerme retos, a luchar por lo que quiero y sobre todo a buscar constantemente expandir mis límites.

A mi familia, que sin lugar a duda ha sido el pilar fundamental de apoyo en mi vida, me ha brindado un hogar, alimento, amor y toda la sabiduría necesaria para poder ser el creador de mi realidad.

Principalmente, a mi gran amigo, compañero y maestro, Pedro Cárdenas, que ha sabido compartir todas sus experiencias, tanto dentro de la universidad como fuera de ella, llenando de magia los días universitarios, haciendo de ellos mucho más divertidos, llevaderos y verdaderos. Gracias amigo por tu constante apoyo, paciencia, dedicación y esfuerzo, has hecho de la universidad una experiencia mágica.

Pedro Sebastián Corral Jaramillo

RESUMEN

El presente trabajo de titulación consiste en la implementación de un prototipo de un sistema domótico para controlar iluminación y cámaras de seguridad mediante dispositivos móviles, se explicará la estructura de dicho sistema, los módulos que lo componen, su arquitectura, funcionamiento e implementación, además de las tecnologías utilizadas para la codificación del mismo. Obteniendo como producto final el prototipo de un sistema dómotico centralizado, el cual se probará para demostrar su funcionamiento por medio de las aplicaciones móviles clientes desarrolladas sobre las plataformas iOS y Android.

ABSTRACT

This graduation paper aims at implementing a prototype of an automation system to control lighting and security cameras via mobile devices. We will explain the system's structure, its modules, architecture, operation and implementation, as well as the technologies used for its encoding. The final product obtained is the prototype of a centralized automation system, which will be tested to demonstrate its operation by mobile-client applications developed on iOS and Android platforms.

UNIVERSIDAD DEL
AZUAY
Dpto. Idiomas

Translated by,
Lic. Lourdes Crespo

1. Tabla de contenido

INTRODUCCIÓN

La evolución de las diferentes ramas tecnológicas, como la informática, las telecomunicaciones y la electrónica han permitido la integración de las mismas en un solo sistema con el fin de satisfacer necesidades dentro de los hogares, dando origen así a lo que hoy se conoce como sistema domótico.

Un sistema domótico ofrece seguridad, vigilancia, comodidad, confort y facilidad de control de un hogar, minimizando así el nivel de inseguridad al que se encuentra expuesto el mismo, y brindando a las personas mayor bienestar y comodidad familiar.

Los servicios que ofrecen los sistemas domóticos actuales tienen un limitante en el funcionamiento, ya que dependen de comandos y parámetros preestablecidos. Es por ello que a lo largo de este trabajo se encontrará una solución para que el control del sistema domótico se realice mediante un dispositivo móvil, el cual facilitará el acceso al sistema de vigilancia e iluminación y por ende brindará mayor facilidad, seguridad y comodidad al usuario final en la interacción con su vivienda.

1. CAPÍTULO 1: Marco Teórico

Introducción

La evolución de las diferentes ramas tecnológicas, como la informática, las telecomunicaciones y la electrónica han permitido la integración de las mismas en un solo sistema con el fin de satisfacer necesidades dentro de los hogares, dando origen así a lo que hoy se conoce como sistema domótico.

En este capítulo se describen los conceptos principales y los términos fundamentales para comprender las características generales y la funcionalidad que brinda un sistema domótico, además se explicará la arquitectura que utilizará dicho sistema y las plataformas de desarrollo móvil iOS y Android sobre las que se implementará.

1.1. Conceptos principales

1.1.1. Domótica

"La palabra domótica, proviene de la unión de la palabra "domo" y el sufijo "tica". La palabra "domo" etimológicamente proviene del latín *domus* que significa casa, y el sufijo "tica" proviene de la palabra automática, aunque algunos autores también diferencian entre "tic" de tecnologías de la información y de la comunicación y "a" de automatización." (Domótica e Inmótica: Viviendas y edificios inteligentes, pag.23).

La domótica es una rama tecnológica en auge que permite integrar y controlar de manera virtual muchas de las tareas de un hogar, tales como, iluminación, seguridad, control de dispositivos electrónicos, entre otros. (Anton)

La domótica permite a un usuario administrar y controlar su hogar de manera centralizada y virtual.

Las opciones dentro del campo de la domótica son muy extensas, ya que puede ir desde un simple sistema de seguridad hasta un control integrado de prácticamente todas las funciones de un hogar tales como, abrir y cerrar cortinas, encender y apagar luces, controlar televisores, controlar un sistema de cámaras de seguridad, abrir y cerrar puertas, control de acceso, etc.. (Anton)

1.2.Arquitectura Cliente Servidor

La arquitectura cliente servidor es un modelo utilizado para el desarrollo de sistemas de información. Hace referencia al proceso en el cual actúan dos participantes, el primero es el cliente que es el encargado de iniciar la comunicación, o en otras palabras, realizar una petición al otro actor o participante que es el servidor o encargado de responder a esta petición. Se podría decir que el cliente solicita recursos y el servidor responde a estas solicitudes, formando así el proceso de comunicación Cliente/Servidor. (Lopéz)

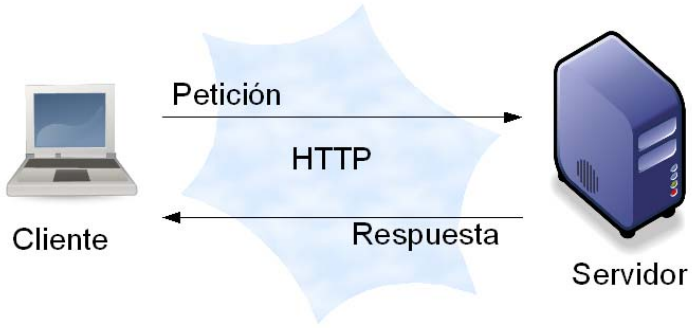


Fig. 1.1 Cliente/servidor

Fuente: http://grupo-701-marce nestore.wikispaces.com/Cliente-Servidor

1.2.1. Cliente

Según (Lopéz), el cliente es la parte frontal de la comunicación o *front-end*, es el proceso que interactúa con el usuario y le permite realizar las peticiones o solicitudes, por esta razón deben tener una interfaz gráfica (GUI), que es el rostro de la aplicación que le permite al usuario manejar de maneara más usable y fácil la misma, dentro de la arquitectura cliente/servidor, el cliente es el que establece la comunicación con el servidor a través de un usuario. En términos generales el cliente realiza las siguientes funciones:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados

1.2.2. Servidor

De acuerdo a lo mencionado por (Lopéz), el servidor como su nombre lo indica, es el proceso encargado de servir las peticiones de uno o múltiples clientes, también conocido como el *back-end.* Es el encargado de las funciones relacionadas con las reglas del negocio y los recursos de datos.

Entre las funciones que generalmente ejerce el servidor están:

- Aceptar los requerimientos de bases de datos que hacen los clientes.
- Procesar requerimientos de bases de datos.
- Formatear datos para trasmitirlos a los clientes.

- Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

Según (Berson, 2007), esta arquitectura brinda muchas ventajas tales como:

- Integración de diversos sistemas y plataformas.
- Al hacer uso de interfaces gráficas los sistemas basados en esta arquitectura son de fácil uso e interacción sencilla, además de brindar ayuda intuitiva para el usuario.
- Rápido mantenimiento y desarrollo de aplicaciones.
- De fácil escalabilidad, permitiendo integración y crecimiento rápido.

1.3. iOS

Fig.1.2. iOS (Apple)

Fuente: http://www.technied.com/apple

Las siglas iOS provienen de *iPhone Operating System*, debido a que este sistema operativo fue desarrollado por Apple para su *smartphone* conocido como *iPhone*. Este nació a partir del sistema operativo para computadores Mac OS X que a su vez tenía como base a DarwinBSD con su núcleo basado en Unix. Después, con la aparición de distintos dispositivos como el iPad y el Apple TV el sistema utiliza el acrónimo iOS como nombre oficial. (Apple Inc., 2014)

Este sistema operativo trabaja con los lenguajes de programación C y Objective-C como principales plataformas de desarrollo, siempre enfocando sus aplicaciones y nuevas actualizaciones en lograr un interfaz gráfico de usuario con un nivel de usabilidad cada vez más alto.

La definición de iOS que brinda Apple al mundo explica claramente lo que esta compañía quiere lograr con su software:

> "iOS es la base del iPhone, iPad, e iPod touch. Incluye una colección de apps que te permiten hacer las cosas de todos los días, e incluso otras no tan comunes, de manera intuitiva, simple y divertida." (Apple Inc., 2014)

Este sistema operativo consta de una tecnología de 4 capas:

1. *Core OS*: consta de las características y funcionalidades de más bajo nivel del núcleo del sistema operativo.
2. *Core Services:* está conformada por los servicios principales e interfaces necesarias para el control de las funciones básicas del sistema como la agenda, el reloj, gestión de datos, conexiones de internet y redes, etc.
3. *Media:* brinda la tecnología necesaria para el control del contenido multimedia: imágenes, audio, video, animaciones, etc. Además permite el control de la vibración del dispositivo móvil. Está implementada en C y Objective-C.
4. *Cocoa Touch:* es el framework de desarrollo basado en Objective-C que permite el control de las aplicaciones (apps) a través de interfaces táctiles (gestos) y el uso del acelerómetro del dispositivo. (Apple Inc., 2014)

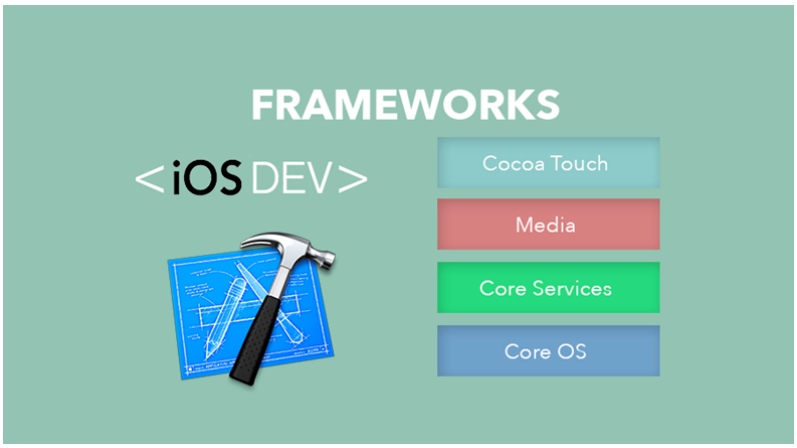


Fig. 1.3. Tecnología de Capas iOS.

Fuente: http://iosdev.es/frameworks/

1.4.Android

Fig.1.4. Android

Fuente: http://www.guioteca.com

Android, es un sistema operativo desarrollado por la empresa Android Inc. para dispositivos móviles inteligentes, pero Google compró esta empresa en el año 2005 cuando aún esta tecnología no era tan conocida como hoy en día. Ahora, este sistema operativo es utilizado en tablets, computadores, reproductores de música, netbooks, entre otros. (Báez)

Este sistema operativo es de código libre y trabaja con el lenguaje de programación Java, por lo que se puede crear cualquier aplicación que una persona desee con tan solo conocer conceptos básicos de programación orientada a objetos y conocimientos en dicho lenguaje. Además, brinda la posibilidad de realizar cambios directamente sobre el sistema operativo. (Báez)

Este sistema operativo consta de una tecnología de 5 capas:

1. Kernel de Linux: el sistema operativo está construido sobre el núcleo de Linux (kernel 2.6), está conformado por los drives necesarios para permitir la interacción entre el software y el hardware del dispositivo.
2. Librerías: brindan la posibilidad de manejar diferentes tipos de datos dentro de una aplicación: gráficos en 2D y 3D, contenido HTML, acceso a bases de datos, contenido multimedia, etc. Las librerías están desarrolladas en C o C++.
3. Android Runtime: consiste en una especie de Máquina Virtual de Java llamada *Dalvik Virtual Machine* que permite correr las aplicaciones en los dispositivos, optimizando el procesamiento y la cantidad memoria. Además permite la utilización de las funcionalidades de las librerías de *Java SE*.
4. Framework de Aplicaciones: permite gestionar las funciones básicas del dispositivo al interactuar con la aplicación como: gestión de actividades, gestión de los recursos del dispositivo, gestión de las llamadas, gestión de datos, etc.
5. Aplicaciones: consta de las aplicaciones que vienen preinstaladas en los dispositivos móviles, además de brindar un entorno de desarrollo basado en Java para la creación de una infinidad de aplicaciones. (Pérochon, 2012)

Fig. 1.5. Tecnología de Capas Android

Fuente: http://androideity.com

Conclusiones del capítulo

Conociendo el significado y las principales funcionalidades que brinda la dómotica, el comportamiento que tiene un sistema basado en la arquitectura cliente/servidor y comprendiendo los lenguajes y los requerimientos sobre los que trabajan las plataformas de desarrollo móvil iOS y Android, se puede tener una idea clara de la forma en que se integrarán las distintas tecnologías mencionadas, estas serán explicadas a detalle en los próximos capítulos.

2. CAPÍTULO 2: Análisis de situación actual

Introducción

En este capítulo se expondrá la estructura global y la funcionalidad que el sistema domótico brindará. Se explicará cómo está conformado internamente, es decir las diferentes capas que lo constituyen y cómo estas se interrelacionan entre sí para crear un sistema domótico. También se describirán de forma muy general los módulos en los que se divide dicho sistema.

2.1.Estructura del sistema

El sistema domótico está conformado por 4 capas:

- Capa de Hardware
- Capa de Servicios
- Capa Middleware
- Capa de Aplicación

A continuación se describe cada una de las capas mencionadas:

Capa de Hardware: está formada por los distintos dispositivos eléctricos y componentes electrónicos necesarios para el funcionamiento del sistema domótico.

Capa de Servicios: consta del diseño arquitectónico seleccionado para estructurar el sistema, en este caso el diseño que se utilizará es el antes

mencionado modelo cliente – servidor. En esta capa se encuentra el servidor junto con los servicios disponibles que ofrecerá a los clientes.

Capa Middleware: esta capa provee transparencia a los clientes para el uso de las aplicaciones a través de servicios comunes, brindando así interfaces estandarizadas. Es un software que sirve de puente para facilitar las conexiones y envío de mensajes entre diferentes aplicaciones clientes y el servidor; de esta manera proporciona una interfaz de programación de aplicaciones (API) sencilla que oculta al usuario la complejidad y las diferencias entre los distintos sistemas operativos, plataformas de desarrollo y conexiones que forman parte del sistema. (Tarkoma, 2009)

Capa de Aplicación: está formada por todos los clientes (aplicaciones) que van a consumir los servicios disponibles en el lado del servidor, en este caso los clientes serán las aplicaciones móviles que se desarrollarán en las plataformas iOS y Android, las cuales realizarán las peticiones a los servicios del servidor.

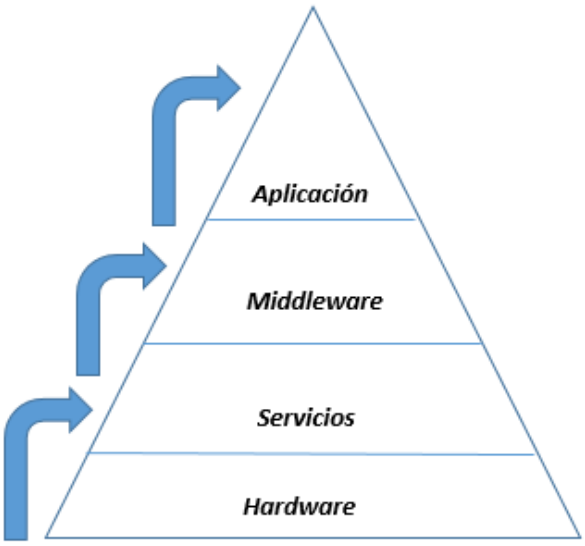


Fig.2.1 Capa Sistema Domótico

Fuente: Los autores.

2.2.Funcionalidad

El sistema se dividirá en dos subsistemas diferentes que se intercomunicarán entre sí para conformar un sistema global.

El primero consta de un sistema de iluminación que será controlado a través de un dispositivo móvil inteligente con conexión a internet, de manera que desde cualquier lugar del hogar en el cuál se disponga de dicha conexión, se podrá acceder a la aplicación que estará conectada con las lámparas del hogar haciendo que las mismas sean controladas según las necesidades del usuario. Este sistema consta de 3 módulos:

Módulo de control: el usuario podrá encender y apagar las lámparas de su hogar en el instante que desee.

Módulo de Simulación de Presencia: podrá programar una simulación de presencia en caso de abandonar su hogar por un periodo de tiempo prolongado, como por ejemplo al realizar un viaje de negocios o de vacaciones. De esta manera, las luces se encenderán y apagarán automáticamente de acuerdo al horario programado por el usuario, creando así una presencia virtual en la misma sin necesidad de mover un solo dedo.

Módulo de Bitácora: el sistema enviará notificaciones mediante correo electrónico cada vez que la lámpara sea encendida o apagada.

El segundo consta de un sistema de vigilancia con cámaras, estas cámaras serán instaladas y se podrá visualizar en tiempo real lo que está sucediendo dentro del hogar desde un dispositivo móvil.

Ambos sistemas serán controlados centralizadamente a través de una sola aplicación, que funcionará en un dispositivo móvil inteligente con conexión a internet, como ya se mencionó anteriormente. Dichos sistemas

conformarán un solo sistema global que se encargará de brindar seguridad y ayudará a sus usuarios a tener mayor control sobre sus hogares. De esta manera se integrará un hogar al mundo tecnológico.

Conclusiones del capítulo

Conociendo las capas en las que se divide la estructura interna del sistema y su respectiva funcionalidad, además de las tareas que realizará cada uno de los módulos del sistema de iluminación, se puede preparar una propuesta tecnológica que satisfaga las funcionalidades expuestas en este capítulo.

3. CAPÍTULO 3: Propuesta tecnológica

Introducción

En este capítulo se encontrará una solución que satisfaga cada una de las funcionalidades del sistema domótico que fueron expuestas en los capítulos anteriores, se modelará el sistema y su comportamiento a través de casos de uso, se analizará distintas opciones de cirucitos y dispositivos electrónicos, de las cuales se describirá a detalle la que será implementada para el control del encendido y apagado de la lámpara, se describirá la estructura, configuración y el proceso de creación de la base de datos necesaria para guardar el cambio de estado de la lámpara y la generación de la bitácora. Además se explicará paso a paso la configuración del middleware, del web service y la implementación de los servicios que controlarán el sistema de iluminación.

3.1.Análisis del sistema:

3.1.1. Levantamiento de Información y Requisitos para el Sistema Domótico.

Se realizará una encuesta a 2 empresas de domótica que existen en la ciudad de Cuenca:

Dommot

Sodel

La encuesta realizada a la empresa Dommot se muestra a continuación:

Encuesta para el levantamiento de requisitos para la realización del trabajo de titulación: "PROTOTIPO DE UN SISTEMA DOMÓTICO PARA CONTROLAR ILUMINACIÓN Y CÁMARAS DE SEGURIDAD MEDIANTE DISPOSITIVOS MÓVILES"

Nombre de la Empresa: dommot: Domótica y Automatización

Nombre del Propietario: Christian Guillén B.

1. **¿Qué productos y/o servicios ofrece su empresa?**

   Diferentes paquetes de sistemas de iluminación

   Sistemas de dimerización

   Sistema de vigilancia con cámaras de seguridad

   Control de Audio y Video

   Control de Persianas

   Programación personalizada de diferentes escenas
   Paneles de control centralizados
   Control de acceso para puertas
   Inmótica

2. **¿Qué productos y/o servicios son los más solicitados?**

   Sistemas para el control de la iluminación

Sistema de vigilancia con cámaras de seguridad
Sistemas de dimerización.

3. **¿Su empresa dispone de productos y/o servicios que ofrecen seguridad?**

   Si, todos los productos y sistemas que ofrece mi empresa implementan seguridad con autenticación personalizada y/o alarmas.

4. **¿Qué ventajas obtendrá el usuario al obtener sus productos y/o servicios?**

   Comodidad, confort, seguridad, control centralizado y lo más importante que cada módulo instalado es independiente, es decir, en caso de que uno de los módulos sufra algún daño, solo tenemos que cambiar ese módulo pero el sistema sigue funcionando normalmente con los otros módulos. Además los módulos instalados disponen de una memoria donde se guardan todos los eventos realizados por los usuarios, la hora y fecha en que se produjeron, etc. Los servicios pueden ser controlados por una aplicación móvil.

5. **¿Cuál es el producto y/o servicio más básico que ofrece y cuál es su costo?**

   El control de iluminación de las luces en un hogar, con un costo desde $ 1.900 dólares.

6. **¿En cuanto a versatilidad y facilidad de acoplamiento de sus productos y/o servicios qué soluciones puede escoger el usuario?**

   El usuario debe escoger la solución que más le convenga entre los paquetes predefinidos que nosotros ofrecemos, por ejemplo tenemos paquetes de iluminación que controlan 6, 10, o 12 luces, otros que vienen con sistema de vigilancia y seguridad, otros que permiten el control de 6 luces junto con la instalación de alarmas, otros paquetes que solo permiten control de audio y video, etc. Además podemos armar una solución domótica de acuerdo al presupuesto del usuario.

7. **¿Qué nivel de satisfacción ha obtenido por parte de los usuarios que han consumido sus productos y/o servicios?**

   Somos una empresa relativamente nueva, que estamos trabajando en algunos proyectos en los cuales el cliente se está sintiendo satisfecho con las soluciones que les brindamos. Ahora estamos centrados en controlar toda la iluminación de un edificio.

8. **¿Cuál es el proceso de instalación de las soluciones domóticas que ofrece su empresa?**
   Los contratos se realizan con una empresa aseguradora o con un arquitecto para implementar el cableado en el momento de la construcción del hogar, y la instalación consiste en tender el cable desde cada dispositivo a controlar hacia la unidad de control centralizada donde se encuentran los módulos, en el caso de un hogar nos tomará de 1 a 2 días, y en el caso de 1 edificio entre 2 o 3 meses. Por otra parte, podemos instalar nuestros equipos en casas que no sean costosas necesariamente.

A continuación se muestra la encuesta realizada a la empresa Sodel:

Encuesta para el levantamiento de requisitos para la realización del trabajo de titulación: "PROTOTIPO DE UN SISTEMA DOMÓTICO PARA CONTROLAR ILUMINACIÓN Y CÁMARAS DE SEGURIDAD MEDIANTE DISPOSITIVOS MÓVILES"

Nombre de la Empresa: SODEL: Soluciones Domóticas y Eletrónicas

Nombre del Propietario: Ing. Adrián Ortega.

1. **¿Qué productos y/o servicios ofrece su empresa?**

   Automatización de iluminación
   Control de cortinas y persianas
   Control de aire acondicionado o calefacción
   Control e instalación de iluminación LED
   Control de audio centralizado
   Control de electrodomésticos
   Control de ahorro energético
   Sistemas de seguridad con cámaras y dispositivos IP para el control de acceso.
   Video Vigilancia
   Instalación de dispositivos de aviso en caso de intrusión o avería (robos, incendios, fugas de gas, etc).

2. **¿Qué productos y/o servicios son los más solicitados?**

   Automatización de iluminación
   Video Vigilancia
   Control e instalación de iluminación LED

3. **¿Su empresa dispone de productos y/o servicios que ofrecen seguridad?**

   Si, los sistemas de seguridad con cámaras y dispositivos IP para el control de acceso y la instalación de dispositivos de aviso en caso de intrusión o avería o robo.

4. **¿Qué ventajas obtendrá el usuario al obtener sus productos y/o servicios?**

   Confort, entretenimiento, tranquilidad, respaldo y control totalmente centralizado desde un PC o internet. Además nuestros sistemas disponen de una aplicación móvil.

5. **¿Cuál es el producto y/o servicio más básico que ofrece y cuál es su costo?**

   El módulo de control de iluminación LED que permite encender, apagar y dimerizar, 12 luces, con un costo de $ 2.800 dólares.

6. **¿En cuanto a versatilidad y facilidad de acoplamiento de sus productos y/o servicios qué soluciones puede escoger el usuario?**

   El usuario escoge que ambiente de su hogar quiere controlar, luego se analiza la cantidad de luces o dispositivos que desea controlar, y si requiere video vigilancia y seguridad para el servicio escogido.

7. **¿Qué nivel de satisfacción ha obtenido por parte de los usuarios que han consumido sus productos y/o servicios?**

   Muy buena, los clientes se sienten muy contentos cuando pueden controlar la iluminación LED, incluso brindamos la posibilidad de cambiar de color la iluminación, eso siempre les gusta.

8. **¿Cuál es el proceso de instalación de las soluciones domóticas que ofrece su empresa?**

   Colocar la unidad central en el hogar, realizar el cableado en el momento de la construcción del hogar, o en caso de un hogar ya construido utilizando canaletas para llevar el cable desde el punto de control hasta los módulos correspondientes.

Una vez realizadas las encuestas se obtuvieron los siguientes resultados:

Según lo expresado por ambas empresas los módulos más solicitados para ser implementados en los hogares son el módulo de control de iluminación y el sistema de video vigilancia.

Ambas empresas ofrecen una aplicación móvil para el control de las soluciones domóticas desarrollada por la marca que importa los dispositivos controladores, la cual no puede ser modificada ni personalizada, además no permite acceso a su código fuente.

En base a estos resultados, se ha decidido implementar el control de iluminación y video vigilancia a través del uso de cámaras IP, además de realizar una aplicación móvil sobre las plataformas IOs y Android que permitan el control de dicho sistema según las necesidades especificas del usuario.

3.1.2. Análisis Orientado a Objetos del problema (AOO).

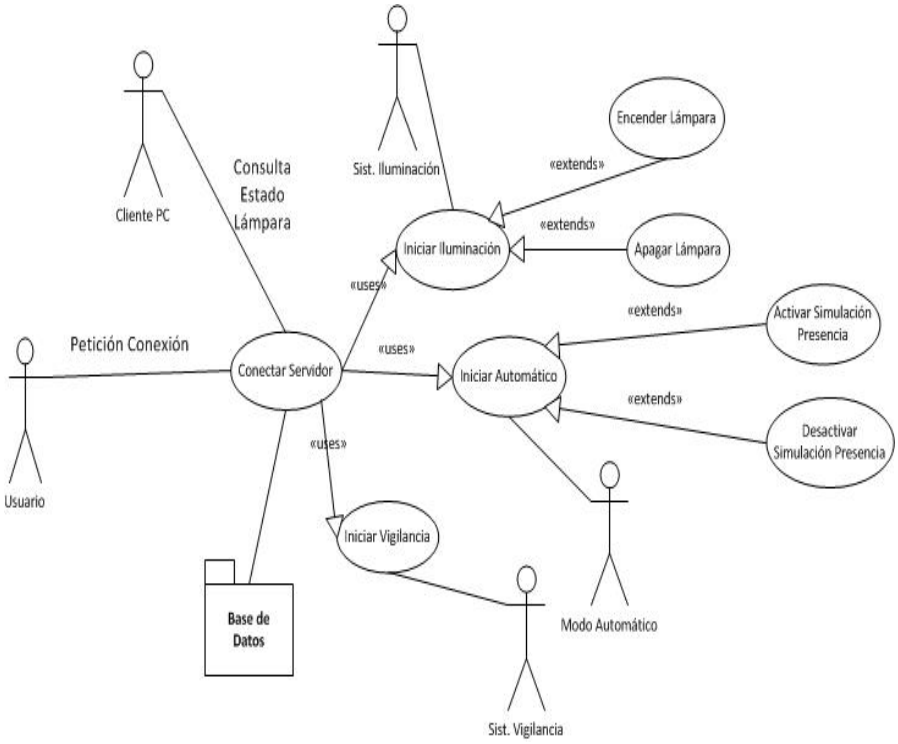


Fig.3.1.1 Modelo Casos de Uso

Fuente: Los Autores.

Descripción de Actores

Usuario: Representa las aplicaciones cliente de los dispositivos móviles inteligentes.

Cliente PC: Se encarga de obtener el estado en que se encuentra la lámpara realizando una consulta al servidor.

Sist. Iluminación: Se encarga de encender o apagar la lámpara.

Sist. Vigilancia: Se encarga de mostrar la imagen que captura la cámara IP.

Modo Automático: Se encarga de activar o desactivar la simulación de presencia en base a las fechas y las horas especificadas.

Descripción de casos de uso

Caso de uso 1	Conectar Servidor
Actor:	Usuario
***Descripción*:**	Permite que los usuarios (dispositivos moviles) se conecten al servidor que controla el sistema domótico.
Caso de uso 2	Iniciar Iluminación
Actor:	Sist. Iluminación
***Descripción*:**	Permite que los usuarios conectados al servidor puedan encender y apagar la lámpara.
Caso de uso 3	Iniciar Automático
Actor:	Modo Automatico
***Descripción*:**	Permite que los usuarios conectados al servidor puedan activar o desactivar la simulación de presencia.
Caso de uso 4	Iniciar Vigilancia
Actor:	Sist. Vigilancia
Caso de uso 5	Encender Lámpara
Actor:	Permite que los usuarios puedan observar la imagen que transmite la cámara IP.
Caso de uso 6	Apagar Lámpara
Actor:	Permite encender la lámpara.
Caso de uso 7	Activar Simulación Presencia
Actor:	Modo Automático
***Descripción*:**	Permite que la simulación de presencia sea activada.
Caso de uso 8	Desactivar Simulación Presencia
Actor:	Modo Automático
***Descripción*:**	Permite que la simulación de presencia sea desactivada.
REQUISITOS ASOCIADOS	
R.8.1 El usuario debe haber iniciado el Modo Automático.	

de alternativas de microcontroladores y dispositivos electrónicos.

Con el fin de encontrar una solución para el circuito controlador del encendido y apagado de la lámpara del sistema de iluminación, a continuación se analizan 2 alternativas distintas, el PIC18f4550 y la placa hardware Arduino.

PIC18f4550: es un microcontrolador de la familia PIC18XXXX. Su característica más importante es la capacidad de manejar periféricos de comunicación avanzada como es el caso del puerto USB 2.0. Otras de sus características son:

Posee 3 interrupciones externas.
Tiene 4 timers (TMR0 A TMR3).
Soporta hasta 13 canales analógico/digital (A/D) con módulo conversor.
Tiene 20 fuentes de interrupción.
48 MHz de frecuencia de operación.
Conjunto de 75 instrucciones.
Arquitectura Harvard con tecnología RISC.
Soporta control, interrupciones y transferencias isócronas.
Permite la conexión de hasta 32 endpoints.

(Inc, 2009)

La distribución de los pines de este microcontrolador se muestra en la siguiente figura:

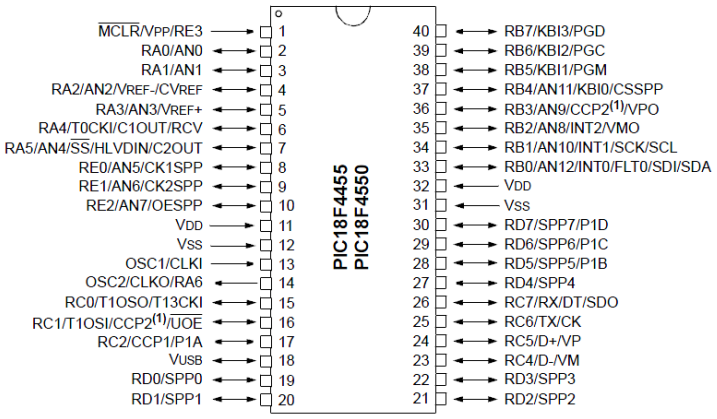


Fig.3.1.2. Pines PIC18F4550

Fuente: Microchip Technology Inc.

Arduino: es una placa de circuito impreso que contiene un microcontrolador y un diseño definido de circuitería interna que permite la conexión de varios actuadores, periféricos y sensores de manera fácil y rápida. (Torrente, 2013)

Existen distintos tipos de placas Arduino que difieren por su tamaño físico, el microcontrolador utilizado, la cantidad de pines que poseen y el tamaño de su memoria.

Fig.3.1.3 Placa Arduino

Fuente:http://www.electronicaestudio.com/arduino_productos.htm

Arduino dispone de un entorno de desarrollo de código abierto que permite transferir las instrucciones (programa) a la memoria del microcontrolador. La transferencia del programa se realiza a través de un cable USB conectado entre el computador y la placa. El entorno de desarrollo es multiplataforma, ya que puede ser instalado sobre sistemas operativos Windows, Linux y MacOS. El lenguaje de programación Arduino posee una gran cantidad de reglas sintácticas, estructuras de control, bloques condicionales y bucles que se definen de manera muy similar a otros lenguajes de programación debido a que Arduino, internamente está basado en C/C++. Además ofrece funciones intuitivas y simples de comprender para las instrucciones específicas que necesitan ser grabadas en el microcontrolador de la placa. (Torrente, 2013)

Cuando se requiera desarrollar un software específico que necesite intercambiar datos entre el computador y la placa, el software puede ser desarrollado en cualquier lenguaje de programación como Java, C++, Python, etc., sin causar problemas de compatibilidad con el entorno de desarrollo Arduino. (Torrente, 2013)

3.1.4. Análisis de Protocolos de Comunicación.

USB (Universal Serial Bus – Bus de Serie Universal):

Es una especificación abierta o un tipo de puerto creado en 1996 por 7 empresas (IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC). Que permite la conexión y transmisión de datos entre diversos dispositivos externos, tales como, teclado, mouse, impresora, teléfonos móviles, cámaras fotográficas o de video, discos duros externos,

reproductores multimedia, tarjetas de sonido y de video, etc… y un ordenador.

Está basado en una arquitectura serial. USB es una interfaz de entrada/salida que trabaja mucho más rápido que otro tipo de puerto serial estándar.

Unas de las razones por las cuales se introdujo en este tipo de puertos la tecnología serial es porque:

- Brinda una velocidad de reloj más alta y sin tanto retraso como lo hacía la arquitectura en paralelo.

- El cableado serial es más económico.

USB posee una característica muy importante que lo hace aún más fácil de usar, denominada *Plug–n-Play*, quiere decir que tan solo con conectar al ordenador cualquier dispositivo, este lo reconocerá y funcionara inmediatamente sin necesidad de reiniciar el mismo o realizar algún tipo de reconocimiento manual.

Tipos de Conectores USB:

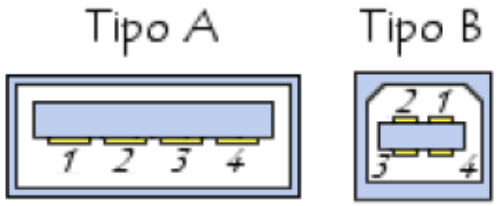


Fig. 3.1.4. Tipos de conectores USB

Fuente: http://es.kioskea.net/contents/407-usb-bus-de-serie-universal

Conector Tipo A:

Utilizados para dispositivos que no requieren un gran ancho de banda (teclados, ratón, webcam, etc…).

Conector Tipo B:

Utilizados más comúnmente para dispositivos de alta velocidad (Discos duros externos).

Tipos de Transferencia:

Transferencia de Control: Es utilizada en caso de que el software realice una petición/respuesta de comunicación.

Transferencias Isócronas: Esta transferencia utiliza comunicaciones periódicas entre controlador y dispositivo, es decir su tiempo de comunicación se realiza entre intervalos de un mismo tiempo.

Transferencias de Interrupción: “Son datos pequeños, no muy frecuentes, que provocan la espera de otras transferencias hasta que son realizadas”. (alvarez, 2004)

Transferencias de Volumen: Se utiliza todo el ancho de banda de la comunicación, ya que transmite paquetes de gran tamaño.

Protocolo de Comunicación:

“Las operaciones entre el dispositivo USB y el controlador USB del PC, se realizan a través de un mecanismo denominado cola de transferencia”. (alvarez, 2004)

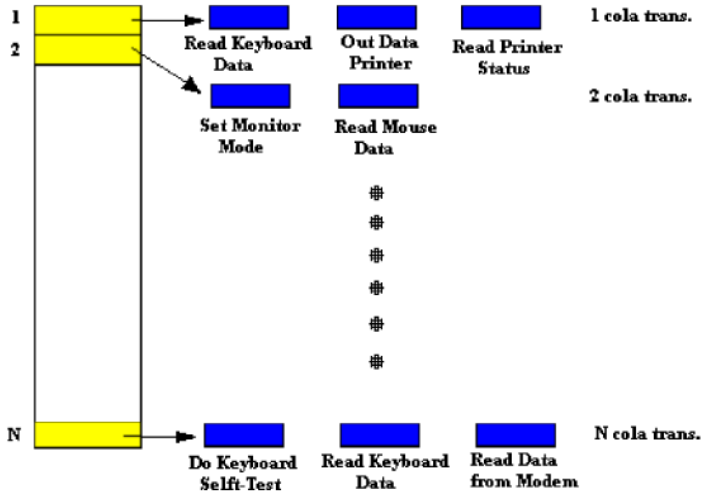


Fig. 3.1.5 Protocolo de Comunicación

Fuente: http://catarina.udlap.mx

“Antes de que el controlador inicie una operación con un dispositivo, el driver USB, coloca la petición de operación en la cola. El controlador inspecciona la cola de peticiones de transferencia y éstas son llevadas a cabo de forma secuencial "FIFO" (First In First Out, o en español, Primero en Entrar Primero en Salir), o sea, por orden de entrada.” (alvarez, 2004).

3.1.5. Modelo del Sistema Domótico

En el siguiente gráfico se puede observar el modelo cliente/servidor que utilizará el sistema domótico, en el lado del cliente se encuentran las aplicaciones de los dispositivos móviles conectadas con el servidor a través del middleware GlassFish, mientras que en el lado del servidor, éste está enlazado con una base de datos MySQL para una correcta sincronización entre la interfaz de usuario de los dispositivos móviles y el circuito controlador que permitirá el encendido y apagado de la lámpara. Además se puede observar que se utilizará la placa Arduino para la comunicación digital/analógica entre el servidor y el circuito controlador.

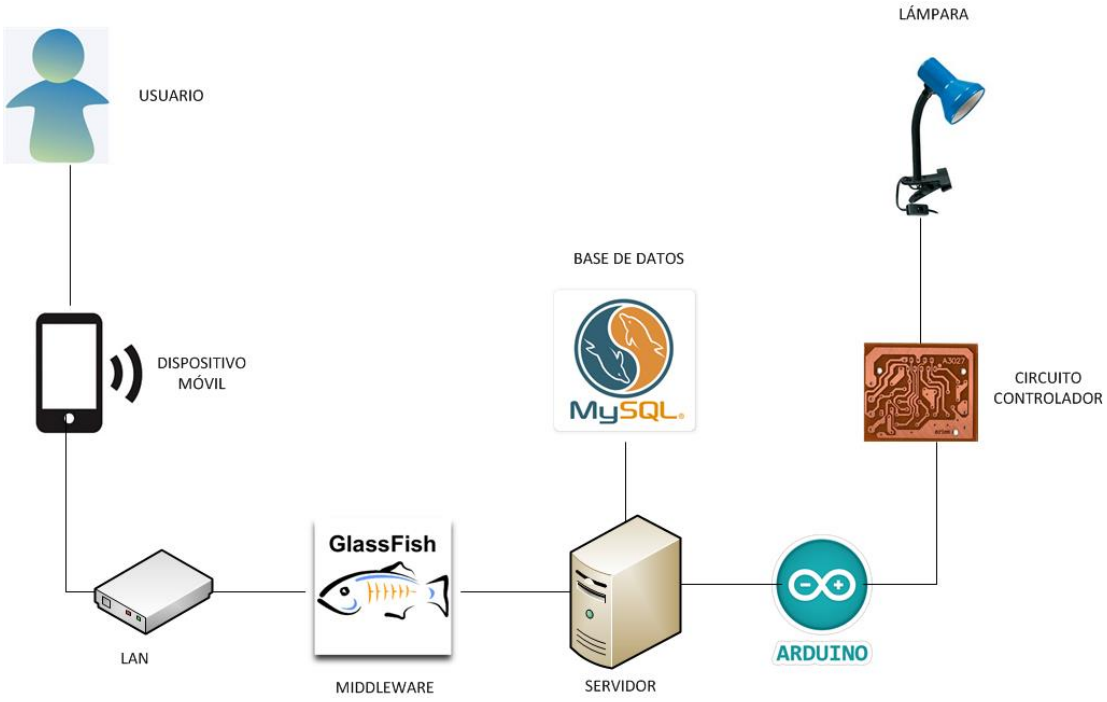


Fig.3.1.6. Modelo Sistema Domótico

Fuente: Los Autores.

3.2. Hardware y Dispositivos Electrónicos:

3.2.1. Diseño del Circuito Controlador.

Para el diseño del circuito encargado de leer el estado de la luz, a más de encenderla y apagarla se utilizó la herramienta Proteus para realizar su respectiva simulación, a continuación se presenta el mismo:

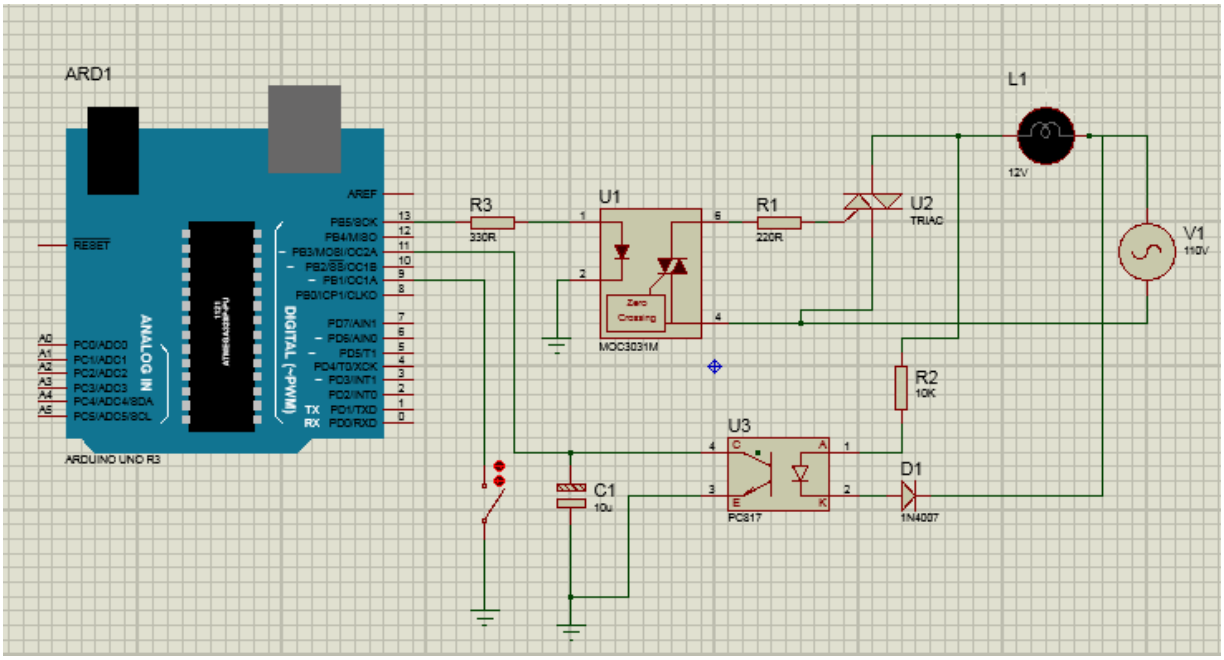


Fig.3.1.7. Simulación Circuito Controlador

Fuente: Los Autores.

Este circuito está diseñado para tener una conexión entre el PC (ordenador) y la lámpara, las funciones que cumple el mismo son las siguientes:

- Encender y apagar la lámpara.
- Leer el estado de la lámpara.
- Transmitir los datos al computador sobre el estado de la lámpara y el estado del switch de la misma.

Este circuito está dividido en 2 partes, la primera es, una placa de circuito impreso prediseñada (Arduino) y la segunda, otra placa de circuito impreso al cual se lamará Circuito controlador, diseñado propiamente para las funciones específicas de este proyecto. Ambas placas se comunican entre sí realizado diferentes funciones, que a continuación se explican por separado:

Arduino: Se encarga de la comunicación intermedia entre el ordenador y el circuito controlador a través del puerto USB, es decir esta placa recibe los datos que vienen desde el ordenador y los transmite al circuito controlador y viceversa, recibe la lectura de datos del estado de la lámpara que realiza el circuito controlador y los envía hacia el ordenador.

Circuito Controlador: Este circuito se encarga de 2 funciones, enviar las señales que recibe desde el Arduino a la lámpara para encenderla y apagarla según sea el caso y lee el estado del switch de la lámpara, para enviar este dato de regreso a la placa Arduino.

3.2.2. Implementación y Construcción.

A continuación se describirán cada uno de los elementos que se utilizarán en la construcción del circuito controlador mencionado anteriormente:

MOC3021

Fig.3.1.8. Moc3021

Fuente: http://www.idelectrónica.com.mx

Es un opto acoplador, es decir, en su interior posee un fototransistor que es activado por medio de un led infrarrojo con el fin de mantener separado el circuito de carga y el circuito de control. Esto permite que el haz de luz transmita una señal de un circuito a otro sin necesidad de tener una conexión eléctrica. (Inc, 2009) A continuación se muestra el diagrama lógico y la distribución de pines correspondiente a este elemento:

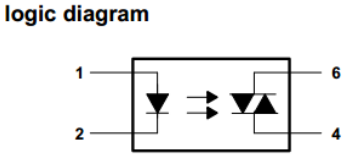


Fig.3.1.9. Diagrama Lógico moc3021

Fuente: http://pdf.datasheetcatalog.com/

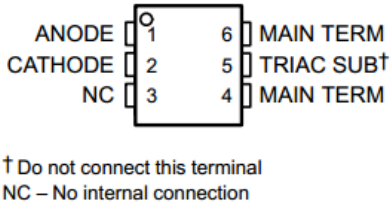


Fig.3.2.1 Pines moc3021

Fuente: http://pdf.datasheetcatalog.com/

PC817

Fig.3.2.2. PC187

Fuente: http://www.electronicoscaldas.com

Es un opto acoplador que contiene un haz de luz infrarrojo acoplado ópticamente a un fototransistor. (Inc, 2009) A continuación se muestra la distribución de sus pines:

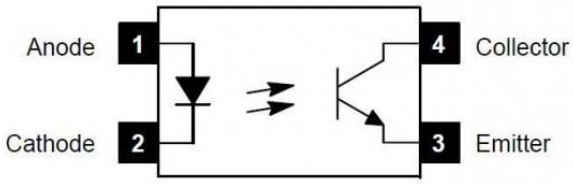


Fig.3.2.3. Pines PC187

Fuente: http://www.electronicoscaldas.com

TRIAC BT138

Fig.3.2.4 Pines PC187

Fuente: http://www.haopin.com

Es un elemento semiconductor que consta de tres terminales, es utilizado para regular el flujo de corriente requerido por una carga, el flujo es bidireccional y permite bloquear por inversión de voltaje cuando este es menor al valor de mantenimiento. El disparo puede realizarse mediante la compuerta positiva o negativa. (Inc, 2009) A continuación se muestra la distribución de sus pines:

PINNING - TO220AB

PIN	DESCRIPTION
1	main terminal 1
2	main terminal 2
3	gate
tab	main terminal 2

PIN CONFIGURATION

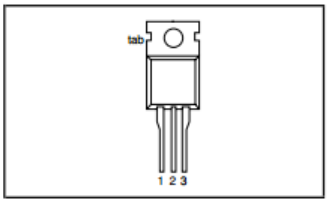


Fig.3.2.5 Pines PC187

Fuente: http://www.electronicoscaldas.com

Diodo 1N4007

Fig.3.2.6. Pines PC187

Fuente: http://www.parts-ecpress.com

Es un elemento semiconductor utilizado para rectificar la corriente alterna, el cual permite el paso de la corriente en un sentido y lo bloquea en el otro. (Inc, 2009)

Una vez que se conocen los elementos que van a ir en la placa impresa del circuito controlador, se realiza su respectivo diagrama electrónico, donde se podrá observar la posición de sus componentes y cómo estarán conectados, para esto se utilizará el software *Cad Soft EAGLE:*

Fig.3.2.7 CadSoft EAGLE

Fuente: http://www.blog.adafruit.com

EAGLE, siglas en inglés de *Easily Applicable Graphical Layout Editor*, es un software para diseñar diagramas y PCBs (Placas de Circuito Impreso) que realiza el enrutamiento de las pistas que conectarán los elementos de manera automática, esta característica se conoce como autoenrutamiento. (CadSoft Computer, 2011)

En la siguiente figura se muestra el diagrama electrónico generado en el software *CadSoft EAGLE* donde se puede observar la posición de cada componente:

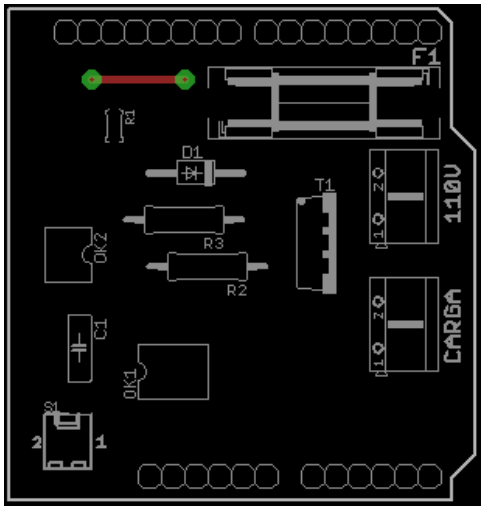
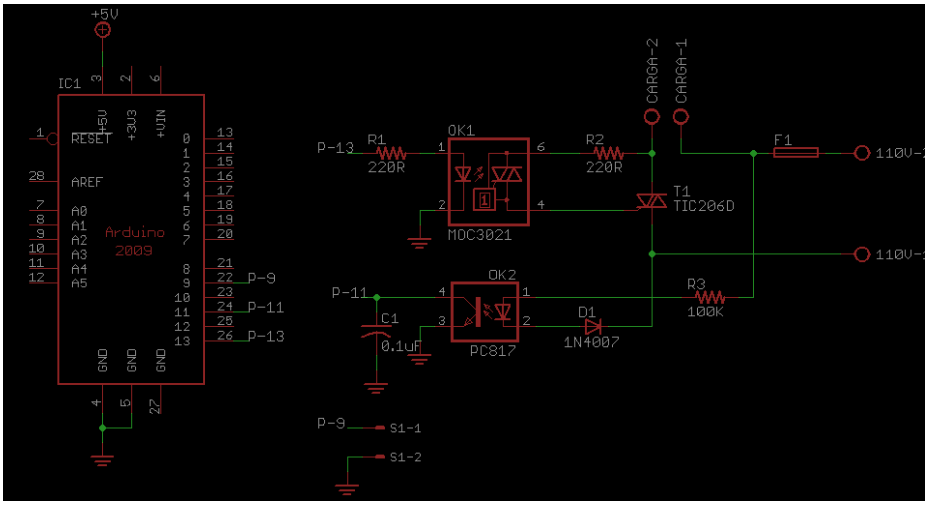


Fig.3.2.8 Diagrama Electrónico

Fuente: Los Autores

A continuación se observa el diagrama con sus c

o



interconectados:

Fig.3.2.9 Diagrama Electrónico

Fuente: Los Autores

Finalmente el software crea de manera automática las pistas que conectarán los componentes. A continuación se observa el diagrama que generará la placa de circuito impreso y la propia placa ya producida:

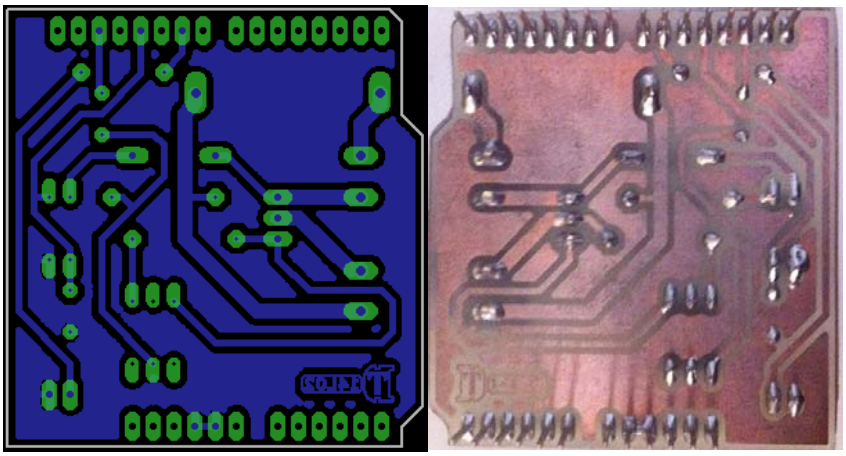


Fig.3.3.1 Diagrama y Placa de Circuito Impreso

Fuente: Los Autores

La placa de circuito impreso producida, será el circuito controlador que deberá ser colocado sobre el Arduino como se observa en la siguiente figura:

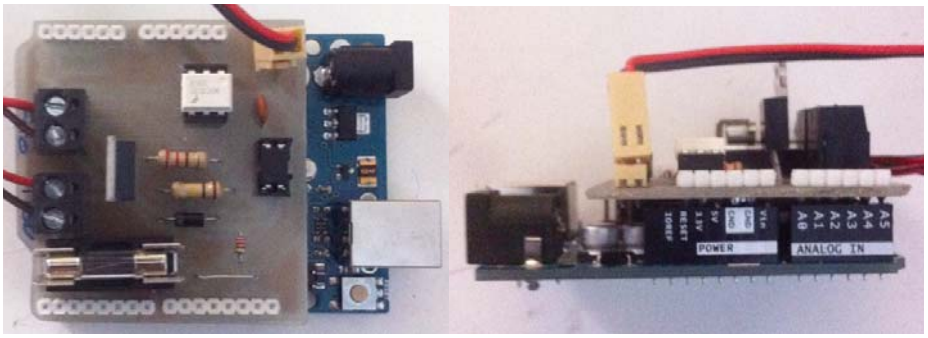


Fig.3.3.2. Circuito Controlador

Fuente: Los Autores

A continuación se explicará la codificación que se realizó en el Arduino para que pueda realizar la comunicación entre el circuito controlador y el ordenador (servidor), dicha codificación se realizó sobre el entorno de desarrollo propio de Arduino:

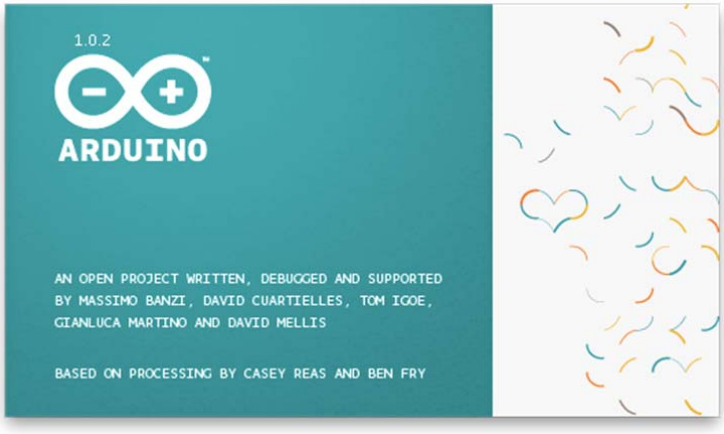


Fig.3.3.3. Entorno de Desarrollo Arduino

Fuente: Los Autores

Primero se declaran las variables de tipo primitivo que se utilizarán en la implementación de los métodos:

```
int input;
int output;
boolean flag = false;
boolean flagEsc = false;
```

Dentro de la función *setup* se inicializan las variables, el modo en que van a trabajar los pines que se van a utilizar, y las librerías necesarias. Esta función se ejecuta una sola vez cuando el programa es ejecutado (Arduino, 2015):

```
void setup()
{
  //Se define el pin número 13 como
  salida
  pinMode(13, OUTPUT);
```

```
  //Se define el pin número 9 como
  entrada
  pinMode(9, INPUT);


  //Se activa la salida digital del pin
  número 9 con un "1" lógico
  digitalWrite(9, HIGH);

  //la función begin de la clase Serial
  establece la velocidad con que se va a
  realizar la transmición serial en 9600
  bits por segundos
  Serial.begin(9600);

}
```

Dentro de la función *loop* se define un proceso cíclico que lee constantemente el pin de entrada (9) del Arduino, donde llegarán los datos enviados desde el ordenador indicando si se desea encender o apagar la lámpara. Si encuentra un cambio en el estado, este se ve reflejado en el pin 13 que es el encargado de encender o apagar la lámpara y también notifica enviando un 1 en caso de encendido o un 0 en caso de apagado hacia el puerto serie en el que se encuentra conectado el Arduino, es decir hacia el ordenador.

```
void loop(){
  if (Serial.available()>0){

    input=Serial.read();

    if (input=='1')
    {
      digitalWrite(13, HIGH); //Si el
      valor de input es 1, se enciende el
      led
      //Serial.write(1);
```

```
    }

    else

    {
      digitalWrite(13, LOW); //Si el
      valor de input es diferente de 1,
      se apaga el LED
      //Serial.write(0);

    }

  }

  if(digitalRead(9)==0 & flag == false)
    {
      digitalWrite(13, LOW);
      Serial.write(0);
      flag = true;
    }
      else if(digitalRead(9)==1 & flag ==
      true)
    {
      digitalWrite(13, HIGH);
      Serial.write(1);
      flag = false;
    }
    else if (flag == false)
    {
      delay(500);
      Serial.write(0);
    }
    else
    {
      delay(500);
      Serial.write(1);
    }

}
```

3.3. Base de Datos:

3.3.1. Modelo Entidad – Relación.

Una vez identificadas las entidades principales que componen el sistema domótico, se genera el modelo entidad-relación correspondiente al mismo, en él se puede observar que está compuesto de 3 tablas:

La tabla lámpara almacenará el estado de la lámpara cada vez que ésta sea encendida o apagada gracias a su código de identificación *id*.

La tabla simulación almacenará los datos necesarios para cada simulación de presencia que se ingrese en el sistema, es decir la hora de inicio, la hora de finalización y la fecha en la que se realizará la simulación automáticamente.

La tabla bitácora almacenará todos los eventos que se produzcan en la lámpara, es decir, cuando fue apagada o encendida y por cuánto tiempo.

A continuación se muestra el modelo entidad-relación con los campos y las relaciones de las tablas mencionadas.

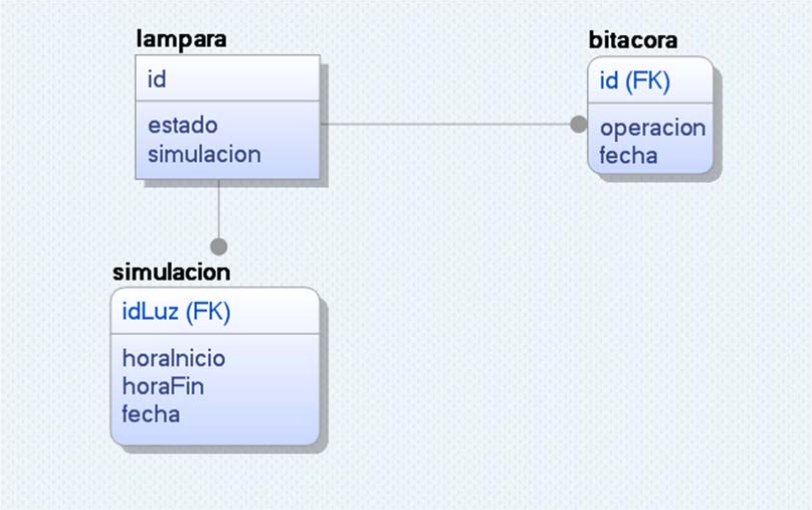


Fig.3.3.4 Modelo E/R Sistema Domótico

Fuente: Los Autores

3.3.2. Creación de la Base de Datos.

Para la creación de la base de datos que gestionará el encendido y apagado de la lámpara se debe realizar los siguientes pasos:

1. Crear la base de datos y asignarle un nombre.

   Se utiliza la siguiente sentencia:

   CREATE DATABASE nombreBaseDeDatos;

   USE nombreBaseDeDatos;

   Ejemplo:

   CREATE DATABASE *domotivedatabase;*

   USE *domotivedatabase*

2. Crear la tabla lámpara.

   Esta tabla será la encargada de almacenar las lámparas disponibles en el sistema diferenciándolas con un "id" o identificador, además almacenará su estado (encendido o apagado) y tendrá un campo que indicará si está activada o desactivada la simulación de presencia en la misma.

```
CREATE TABLE lampara
(

VARCHAR(4) NOT NULL,    id
                        estado
VARCHAR(50) NULL,
                        simulacion
VARCHAR(5) NULL
)
CREATE UNIQUE INDEX XPKlampara ON lampara
(
      id ASC
)
ALTER TABLE lampara
```

```
ADD PRIMARY KEY (id)
```

3. Crear la tabla bitacora.

   Esta tabla será la encargada de almacenar los eventos ocurridos en la lámpara, es decir cada vez que se encienda o apague la lámpara ésta tabla almacenará que evento sucedió junto con el identificador (id) de la lámpara, la fecha y la hora del evento.

```
CREATE TABLE bitacora
(
      id            VARCHAR(4) NOT NULL,
      operacion     VARCHAR(50) NULL,
      fecha         TIMESTAMP NULL
)
CREATE UNIQUE INDEX XPKbitacora ON
bitacora
(
      id ASC
)
ALTER TABLE bitacora
ADD PRIMARY KEY (id)
```

4. Crear la tabla simulación.

   Esta tabla será la encargada de manejar la fecha y horas en la cual la lámpara iniciará y culminará el modo automático o simulación de presencia.

```
CREATE TABLE simulacion
(
      horaInicio      VARCHAR(20) NULL,
      horaFin         VARCHAR(20) NULL,
      fecha           VARCHAR(20) NULL,
      idLuz           VARCHAR(4) NOT NULL
)


CREATE UNIQUE INDEX XPKsimulacion ON
simulacion
(
      idLuz ASC
)

ALTER TABLE simulacion
ADD PRIMARY KEY (idLuz)ALTER TABLE
bitacora
ADD FOREIGN KEY R_1 (id) REFERENCES
lampara (id)
ALTER TABLE simulacion
```

```
ADD FOREIGN KEY R_2 (idLuz) REFERENCES
lampara (id)
```

Los pasos del 1 al 4 recién mencionados se pueden realizar con ayuda de un interfaz web para evitar escribir los scripts de creación.

Creación de la base de datos a través de *phpMyAdmin*:

1. Igresar al entorno phpMyadmin, seleccionar la pestaña *Bases de datos,* luego dirigirse al campo *Crear base de datos*, proceder a nombrar la base de datos y dar click en *Crear.*

Fig.3.3.5 Creación Base de Datos

Fuente: Los Autores

2. Observar que al lado izquierdo de la interfaz, se ha creado corectamente la base de datos y seleccionar la misma.

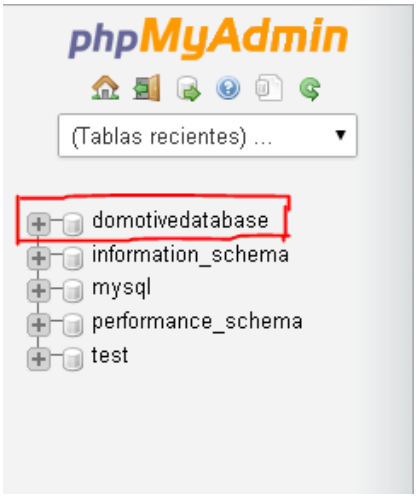


Fig.3.3.6 DomotiveDatabase

Fuente: Los Autores

3. Seleccionar la pestaña *Estructura*

Fig.3.3.7 Estructura Base de Datos

Fuente: Los Autores

4. Proceder a la opción *Crear tabla*, seguido de esto nombrar la tabla que se creará e indicar el número de columnas que contedrá la misma, a continuación presionar *Continuar*.

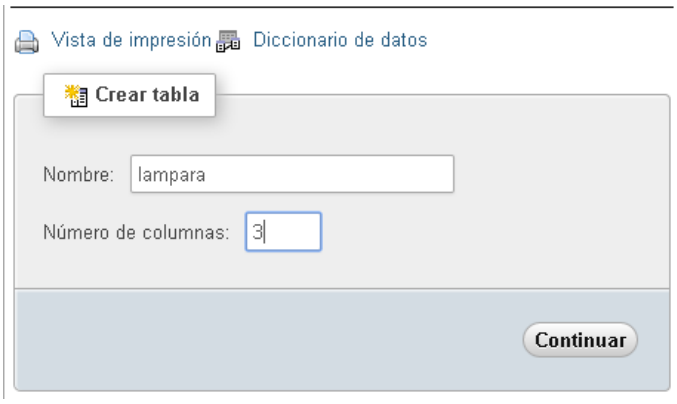


Fig.3.3.8 Creación Tabla

Fuente: Los Autores

5. Especificar los diferentes campos de cada una de las columnas en la tabla, tales como, *Nombre del Campo, Tipo de Dato Longitud del campo, etc…* Seguido de esto presionar *Guardar*.

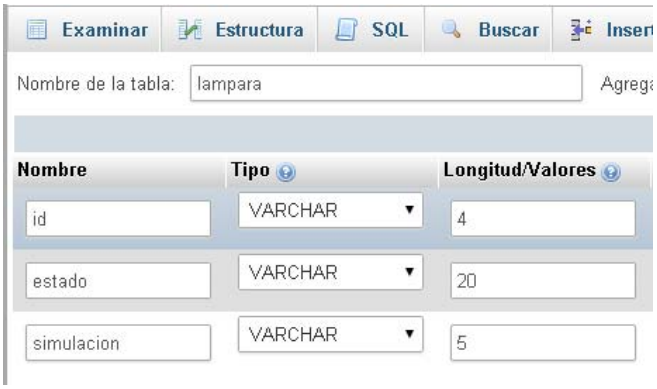


Fig.3.3.9 Campos Tabla

Fuente: Los Autores

6. Repetir el proceso desde el paso 3 para cada tabla que se desee crear.

3.3.3. Documentación.

Se necesita tener documentada la base de datos creada anteriormente, para esto se requerirá de un diccionario de datos, el cual consiste en un conjunto de metadatos, los cuales abarcan las distintas características específicas y lógicas correspondientes a los datos que requiere el sistema domótico, estos metadatos describen el

significado, la composición y el tipo de valores que pueden ser asignados al mismo.

Este diccionario permite comprender la estructura y el flujo que siguen los datos del sistema, de manera que se pueda entender más fácilmente el funcionamiento del mismo.

Para generar el diccionario de datos automáticamente se debe seleccionar la base *domotivedatabase* y seleccionar la opción *Diccionario de datos* como se muestra a continuación:

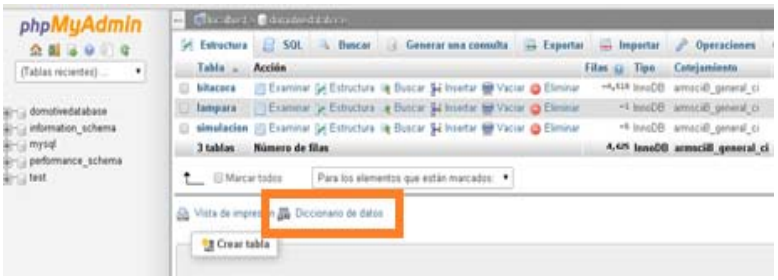


Fig.3.4.1 Diccionario de Datos

Fuente: Los Autores

El documento obtenido correspondiente al diccionario de datos se encuentra en el Anexo B.

3.4. Web Service

A continuación se explicará la arquitectura que tiene el *web service*, la descripción de sus componentes y el funcionamiento de los mismos en sus diferentes capas estructurales. Además se

describirá paso a paso la implementación de dicho *web service* que es el encargado del control de cada módulo que compone el sistema de iluminación.

3.4.1. Arquitectura y Funcionamiento

Un *web service* es una API que permite ofrecer servicios a varios clientes a través de la web, por lo que su arquitectura está basada en el estándar XML (eXtensible Markup Language).

Este lenguaje permite que los clientes llamen a métodos remotos y consuman sus servicios utilizando HTTP como protocolo de comunicación gracias al lenguaje WSDL (Web Services Description Language) que se encarga de especificar el protocolo utilizado para enviar y recibir mensajes junto con la URL necesaria y la descripción de sus procedimientos remotos. (Tarkoma, 2009)

Según (Tarkoma, 2009), la implementación del *web service* está basada en una arquitectura 3-tier, que consta de 3 capas:

1. **Capa de presentación**

   Se encarga de la lógica del usuario, es decir, está formada por todos los clientes que pueden ser ordenadores o dispositivos móviles en donde están alojadas las aplicaciones nativas o aplicaciones web.

2. **Capa de negocio**

Es la máquina donde se encuentra alojado el servidor de aplicaciones que ofrece los servicios disponibles a los clientes.

3. **Capa de datos**

Está formada por la base de datos, el acceso y la gestión de los datos solo puede ser realizada directamente por la capa de negocios.

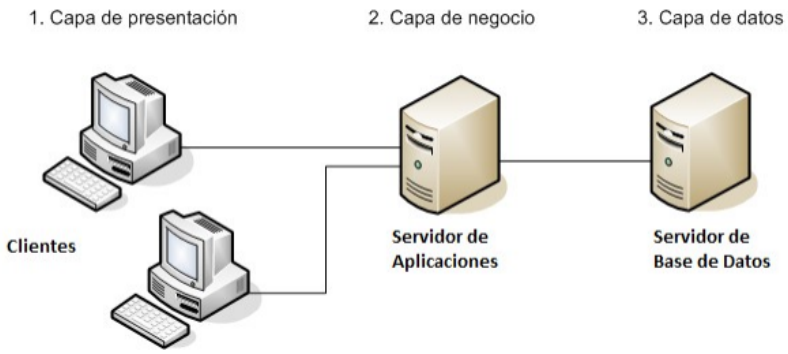


Fig.3.4.2. Arquitectura Web Service

Fuente: http://www.recercat.net/bitstream

A continuación se describe el servidor de aplicaciones (*Middleware*) que se va a utilizar para alojar el servidor del sistema domótico: Glassfish.

El término Glassfish, traducido al español sería algo parecido como “Pez de Cristal”, es el nombre de un pez que realmente existe y vive en agua dulce, su cuerpo es transparente, por lo

que sus huesos son visibles. El nombre fue elegido debido a la transparencia que los creadores querían darle al proyecto, que utiliza una licencia Open Source, concretamente la licencia Common Development and Distribution License (CDDL) v1.0 y la GNU Public License (GPL) v2. (Manchado, 2010)

Glassfish es un servidor de aplicaciones desarrollado por Sun Microsystems que soporta tecnologías como Java EE, JSP, EJSs Servlets, JAX/WS y XML. (Manchado, 2010)

Es un servidor de aplicaciones modular, extensible e integrable ya que permite descargar e instalar únicamente los módulos necesarios para una aplicación específica, lo que permite disminuir el tiempo de arranque del servidor, la cantidad de memoria consumida y el espacio de disco utilizado. Además estos módulos pueden ser instalados y actualizados remotamente sin reiniciar el servidor o utilizados como una librería más dentro de la máquina virtual de java. (Manchado, 2010)

Fig.3.4.3 Glassfish

Fuente: http://javacodegeeks.com/glassfish.jpg

3.4.2. Protocolos

SOAP (Simple Object Access Protocol) es un protocolo de intercambio de mensajes que permite que programas que corren en diferentes sistemas operativos, se comuniquen usando el protocolo HTTP (Hypertext Transfer Protocol), además del protocolo XML (Extensible Markup Languaje).

(Rouse, 2014)

REST (Representational State Transfer) es un estilo de arquitectura de software para sistemas distribuidos tales como la Web.

El término fue introducido en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de HTTP. En realidad, REST se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen como los recursos son definidos y diseccionados. El término frecuentemente es utilizado en el sentido de describir a cualquier interfaz que transmite datos específicos de un domino sobre HTTP sin una capa adicional, como hace SOAP. Estos dos significados pueden chocar o incluso solaparse. Es posible diseñar un sistema software de gran tamaño de acuerdo con la arquitectura propuesta por Fielding sin utilizar HTTP o sin interactuar con la Web. Así como también es posible diseñar una simple interfaz XML+HTTP que no sigue los principios REST, y en su lugar sigue un modelo RPC.

(Marset, 2006)

3.4.3. Implementación.

Para la implementación del *Web Service*, sus respectivas funcionalidades y los métodos que va a ofrecer el servidor a sus clientes se utilizará el entorno de desarrollo Netbeans IDE, la versión 8.0.

Fig.3.4.4. Netbeans IDE 8.0

Fuente: http://phpgranada/glassfish.jpg

Instalación Glassfish

1. Dentro del entorno de desarrollo Netbeans ir a la pestaña *Tools* y seleccionar la opción *Servers.*

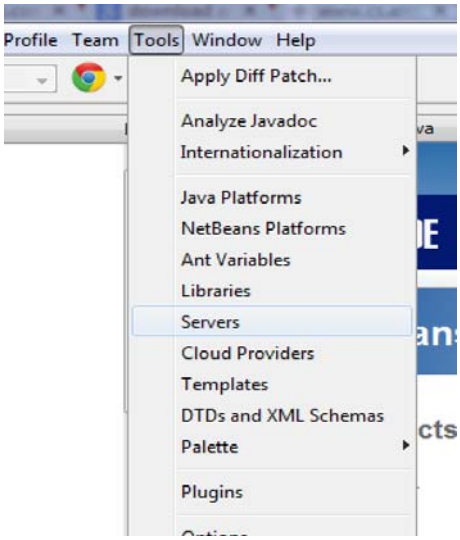


Fig.3.4.5 Servers

Fuente: Los Autores

2. Seleccionar la opción *Add Server*

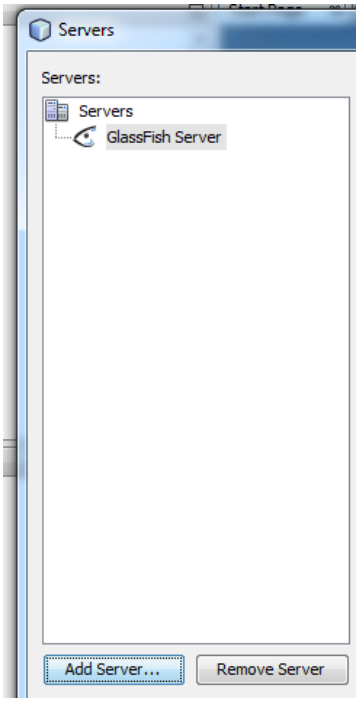


Fig.3.4.6 GlassFish Server

Fuente: Los Autores

3. Escoger el tipo de servidor a utilizar, en este caso seleccionar *Glassfish Server* y asignarle un nombre.

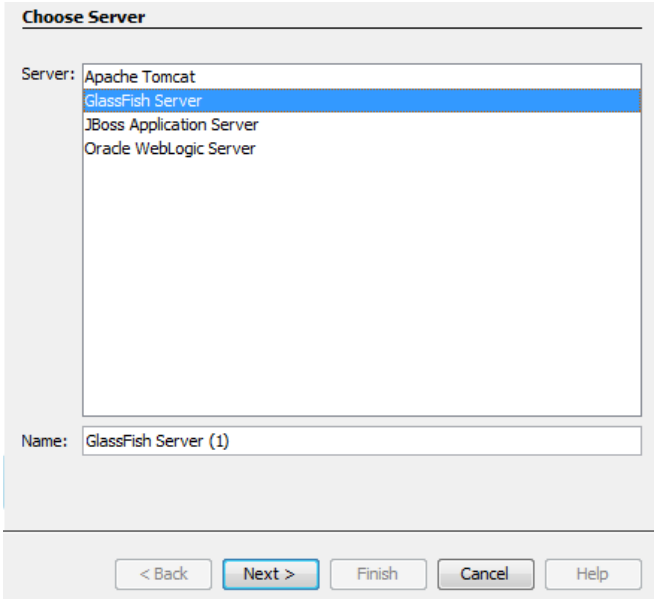


Fig.3.4.7 Glassfish

Fuente: Los Autores

4. Seleccionar la carpeta en dónde se desea descargar y hacer clic en *Download Now.*

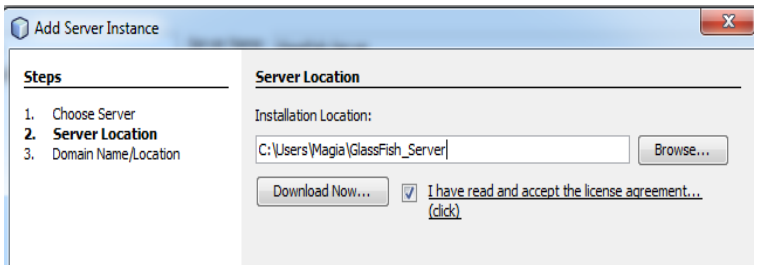


Fig.3.4.8 Ubicación de Instalación

Fuente: Los Autores

5. Una vez descargado, asignar un nombre de dominio para el servidor.

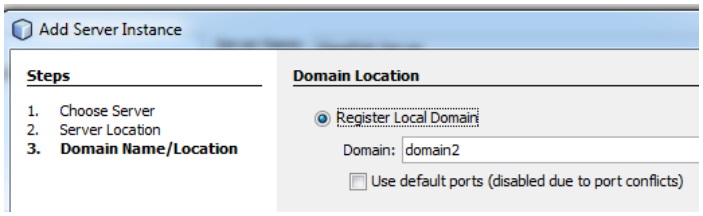


Fig.3.4.9 Dominio

Fuente: Los Autores

6. Hacer clic en finalizar y esperar que el proceso de instalación se complete exitosamente.

7. Dentro de la pestaña *Services* comprobar que se añadió correctamente el servidor *Glassfish* instalado anteriormente.

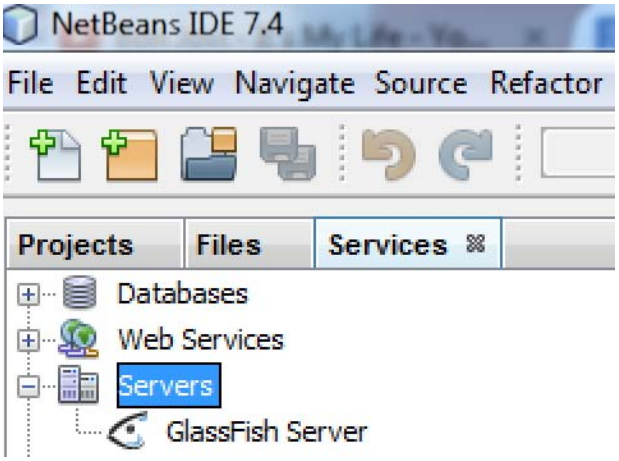


Fig.3.5.1 Instancia Servidor

Fuente: Los Autores

8. Iniciar el servidor *Glassfish.*

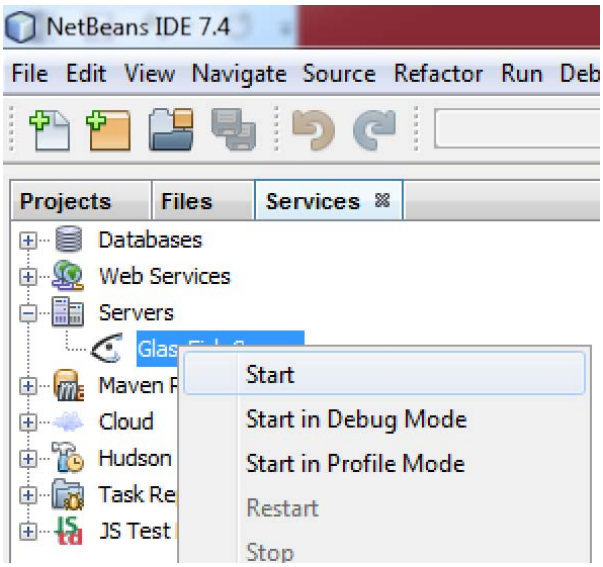


Fig.3.5.2 Iniciar Glassfish

Fuente: Los Autores

9. Verificar que el servidor *Glassfish* se inició correctamente y está listo para ser utilizado.

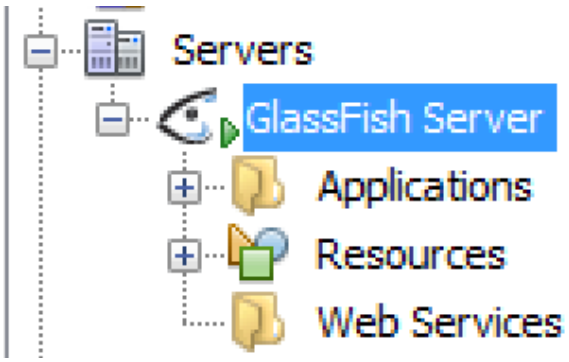


Fig.3.5.3 GlassFish Iniciado

Fuente: Los Autores

Implementación y programación del Web Service

Ahora se explicará la programación de los servicios web creados para generar la comunicación con los distintos clientes, ya sean dispositivos móviles inteligentes Android o IOs.

Para la comunicación con dispositivos Android se utilizará el protocolo SOAP (Simple Object Access Protocol) y para los dispositivos IOs, el protocolo REST (Representational State Transfer). A continuación se expondrán los conceptos y programación de estos servicios web junto con sus protocolos.

Creación del proyecto

Estos pasos iniciales se repetirán para los servicios web que hagan uso del protocolo SOAP o del protocolo REST.

1. Dentro de NetBeans, dar click en *File*, seguido de *New Project.*

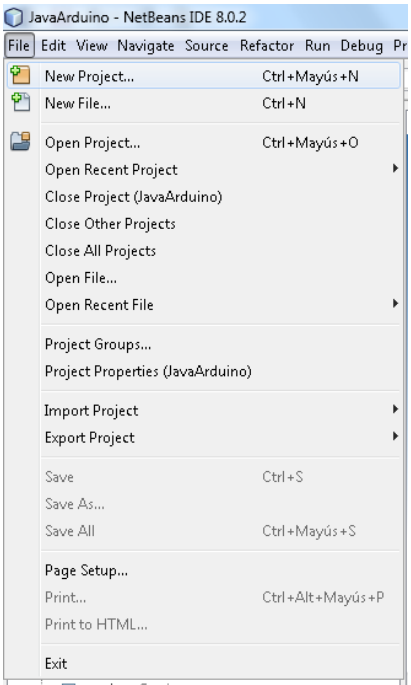


Fig.3.5.4 Nuevo Proyecto

Fuente: Los Autores

2. Dentro de la carpeta “Java Web”, escoger la opción de la derecha “Web Application” y seleccionar “Next”

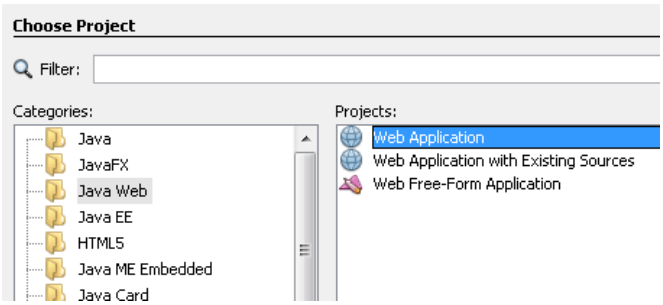


Fig.3.5.5 Aplicación Web

Fuente: Los Autores

3. Escoger un nombre para el proyecto y presionar *Next*

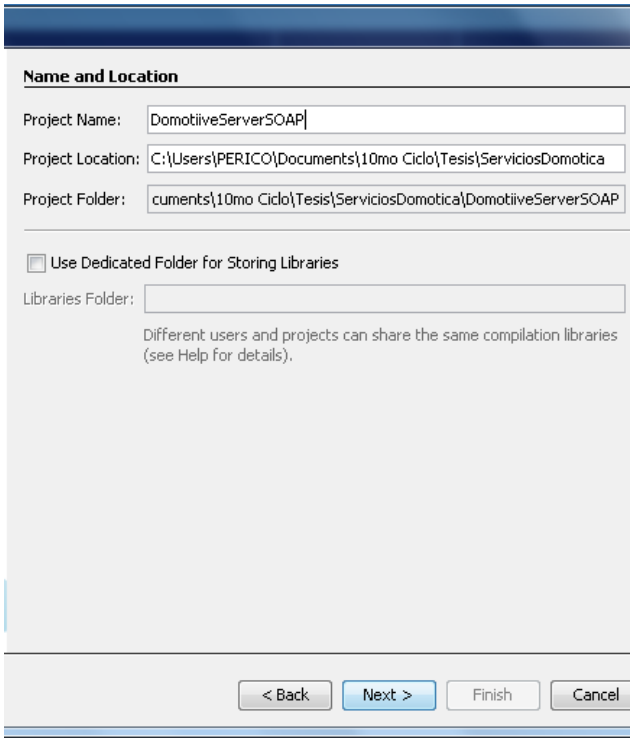


Fig.3.5.6 Nombre Proyecto

Fuente: Los Autores

4. Escoger el servidor y la versión de Java EE con la que se trabajará.

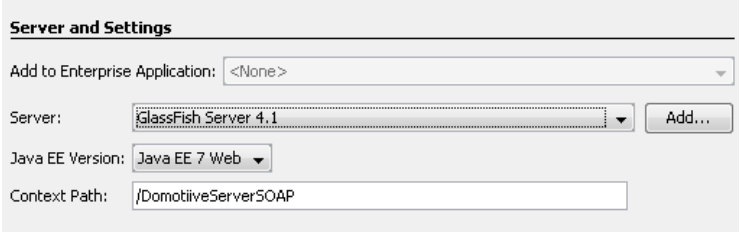


Fig.3.5.7 Servidor Java

Fuente: Los Autores

5. Dar clic en *finish* para obtener el proyecto creado.

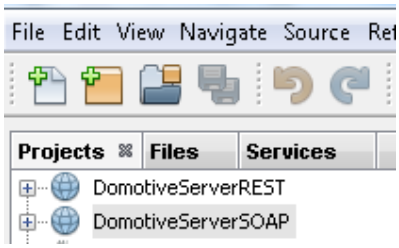


Fig.3.5.8 Proyecto Creado

Fuente: Los Autores

A partir de este punto se va a diferenciar entre la creación del web service que va a hacer uso del protocolo SOAP y el que va a utilizar el protocolo REST.

Web Service SOAP (Simple Object Access Protocol)

1. Dentro del proyecto creado con anterioridad, añadir una instancia de *web service* al proyecto; hacer clic derecho en el proyecto y seleccionar la opción *Web Service* dentro de la pestaña *New*.

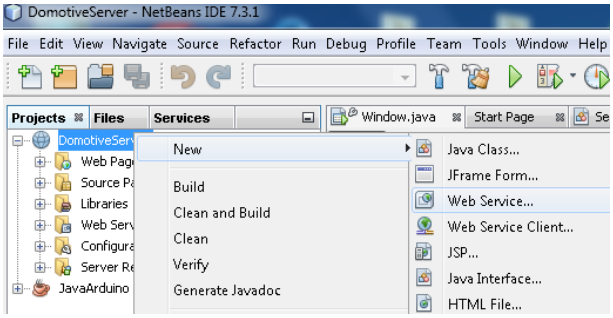


Fig.3.5.9 Agregar Web Service

Fuente: Los Autores

2. Asignar un nombre al *Web Service*, en este caso el nombre es *ServicioDomotica* y crear el paquete en el que se localizará el servicio, en este caso el paquete se llamará *servicios.*

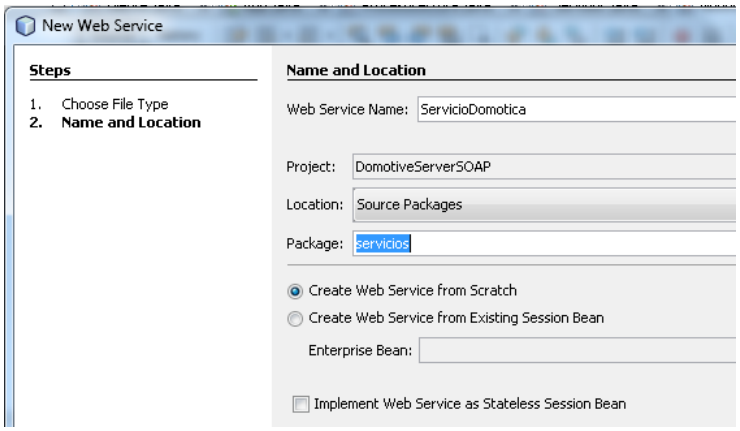


Fig.3.6.1 Nombre Web Service

Fuente: Los Autores

3. Verificar que el *Web Service ServicioDomotica* se muestra en el árbol de trabajo de nuestro proyecto.

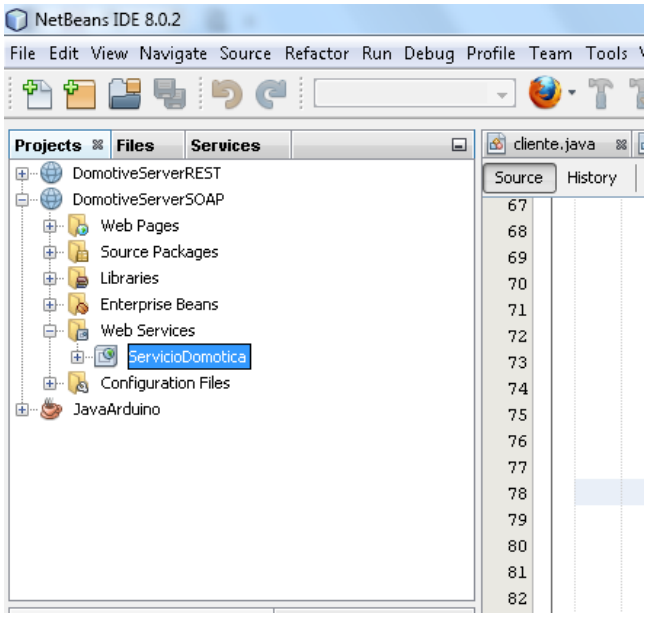


Fig.3.6.2 Servicio Domotica

Fuente: Los Autores

4. Realizar el despliegue de la aplicación dentro del servidor con un clic derecho sobre el proyecto y presionando *deploy.*

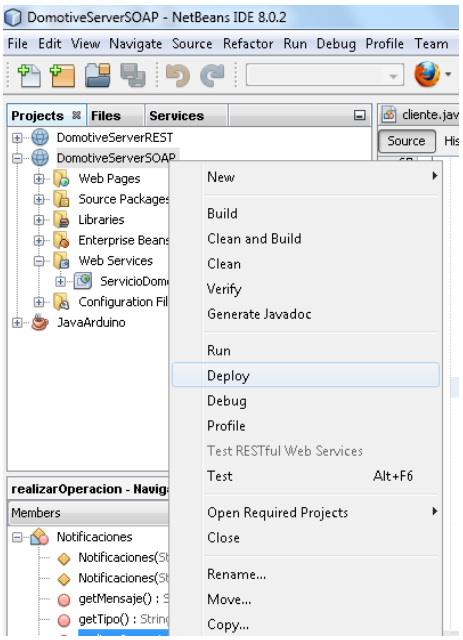


Fig.3.6.3 Despliegue Web Service

Fuente: Los Autores

5. Comprobar que el proyecto *DomtiveServerSOAP* y el *Web Service ServicioDomotica* se encuentran dentro de la instancia de Glassfish creada al inicio, seleccionando la pestaña *Services* y desplegando la carpeta *Applications* dentro de la instancia del servidor.

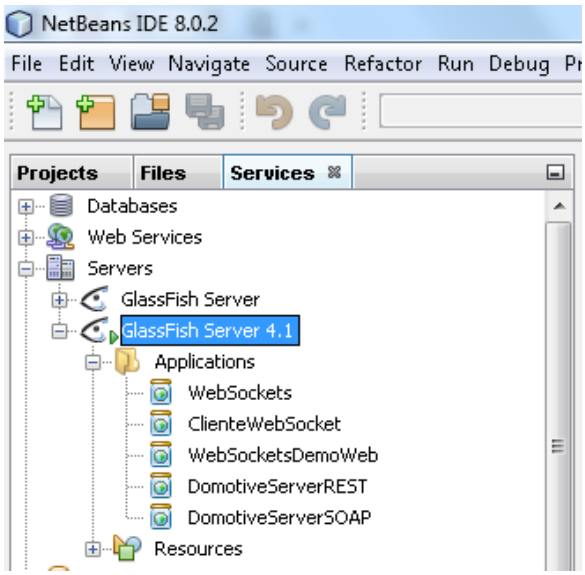


Fig.3.6.4 Instancia Creada

Fuente: Los Autores

6. Dentro de la pestaña *Projects*, desplegar el árbol del proyecto y a su vez la carpeta *Source Packages*, dentro de esta se encontrará el paquete *servicios* que creamos con anterioridad, el cual contiene la clase *ServicioDomotico.java*, en la cual se

programará el código principal del web service.

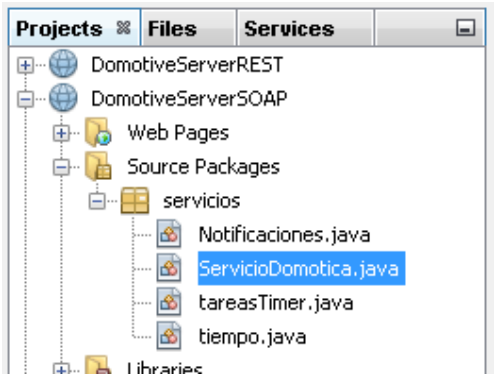


Fig.3.6.5 ServicioDomotica.java

Fuente: Los Autores

Código de programación del Web Service DomotiveServerSOAP:

Código fuente de la clase *ServicioDomótica.java*

1. Método *Conexion*, realiza la conexión con la base de datos.

```
@WebMethod(operationName = “Conexion”)
public String Conexion()
{
try
{
   InitialContext ctx = null;
   try
   {
         ctx = new InitialContext();
   }
   catch (NamingException ex)
   {
         Logger.getLogger(ServicioIluminacio
         n.class.getName()).log(Level.SEVERE
         , null, ex);
         return “err”;
   }

 //The JDBC Data source that we just created
 DataSource ds;
 ds = (DataSource)
 ctx.lookup(“jdbc/MySQLDataSource”);
 try
```

```
    {
      connection = ds.getConnection();
    }
      catch (SQLException ex)
    {
      Logger.getLogger(ServicioIluminacion.clas
      s.getName()).log(Level.SEVERE, null, ex);
      return “err”;
    }

      if (connection == null)
    {
      try
      {
            throw new SQLException(“Error
            establishing connection!”);
      }
      catch (SQLException ex)
      {

      Logger.getLogger(ServicioIluminacion.clas
      s.getName()).log(Level.SEVERE, null, ex);
      return “err”;
      }
    }
      else
    {
      return “si”;
    }
    }
catch (NamingException ex)
    {
      Logger.getLogger(ServicioIluminacion.clas
      s.getName()).log(Level.SEVERE, null, ex);
    }
      return “err”;
    }
```

2. Método *OnOff*, actualiza el estado de la lámpara en la base de datos cuando ésta es encendida o apagada.

```
@WebMethod(operationName = “OnOff”)
public String OnOff(@WebParam(name = “name”)
String est, @WebParam(name = “name2”) String
id)
{
try
  {
    PreparedStatement ps =
    connection.prepareStatement(
     “UPDATE domotivedatabase.lampara SET
    estado = ? WHERE id = ?”);
  {
    ps.setString(1,est);
    ps.setString(2,id);

    if (est.equalsIgnoreCase(“on”))
  {
```

```
        ps.executeUpdate();
        Notificaciones NOT = new
        Notificaciones();
        NOT.notificarLampara(“lamp”,est);
        return est;
    }
      else if(est.equalsIgnoreCase(“off”))
    {
      ps.executeUpdate();
      Notificaciones NOT = new
      Notificaciones();
      NOT.notificarLampara(“lamp”,est);
      return est;
    }
  }
}
catch(SQLException ex)
{
      return “err”;
}
      return “err”;
}
```

3. Método *borrarSimulación*, encargado de eliminar de la base de datos las simulaciones que ya han sido completadas.

```
@WebMethod(operationName =
“borrarSimulacion”)
public String
borrarSimulacion(@WebParam(name = “name”)
String id)
    {
try
        {
PreparedStatement ps =
connection.prepareStatement(“DELETE FROM
domotivedatabase.simulacion WHERE idLuz =
?”);
            {
ps.setString(1,id);
ps.executeUpdate();
return “si”;
            }
        }
catch(SQLException ex)
        {
return “err”;
        }
    }
```

4. Método *getEstado*, encargado de obtener el estado en el que se encuentra una lámpara según el id que se le envía como parámetro.

```
@WebMethod(operationName = “getEstado”)
```

```
public String getEstado(@WebParam(name =
“name”) String id)
 {
   try
   {
         PreparedStatement ps =
         connection.prepareStatement(“SELECT
         estado FROM lampara WHERE id = ?”);
   {
   ps.setString(1,id);
   ResultSet rs = ps.executeQuery();
   if (rs.next())
   {
         try
         {
               return(rs.getString(“estado”)
         );
         }
               catch (SQLException ex)
         {
               Logger.getLogger(ServicioIlum
               inacion.class.getName()).log(
               Level.SEVERE, null, ex);
         }
         }
   }
   }
   catch(SQLException ex)
   {
         return “err”;
  }
   return “err”;}
```

5. Método *insertarSimulacion*, es el encargado de insertar las simulaciones que el usuario envía desde su dispositivo cliente, dentro de la base de datos.

```
@WebMethod(operationName =
“insertarSimulacion”)
public String
insertarSimulacion(@WebParam(name = “id”)
String id, @WebParam(name = “fecha”) String
fecha,
@WebParam(name = “horaIn”) String horaIn,
@WebParam(name = “horaFin”) String horaFin)
```

```
{
try
        {
PreparedStatement ps =
connection.prepareStatement(
```

```
“INSERT INTO domotivedatabase.simulacion
(idLuz, horaInicio, horaFin, fecha)
VALUES(?,?,?,?)”);
            {
ps.setString(1,id);
ps.setString(2,horaIn);
ps.setString(3,horaFin);
ps.setString(4,fecha);
ps.executeUpdate();
tiempo t = new tiempo();
Date inicio = t.DeStringADate(fecha+”
“+horaIn+”:00”);
Date fin = t.DeStringADate(fecha+”
“+horaFin+”:00”);
//Crea un temporizador que se ejecuta en la
fecha y hora ingresada para encender la
//lámpara
TimerTask taskIn = new tareasTimer(“on”);
Timer TIMER = new Timer();
TIMER.schedule(taskIn, inicio);
//Crea un temporizador que se ejecuta en la
fecha y hora ingresada para apagar la
//lámpara
TimerTask taskFin = new tareasTimer(“off”);
Timer TIMER2 = new Timer();
TIMER2.schedule(taskFin, fin);
return “Inserción Exitosa”;
            }
         }
catch(SQLException ex)
         {
return “err”;
         }
    }
```

6. Una vez creados los métodos expuestos anteriormente se puede observar que aparecen disponibles para que puedan ser consumidos por los clientes a través del web service *ServicioDomotica*.

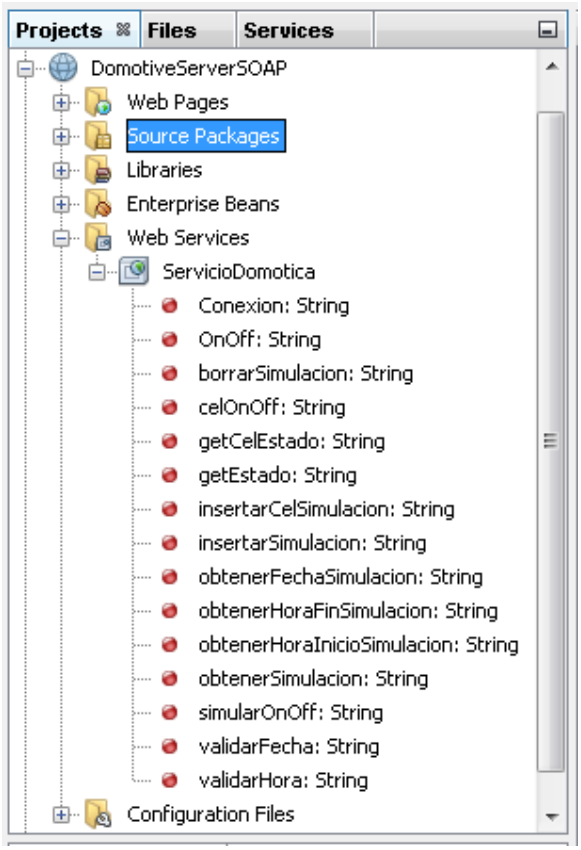


Fig.3.6.6 Métodos Disponibles

Fuente: Los Autores

Código fuente de la clase *Notificaciones.java*

Esta clase se encarga de crear las notificaciones que serán enviadas desde el servidor hacia los clientes conectados al mismo.

```
@ServerEndpoint(“/broadcast”)
@Singleton //es a la vez un EJB para
programar eventos
public class Notificaciones {
static final Logger LOGGER =
Logger.getLogger(Notificaciones.class.getNam
e());
//la lista de conexiones realizadas
static final List<Session> conexiones = new
ArrayList<>();
private static String sim;
```

1. Método *iniciaSesion*, encargado de agregar cada cliente a una lista, cuando se realiza una nueva conexión.

```
@OnOpen
public void iniciaSesion(Session session) {
LOGGER.log(Level.INFO, “Iniciando la
conexion de {0}”, session.getId());
conexiones.add(session); //Se agrega la
sesión a la lista
 }
```

2. Método *finConexion*, encargado de ejecutarse cuando se pierde la conexión con un cliente.

```
@OnClose
public void finConexion(Session session) {
LOGGER.info(“Terminando la conexion”);
if (conexiones.contains(session)) { // se
averigua si está en la colección
try {
   LOGGER.log(Level.INFO, “Terminando la
   conexion de {0}”, session.getId());
             session.close(); //se cierra la
   conexión
             conexiones.remove(session); //
   se retira de la lista
   } catch (IOException ex) {
   LOGGER.log(Level.SEVERE, null, ex);
         }
     }
   }


   public void setSimulacion(String sim)
   {
         this.sim = sim;
   }
```

3. Método *notificarLampara*, se encarga de notificar a los clientes conectados cuando una lámpara cambió de estado.

```
public void notificarLampara(String tipo,
String estado)
 {
String mensaje = tipo+”,”+estado; // el
mensaje a enviar
     for (Session sesion : conexiones)
//recorro toda la lista de conectados
     {
RemoteEndpoint.Basic remote =
sesion.getBasicRemote(); //tomo la conexion
remota con el cliente...
try
         {
remote.sendText(mensaje); //... y envío el
mensaje
         }
   catch (IOException ex)
         {
   LOGGER.log(Level.WARNING, null, ex);
         }
     }
   }
```

Código fuente de la clase *tiempo.java*

Clase encargada de realizar operaciones con horas y fechas, tales como, comparar horas y fechas, retornar hora y fecha actual.

Las horas y fechas deberán tener un formato de tipo de datos String, de la siguiente manera:

- Hora: HH:mm:ss
- Fecha: dd/mm/aa

```
package servicios;

import java.text.ParseException;
```

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public class tiempo {
    private String fechaInicio;
    private String fechaFin;
    private String horaInicio;
    private String horaFin;
```

1. Método *compararHoras*, se encarga de devolver un resultado entero; 1 cuando la hora inicial es mayor a la final, 2 cuando la hora inicial es menor a la final y 0 si son iguales.

```
    public int compararHoras()
    {
                int horaI =
         Integer.parseInt(obtenerHora(horaIn
         icio));
         int horaF =
         Integer.parseInt(obtenerHora(horaFi
         n));
         int minutoI =
         Integer.parseInt(obtenerMinutos(hor
         aInicio));
         int minutoF =
         Integer.parseInt(obtenerMinutos(hor
         aFin));

        if (horaI > horaF)
        {
            return 1;
        }
        else if(horaI < horaF)
        {
            return 2;
        }
        else
        {
            if(minutoI > minutoF)
            {
                return 1;
            }
            else if(minutoI < minutoF)
            {
                return 2;
            }
            else
            {
                return 0;
```

```
        }
      }
    }
```

2. Método *compararFechas*, se encarga de devolver un resultado entero; 1 cuando la fecha inicial es mayor a la final, devuelve 2 cuando la fecha inicial es menor a la final y 0 si son iguales.

```
public int compararFechas()
{
            int anioI =
 Integer.parseInt(obtenerAnio(fechaI
 nicio));
int mesI =
Integer.parseInt(obtenerMes(fechaIni
cio));
int diaI =
Integer.parseInt(obtenerDia(fechaIni
cio));
int anioF =
Integer.parseInt(obtenerAnio(fechaFi
n));
int mesF =
Integer.parseInt(obtenerMes(fechaFin
));
int diaF =
Integer.parseInt(obtenerDia(fechaFin
));

if(anioI > anioF)
{
    return 1;
}
else if(anioI < anioF)
{
    return 2;
}
else
{
    if (mesI > mesF)
    {
        return 1;
    }
    else if(mesI < mesF)
    {
        return 2;
    }
    else
    {
        if (diaI > diaF)
```

```
            {
                return 1;
            }
            else if(diaI < diaF)
            {
                return 2;
            }
            else
            {
                return 0;
            }
        }
    }
}
```

3. Métodos *set*, encargados de asignar valores entrantes a las variables privadas y globales de la clase.

```
public void setFechaInicio(String fecha)
{
    fechaInicio = fecha;
}

public void setFechaFin(String fecha)
{
    fechaFin = fecha;
}

public void setHoraInicio(String hora)
{
    horaInicio = hora;
}

public void setHoraFin(String hora)
{
    horaFin= hora;
}
```

4. Método *obtenerFechaActual*, encargado de devolver la fecha actual del sistema.

```
public String obtenerFechaActual()
{
  Calendar calendar =
  Calendar.getInstance();
    int dia =
  calendar.get(Calendar.DAY_OF_MONTH);
    int mes =
  calendar.get(Calendar.MONTH);
```

```
    int anio =
  calendar.get(Calendar.YEAR);
    mes ++;
    if (mes == 13)
    {
        mes = 1;
    }

     String fechaActual= dia + "/"+
     mes++ + "/" + anio;
    return fechaActual;
}
```

5. Método *obtenerHora*, se encarga de retornar los dígitos correspondientes a la hora de una cadena que contiene hora y minutos.

```
public String obtenerHora(String hora)
{
    int f = hora.indexOf(":");
    String h = hora.substring(0, f);
    return h;
}
```

6. Método *obtenerMinutos*, se encarga de retornar los dígitos correspondientes a los minutos de una cadena que contiene hora y minutos.

```
public String obtenerMinutos(String
hora)
{
    int i = hora.indexOf(":");
    i++;
    String m = hora.substring(i, i+2);
    return m;
}
```

7. Método *obtenerAnio*, encargado de retornar el año de una fecha específica.

```
public String obtenerAnio(String fecha)
{
    int i = fecha.indexOf("/");
    i++;
```

```
        fecha = fecha.substring(i,
        fecha.length());
       i = fecha.indexOf("/");
       i++;
        String a = fecha.substring(i,
        fecha.length());
       return a;
}
```

8. Método *obtenerMes*, encargado de retornar el mes de una fecha específica.

```
public String obtenerMes(String fecha)
{
    int i = fecha.indexOf("/");
    i++;
    String m = fecha.substring(i, i+2);
    return m;
}
```

9. Método *obtenerDia*, encargado de retornar el día de una fecha específica.

```
public String obtenerDia(String fecha)
{
    int f = fecha.indexOf("/");
    String d = fecha.substring(0, f);
    return d;
}
```

10. Método *DeStringADATE*, encargado de convertir una cadena o String tipo dd/MM/yyyy HH:mm:ss a un tipo de dato *Date*.

```
public Date DeStringADate(String fecha){
     SimpleDateFormat formato = new
     SimpleDateFormat("dd/MM/yyyy
     HH:mm:ss");
    String strFecha = fecha;
    Date fechaDate = null;
    try {
     fechaDate =
     formato.parse(strFecha);

     System.out.println(fechaDate.toStri
     ng());
        return fechaDate;
    } catch (ParseException ex) {
        ex.printStackTrace();
```

```
            return fechaDate;
          }
        }
      }
```

Código fuente de la clase *tareasTimer.java*

Clase abstracta llamada por un Timer o Temporizador, que se ejecuta en una fecha y hora determinada. El objetivo de esta clase es encender o apagar la lámpara según lo programado por el Timer que la llama. Esta clase es ejecutada cada vez que se realiza una simulación de presencia.

```
import java.util.Calendar;
import java.util.TimerTask;

public class tareasTimer extends TimerTask
{
    ServicioDomotica SERV = new
ServicioDomotica();
    private String estado;

    public tareasTimer(String estado)
    {
        this.estado = estado;
    }

    @Override
      public void run()
        {
            if(estado.equalsIgnoreCase("on"))
            {

                  SERV.simularOnOff("1", "on");
                  System.out.println("Encendio
                  la simulacion");
            }
            else
            {
                SERV.simularOnOff("1", "off");

                  System.out.println("Apago
                  Simulacion");
            }
        }
}
```

Una vez finalizada la codificación del Web Service SOAP, se realizará la implementación del Web Service REST como se describe a continuación:

Web Service REST (Representational State Transfer)

1. Dentro del proyecto creado con anterioridad, añadir una instancia de RESTful *web service* al proyecto; Hacer clic derecho en el proyecto y seleccionar la opción *RESTful Web Services…* dentro de la pestaña *New*.

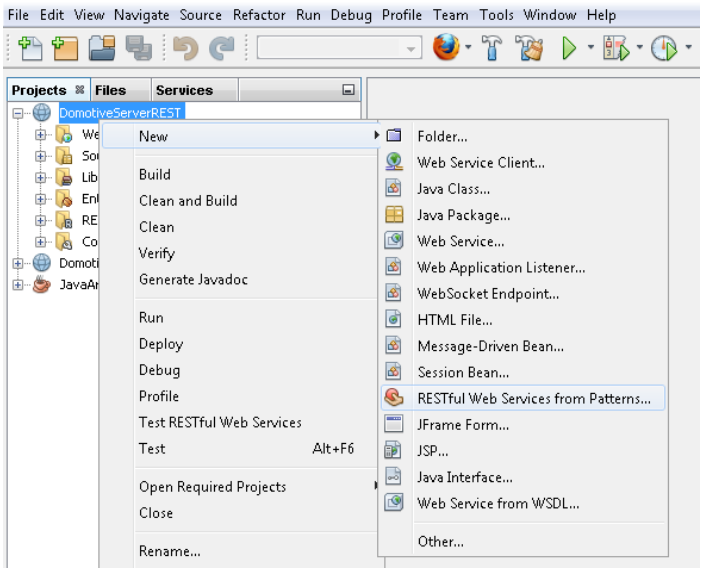


Fig.3.6.7 Restful Web Service

Fuente: Los Autores

2. Seleccionar *Simple Root Resource* y presionar *Next*

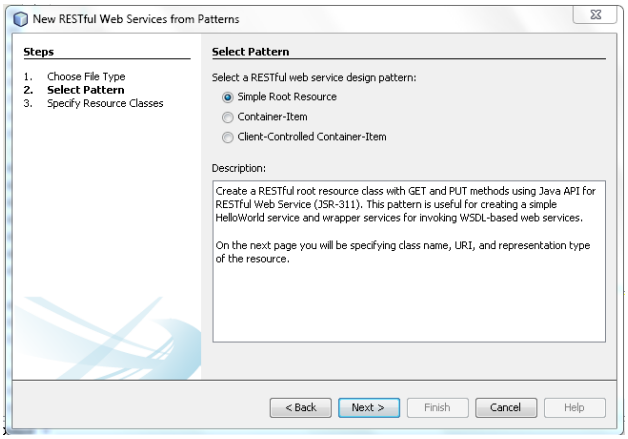


Fig.3.6.8 Seleccionar Recurso

Fuente: Los Autores

3. Asignar un nombre al paquete que contendrá la clase, en este caso será *servicios*, seguido de esto, en la opción *Path* escribir el nombre de la clase, en la opción *Class Name* escribir el mismo nombre, luego en la opción *MIME Type*, elegir *application/json* y finalmente presionar *Finish*..

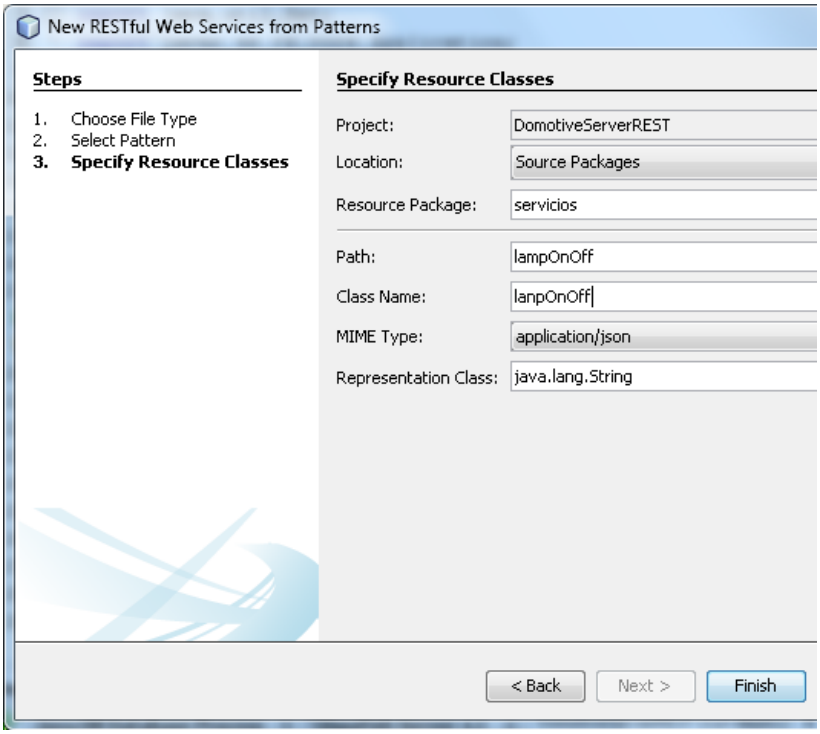


Fig.3.6.9 Clases del Recurso

Fuente: Los Autores

4. El paso 4 se debe repetir para cada clase del web service que se desee crear.

5. Finalmente se podrá observar dentro del árbol del proyecto todas las clases creadas, junto con una clase llamada *ApplicationConfig.java*, la cual se generó automáticamente. Dentro de la capeta *RESTful Web Services*, se podrá ver cada una de las clases Restful que han sido creadas.

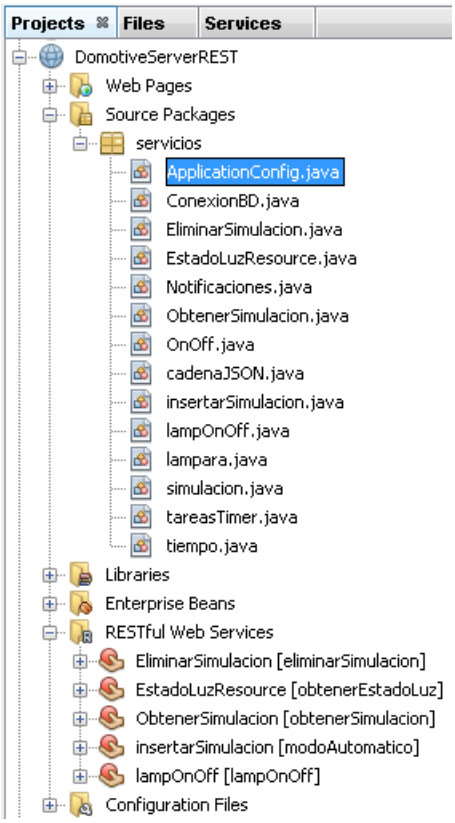


Fig.3.7.1 RESTful Web Services

Fuente: Los Autores

6. Realizar el despliegue de la aplicación dentro del servidor con un clic derecho sobre el proyecto y presionando *deploy*.

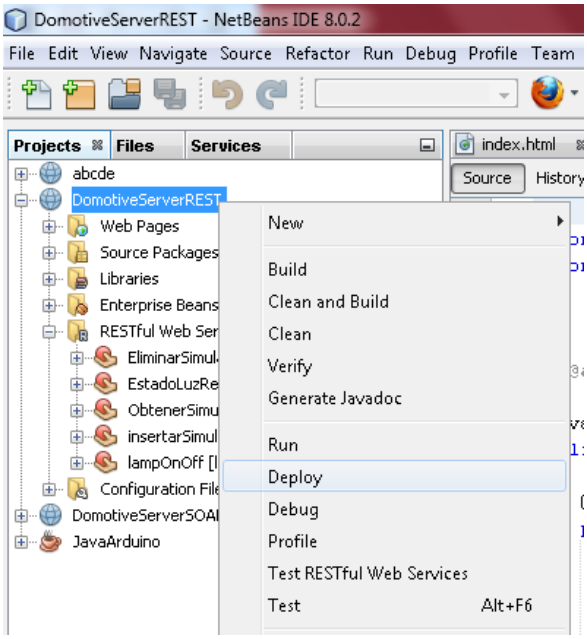


Fig.3.7.2 Despliegue REST

Fuente: Los Autores

7. Comprobar que el proyecto *DomtiveServerREST*, se encuentran dentro de la instancia de Glassfish creada al inicio, seleccionando la pestaña *Services* y desplegando la carpeta *Applications* dentro de la instancia de dicho servidor.

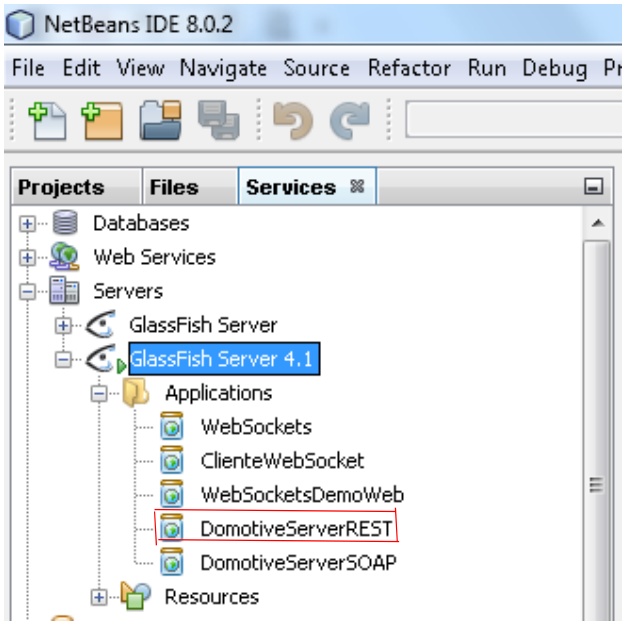


Fig.3.7.3 Instancia REST

Fuente: Los Autores

Código de programación del Web Service DomotiveServerREST:

Código fuente de la clase *lampOnOff.java*

Clase que se encarga de recibir el estado de la lámpara que es enviado desde los distintos clientes, para luego actualizar el estado de la misma en el servidor y posteriormente en la base de datos.

```
@Path("lampOnOff")
public class lampOnOff {

    @Context
    private UriInfo context;


    public lampOnOff() {
    }


    @GET
    @Produces("application/json")
    public String
getXml(@QueryParam("estado")String
estado) {
```

```
        //TODO return proper representation
    object
        ConexionBD con = new ConexionBD();
        Connection c = con.Conexion();
        if(c != null)
        {
               OnOff cambiarEstado = new
               OnOff();
            String resEstado =
            cambiarEstado.OnOff("1", estado,
            c);

         if(resEstado.equalsIgnoreCase("err"
         ) == false)
            {

               ArrayList<lampara> lamp = new
               ArrayList();
                 lampara lampOn=new
                 lampara("1",estado);

                 //agregamos a una lista
                 lamp.add(lampOn);

                 //convertimos la lista a
                 formato JSON
                 Gson JSO = new Gson();

                 String formatoJSON =
                 JSO.toJson(lamp).substring(1
                 ,JSO.toJson(lamp).length()-
                 1);

                 return formatoJSON;
            }
            else
            {
               return "Problema al encender
               la luz";
            }
        }
        else
        {
               return "Falló Conexión con
               la Base de Datos!";
        }
     }

     @PUT
     @Consumes("application/xml")
     public void putXml(String content) {
     }
    }
```

Código fuente *insertarSimulacion.java*

Clase que se encargará de recibir los datos de las simulaciones que son enviados desde los clientes para luego, siendo

validados y sabiendo que son correctos, ingresar los mismos en la base de datos y notificar que han sido ingresado con éxito.

```
@Path("modoAutomatico")
public class insertarSimulacion {

    @Context
    private UriInfo context;

    /**
     * Creates a new instance of modoAutomatico
     */
    public insertarSimulacion() {
    }

    /**
     * Retrieves representation of an instance
of holaMundo.modoAutomatico
     * @return an instance of java.lang.String
     */

    @GET
    @Produces("application/json")
    public String getJson(@QueryParam("id")
String id, @QueryParam("fecha") String fecha,
@QueryParam("horaIn") String horaIn,
@QueryParam("horaFin") String horaFin)
    {
        //TODO return proper representation
object
        simulacion SIMULACION = new
simulacion();
        cadenaJSON SIM;
        String formatoJSON;
        Gson JSO = new Gson();

        ConexionBD con = new ConexionBD();
        Connection CONEXION = con.Conexion();

        tiempo t = new tiempo();
        t.setHoraInicio(horaIn);
        t.setHoraFin(horaFin);

t.setFechaInicio(t.obtenerFechaActual());
        t.setFechaFin(fecha);
        int resFecha = t.compararFechas();

        switch(resFecha)
        {
            case -1:
```

```
                SIM = new cadenaJSON(id, "Los
datos ingresados son incorrectos!");
                formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
                return formatoJSON;

            case 2:
                if
(SIMULACION.insertarSimulacion(id, fecha,
horaIn, horaFin).equalsIgnoreCase("Inserción
Exitosa"))
                {
                    Date inicio =
t.DeStringADate(fecha+" "+horaIn+":00");
                    Date fin =
t.DeStringADate(fecha+" "+horaFin+":00");

                    TimerTask taskIn = new
tareasTimer("on", CONEXION);
                    Timer TIMER = new Timer();
                    TIMER.schedule(taskIn,
inicio);

                    TimerTask taskFin = new
tareasTimer("off", CONEXION);
                    Timer TIMER2 = new Timer();
                    TIMER2.schedule(taskFin,
fin);
                    SIM = new cadenaJSON(id,
"Simulacion Insertada!");
                    formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
                    return formatoJSON;
                }
                else
                {
                    SIM = new cadenaJSON(id,
"Error al ingresar la simulacion!");
                    formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
                    return formatoJSON;
                }

            case 1:
                SIM = new cadenaJSON(id, "Las
fechas ingresadas son incorrectas!");
                formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
                return formatoJSON;
```

```
            case 0:

t.setHoraFin(t.obtenerHoraActual());
                //return t.getHoraFin();
                if(t.compararHoras() == 1)
                {
                    t.setHoraFin(horaFin);
                    if(t.compararHoras() == 2)
                    {
                        if
(SIMULACION.insertarSimulacion(id, fecha,
horaIn, horaFin).equalsIgnoreCase("Inserción
Exitosa"))
                        {
                            Date inicio =
t.DeStringADate(fecha+" "+horaIn+":00");
                            Date fin =
t.DeStringADate(fecha+" "+horaFin+":00");

                            TimerTask taskIn =
new tareasTimer("on", CONEXION);
                            Timer TIMER = new
Timer();

TIMER.schedule(taskIn, inicio);

                            TimerTask taskFin =
new tareasTimer("off", CONEXION);
                            Timer TIMER2 = new
Timer();

TIMER2.schedule(taskFin, fin);
                            SIM = new
cadenaJSON(id, "Simulacion Insertada!");
                            formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
                            return formatoJSON;
                        }
                        else
                        {
                            SIM = new
cadenaJSON(id, "Error al ingresar la
simulacion!");
                            formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
                            return formatoJSON;
                        }
                    }
                    else
                    {
```

```
                        SIM = new
cadenaJSON(id, "Los horas ingresadas son
incorrectas");
                        formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
                        return formatoJSON;
                    }
                }
                else
                {
                    SIM = new cadenaJSON(id,
"Los horas ingresadas son incorrectas");
                    formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
                    return formatoJSON;
                }
        }
        SIM = new cadenaJSON(id, "Los horas
ingresadas son incorrectas");
        formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
        return formatoJSON;
    }

    /**
     * PUT method for updating or creating an
instance of modoAutomatico
     * @param content representation for the
resource
     * @return an HTTP response with content of
the updated or created resource.
     */
    @PUT
    @Consumes("application/json")
    public void putJson(String content) {
    }
}
```

Código fuente *EliminarSimulacion.java*

Clase encargada de eliminar las simulaciones que ya han sido completadas, borrándolas de la base de datos.

```
@Path("eliminarSimulacion")
public class EliminarSimulacion {

    @Context
    private UriInfo context;

    /**
     * Creates a new instance of
EliminarSimulacion
     */
    public EliminarSimulacion() {
    }

    /**
     * Retrieves representation of an instance
of holaMundo.EliminarSimulacion
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("application/json")
    public String getJson(@QueryParam("id")
String id, @QueryParam("fecha") String fecha,
@QueryParam("horaIn") String horaIn,
@QueryParam("horaFin") String horaFin) {
        //TODO return proper representation
object
        cadenaJSON SIM;
        String formatoJSON;
        Gson JSO = new Gson();

        simulacion SIMULACION = new
simulacion();
        if(SIMULACION.borrarSimulacion(id,
horaIn, horaFin, fecha).equalsIgnoreCase("si"))
        {
            SIM = new cadenaJSON(id,
"Simulacion Eliminada!");
            formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
            return formatoJSON;
        }
        else
        {
            SIM = new cadenaJSON(id, "Error al
eliminar la simulacion!");
            formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
            return formatoJSON;
        }
    }
```

```
    @PUT
    @Consumes("application/json")
    public void putJson(String content) {
    }
}
```

Código fuente *EstadoLuzResource.java*

Clase encargada de devolver al cliente que hace la petición el estado en el que se encuentra la lámpara (encendida o apagada).

```
@Path("obtenerEstadoLuz")
public class EstadoLuzResource {

    @Context
    private UriInfo context;

    /**
     * Crea una nueva instancia de
EstadoLuzResource
     */
    public EstadoLuzResource() {
    }

    /**
     * Retrieves representation of an instance
of holaMundo.EstadoLuzResource
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("application/json")
    public String getJson(@QueryParam("id")
String id)
    {
        //TODO return proper representation
object
        String idLuz;
        String estado;
        cadenaJSON SIM;
        String formatoJSON;
        Gson JSO = new Gson();
        lampara LAMP = new lampara();
        LAMP.setID(id);
        ArrayList simu = new ArrayList();

        estado = LAMP.getEstado();
        if
(estado.equalsIgnoreCase("err")==false)
        {
```

```
            LAMP.setEstado(estado);
            simu.add(LAMP);
            formatoJSON =
JSO.toJson(simu).substring(1,JSO.toJson(simu).l
ength()-1);
            return formatoJSON;
        }
        else
        {
            SIM = new cadenaJSON(id, "No se ha
podido recuperar el estado!");
            formatoJSON =
JSO.toJson(SIM).substring(1,JSO.toJson(SIM).len
gth()-1);
            return formatoJSON;
        }
    }

    /**
     * PUT method for updating or creating an
instance of EstadoLuzResource
     * @param content representation for the
resource
     * @return an HTTP response with content of
the updated or created resource.
     */
    @PUT
    @Consumes("application/json")
    public void putJson(String content) {
    }
}
```

Código fuente *ObtenerSimulacion.java*

Clase encargarda de retornar la fecha y horas de las simulaciones previamente programadas a los clientes que la soliciten.

```
@Path("obtenerSimulacion")
public class ObtenerSimulacion {

    @Context
    private UriInfo context;

    /**
      * Creates a new instance of
      ObtenerSimulacion
     */
    public ObtenerSimulacion() {
```

```
}

/**
  * Retrieves representation of an instance
  of holaMundo.ObtenerSimulacion
 * @return an instance of java.lang.String
 */
@GET
@Produces("application/json")
  public String getJson(@QueryParam("id")
  String id) {
    //TODO return proper representation
    object
    String idLuz;
    String horaIn;
    String horaFin;
    String fecha;
    cadenaJSON SIM;
    String formatoJSON;
    Gson JSO = new Gson();
        simulacion SIMULACION = new
        simulacion();
    ArrayList simu = new ArrayList();

        ResultSet rs =
        SIMULACION.obtenerSimulacion(id);
    try
    {

        while (rs.next())
        {
            idLuz = rs.getString("idLuz");
                        horaIn =
        rs.getString("horaInicio");
            horaFin =
        rs.getString("horaFin");
            fecha = rs.getString("fecha");
                        simulacion sm = new
        simulacion(id, horaIn, horaFin,
        fecha);
            simu.add(sm);
        }
                    formatoJSON =
        JSO.toJson(simu).substring(1,JSO.to
        Json(simu).length()-1);
        return formatoJSON;
    }
    catch(SQLException ex)
    {
                    SIM = new
        cadenaJSON(id, "No hay simulaciones
        disponibles!");
```

```
            formatoJSON =
            JSO.toJson(SIM).substring(1,JSO.toJ
            son(SIM).length()-1);
            return formatoJSON;
        }
    }
    @PUT
    @Consumes("application/json")
    public void putJson(String content) {
    }
}
```

Una vez finalizada la codificación de todos los servicios web que ofrecerá el servidor a sus clientes, se expondrá a continuación los detalles de configuración que serán necesarios más adelante para que las aplicaciones móviles puedan consumir dichos servicios:

Consola de administración de Glassfish

Para ingresar a la consola de administración del servidor de las aplicaciones creadas anteriormente: *DomotiveServerSOAP y DomotiveServerREST*, hacer clic derecho en la instancia de Glassfish y seleccionar la opción *View Domain Admin Console*.

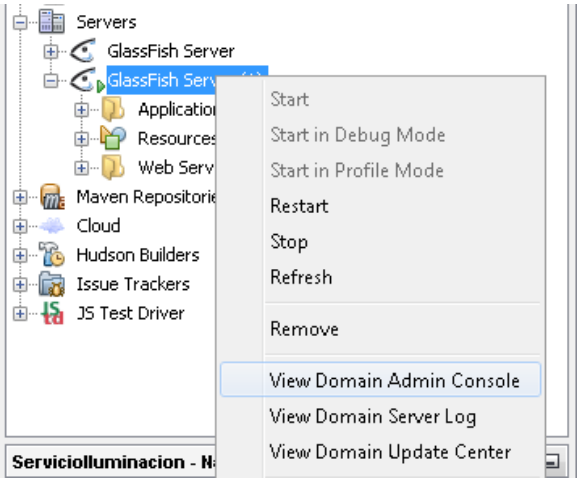


Fig.3.7.4 View Admin Console

Fuente: Los Autores

La consola se abrirá en un navegador web y se observará una pantalla similar a la que se muestra a continuación:

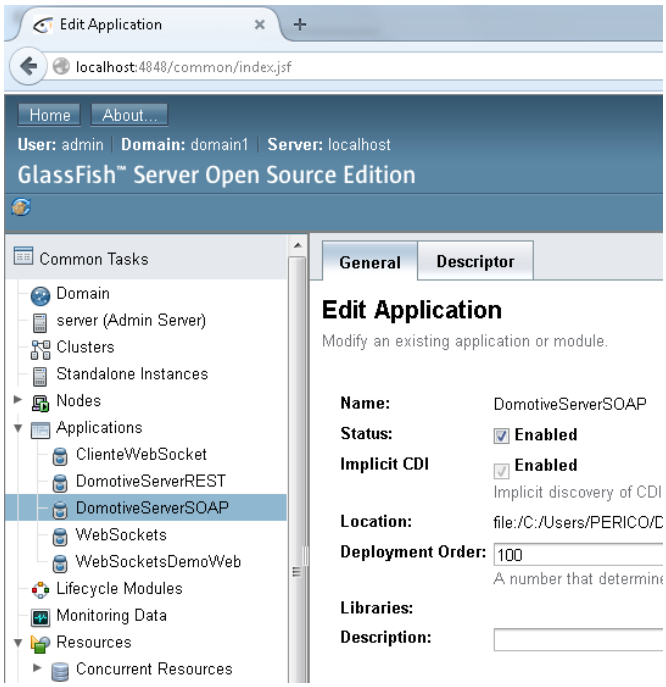


Fig.3.7.5 Consola de Administración

Fuente: Los Autores

Esta consola permite modificar las aplicaciones, los módulos y métodos de los distintos servicios que forman parte del servidor web, además de la posibilidad de monitorear el pool de conexiones, los conectores con las distintas bases de datos y las instancias creadas de cada servicio.

WSDL (Web Service Description Language)

Es un archivo basado en XML que describe los servicios web que ofrece el servidor a sus clientes.
(Christensen, 2001)

Para encontrar este archivo se debe ingresar a la consola de administración mencionada anteriormente, ir al menú *Aplicaciones* y seleccionar el servidor de aplicaciones *DomotiveServerSOAP*.

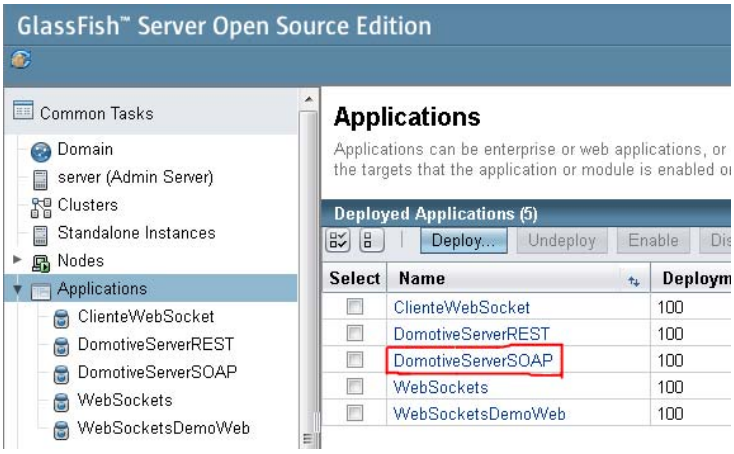


Fig.3.7.6 Aplicación SOAP

Fuente: Los Autores

En la siguiente página, en la parte inferior derecha hacer clic en la opción *View Endpoint*.

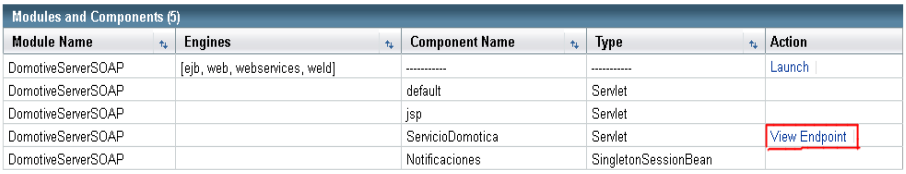


Fig.3.7.7 View Endpoint

Fuente: Los Autores

Dentro de la información de punto final del servicio web se encuentra el enlace que permite visualizar el archivo WSDL del servidor.

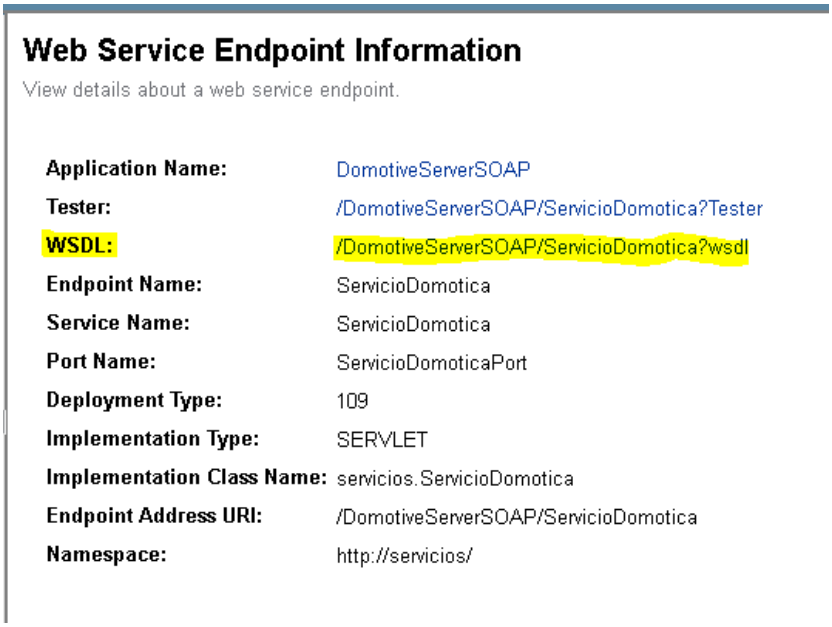


Fig.3.7.8 WSDL

Fuente: Los Autores

El archivo WSDL se abrirá en un navegador web, el cual contiene información necesaria para que los clientes (aplicaciones móviles) puedan comunicarse con el servidor *DomotiveServerSOAP* para el control del sistema de iluminación. La información que se muestra a continuación corresponde a los parámetros que serán utilizados en el capítulo 5 dentro de la codificación de las aplicaciones móviles:

URL:

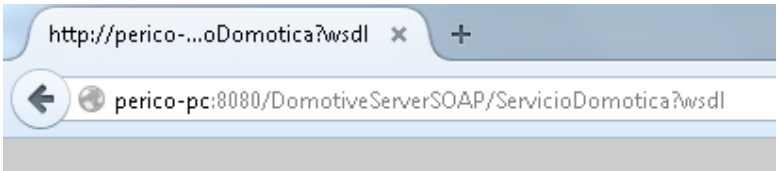


Fig.3.7.9 URL wsdl

Fuente: Los Autores

Target Namespace:

Fig.3.8.1 Target Namespace

Fuente: Los Autores

Name:

Nombre de los métodos que ofrece el servidor

```
  </types>
- <message name="obtenerSimulacion">
    <part name="parameters" element="tns:obtenerSimulacion"/>
  </message>
- <message name="obtenerSimulacionResponse">
    <part name="parameters" element="tns:obtenerSimulacionResponse"/>
  </message>
- <message name="insertarSimulacion">
    <part name="parameters" element="tns:insertarSimulacion"/>
  </message>
- <message name="insertarSimulacionResponse">
    <part name="parameters" element="tns:insertarSimulacionResponse"/>
  </message>
- <message name="borrarSimulacion">
    <part name="parameters" element="tns:borrarSimulacion"/>
  </message>
- <message name="borrarSimulacionResponse">
    <part name="parameters" element="tns:borrarSimulacionResponse"/>
  </message>
- <message name="getCelEstado">
    <part name="parameters" element="tns:getCelEstado"/>
  </message>
- <message name="getCelEstadoResponse">
    <part name="parameters" element="tns:getCelEstadoResponse"/>
  </message>
- <message name="validarFecha">
    <part name="parameters" element="tns:validarFecha"/>
  </message>
- <message name="validarFechaResponse">
    <part name="parameters" element="tns:validarFechaResponse"/>
  </message>
```

Fig.3.8.2 Métodos WSDL

Fuente: Los Autores

```
- <message name="simularOnOffResponse">
    <part name="parameters" element="tns:simularOnOffResponse"/>
  </message>
- <message name="celOnOff">
    <part name="parameters" element="tns:celOnOff"/>
  </message>
- <message name="celOnOffResponse">
    <part name="parameters" element="tns:celOnOffResponse"/>
  </message>
- <message name="getEstado">
    <part name="parameters" element="tns:getEstado"/>
  </message>
- <message name="getEstadoResponse">
    <part name="parameters" element="tns:getEstadoResponse"/>
  </message>
- <message name="Conexion">
    <part name="parameters" element="tns:Conexion"/>
  </message>
- <message name="ConexionResponse">
    <part name="parameters" element="tns:ConexionResponse"/>
  </message>
- <message name="OnOff">
    <part name="parameters" element="tns:OnOff"/>
  </message>
- <message name="OnOffResponse">
    <part name="parameters" element="tns:OnOffResponse"/>
  </message>
- <message name="insertarCelSimulacion">
    <part name="parameters" element="tns:insertarCelSimulacion"/>
  </message>
- <message name="insertarCelSimulacionResponse">
    <part name="parameters" element="tns:insertarCelSimulacionResponse"/>
  </message>
- <message name="obtenerFechaSimulacion">
    <part name="parameters" element="tns:obtenerFechaSimulacion"/>
  </message>
- <message name="obtenerFechaSimulacionResponse">
    <part name="parameters" element="tns:obtenerFechaSimulacionResponse"/>
  </message>
- <message name="obtenerHoraInicioSimulacion">
    <part name="parameters" element="tns:obtenerHoraInicioSimulacion"/>
  </message>
- <message name="obtenerHoraInicioSimulacionResponse">
    <part name="parameters" element="tns:obtenerHoraInicioSimulacionResponse"/>
  </message>
- <message name="obtenerHoraFinSimulacion">
    <part name="parameters" element="tns:obtenerHoraFinSimulacion"/>
  </message>
- <message name="obtenerHoraFinSimulacionResponse">
    <part name="parameters" element="tns:obtenerHoraFinSimulacionResponse"/>
  </message>
```

Fig.3.8.3 Métodos WSDL

Fuente: Los Autores

Despliegue del Servidor Web

Una vez que todo está instalado, configurado, codificado y listo para que los servicios sean consumidos se debe realizar el despliegue o en inglés *deploy* del servidor *DomotiveServerSOAP y DomotiveServerREST*, esto permitirá que los componentes de la aplicación estén disponibles para ser accedidos, es decir, que cada módulo esté descrito y que los requisitos específicos de configuración que necesita para ser llamado por un cliente estén a su disposición.

Para realizar el despliegue del servidor *DomotiveServerSOAP o DomotiveServerREST*, hacer clic derecho en el proyecto y seleccionar la opción *Deploy*.

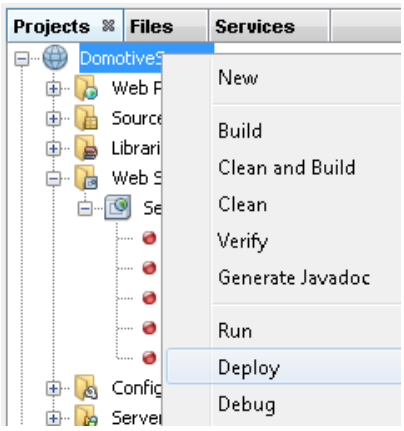


Fig.3.8.4 Despliegue Web Service

Fuente: Los Autores

Esta acción creará un nuevo directorio con el nombre *dist* dentro del espacio de trabajo del proyecto, en donde se

encontrará el archivo descriptor del despliegue con el mismo nombre del proyecto y extensión .war, cada vez que se realicen cambios en el proyecto se debe realizar nuevamente el despliegue del proyecto para que los cambios efectuados se actualicen en el archivo .war.

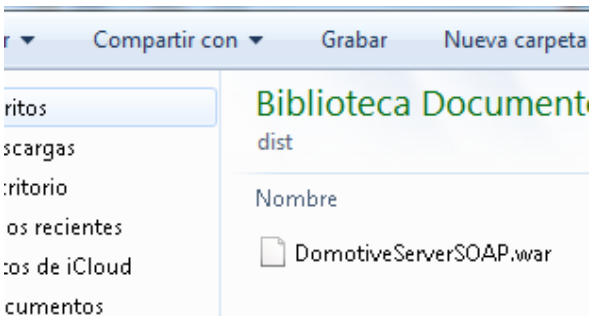


Fig.3.8.5 Archivo .WAR

Fuente: Los Autores

Prueba de los servicios (métodos) que ofrece el Servidor Web *DomotiveServerSOAP*

1. Hacer clic derecho en el proyecto web service *servicioDomotica* que se encuentra dentro de la carpeta *Web Services,* dentro del proyecto web *DomotiveServerSOAP*

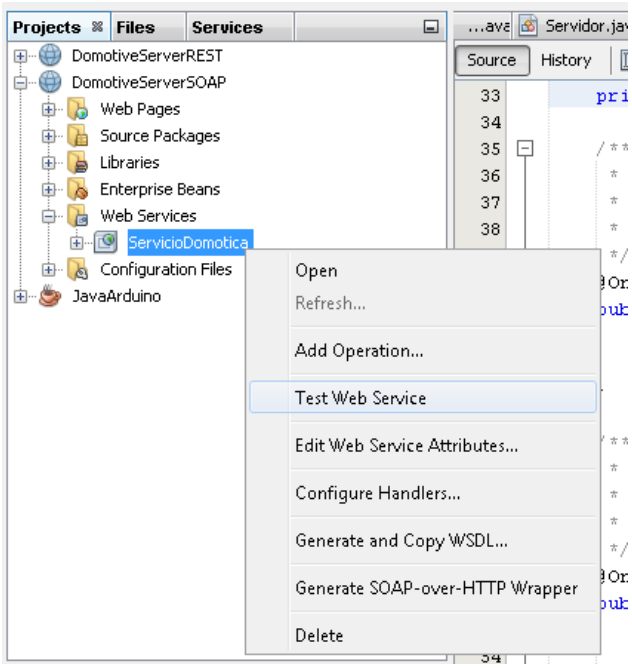


Fig.3.8.6 Test Web Sevice

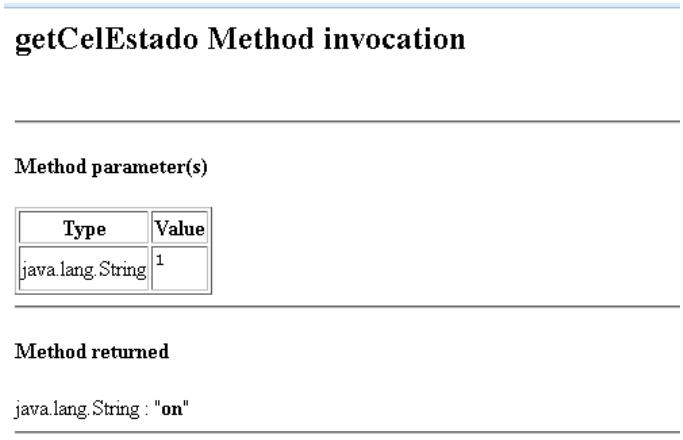
Fuente: Los Autores

2. En el navegador se despliega la siguiente ventana en la que se muestra los métodos implementados en el servidor con sus respectivos parámetros. Como ejemplo se realiza la prueba del método *getCelEstado*, al que se le envía como parámetro el número 1 que corresponde al "id" de la lámpara.

Fig.3.8.7 Web Service Tester

Fuente: Los Autores

3. Y se observa que la llamada al método *getCelEstado* se realiza exitosamente ya que retorna el estado en que se encuentra la lámpara, en este caso debido a que la lámpara se encuentra encendida, retorna la cadena "on".



8 Resultados Prueba

Fuente: Los Autores

Prueba de los servicios (métodos) que ofrece el Servidor Web *DomotiveServerREST*

1. Realizar un clic derecho sobre el proyecto y presionar la opción *Test RESful Web Services*.

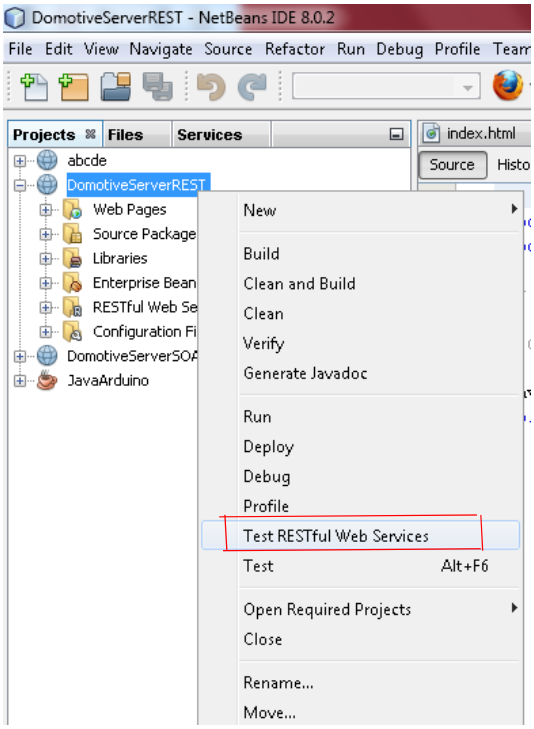


Fig.3.8.9 Test RESTful

Fuente: Los Autores

2. Seleccionar la opción *Web Test Client in Project* y presionar *Ok.*

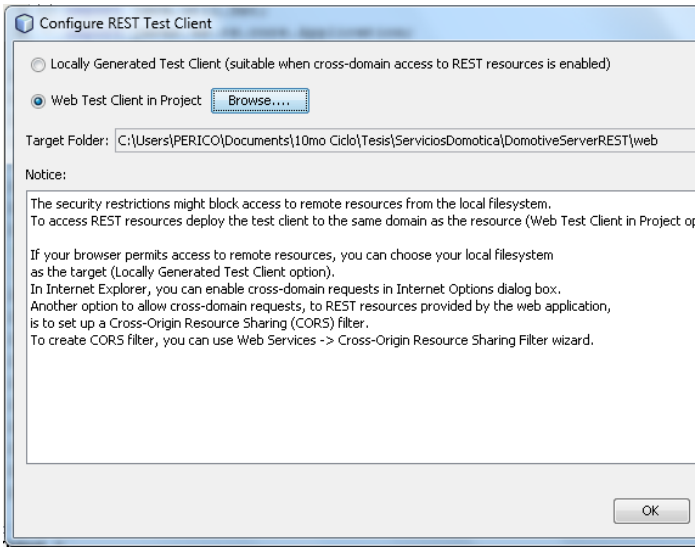


Fig.3.9.1 Cliente Web

Fuente: Los Autores

3. Una ventana del navegador se abrirá automáticamente, en la cual se podrá observar una etiqueta con el nombre *WADL*, esta etiqueta contiene la dirección a la que se debe acceder para poder hacer uso de los servicios. Debajo de esto se encuentra una carpeta desplegable con los distintos servicios disponibles creados anteriormente, se debe dar click en cualquiera de ellos para probarlos, como ejemplo, se realizará la prueba del servicio *lampOnOff*.

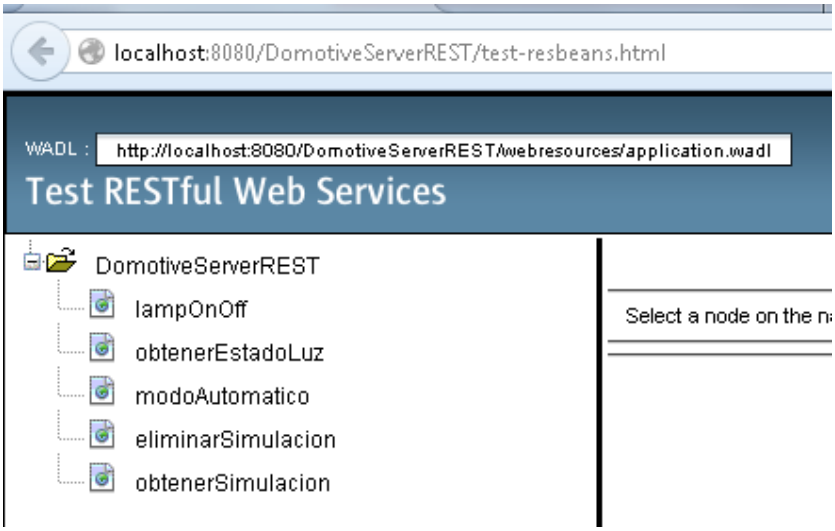


Fig.3.9.2 WADL

Fuente: Los Autores

4. En la parte derecha de la página aparecerá la información necesaria para realizar la llamada a dicho servicio, en la cual se muestra primero el nombre del recurso, seguido de la dirección en la cual se encuentra disponible, también se muestran los campos en los cuales se debe enviar los parámetros, en este campo se debe enviar el estado de la luz (*"on" / "off"*), si se quiere encender o apagar respectivamente y presionar *Test.*

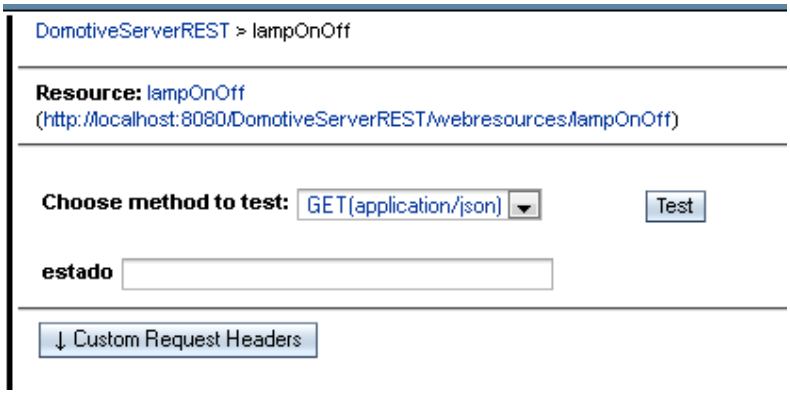


Fig.3.9.3 Test Method

Fuente: Los Autores

5. Vamos a obtener un resultado en la parte inferior derecha de la página, en el cual se expone la cadena que el servidor envía a los clientes que acceden a sus servicios por medio de este método.

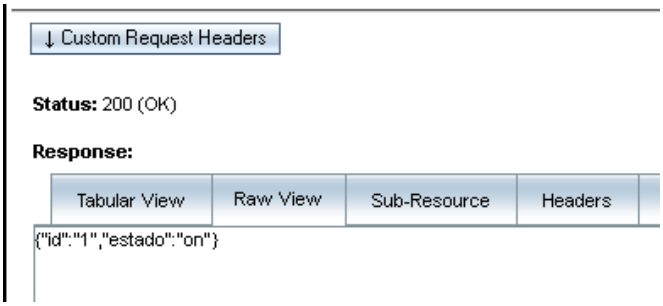


Fig.3.9.4 Resultado Prueba

Fuente: Los Autores

6. El formato con el que trabaja este servidor para envío y recepción de datos es JSON.

Finalmente, al concluir la programación de los servicios web, se explicará la codificación del cliente Java, encargado de comunicarse directamente con el circuito impreso y la placa Arduino para realizar las operaciones sobre la lámpara.

Este cliente, está en constante comunicación con los Servicios Web a espera de recibir notificaciones que le indiquen que operación debe realizar. A continuación se dará a conocer la configuración y la codificación del mismo.

1. Crear un nuevo proyecto en *NetBeans*, tipo *Java Application* y presionar *Next.*

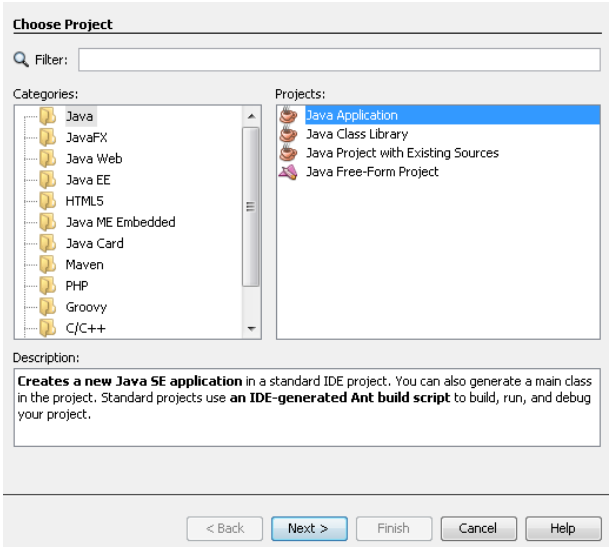


Fig.3.9.5 Aplicación Java

Fuente: Los Autores

2. Escoger un nombre para el proyecto, en este caso será *JavaArduino*, escoger su ubicación y presionar *Finish.*

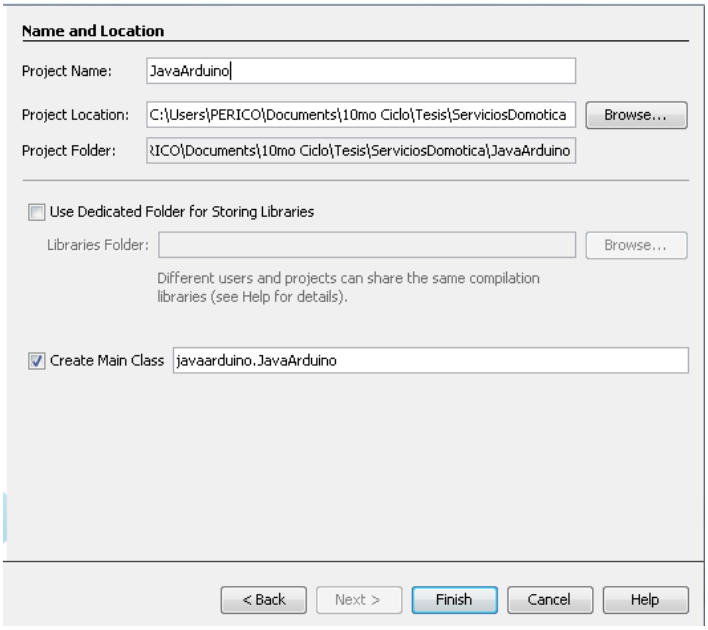


Fig.3.9.6 JavaArduino

Fuente: Los Autores

3. Crear la instancia del servicio web al que se conectará, dar clic derecho en el proyecto, presionar *New*, y escoger la opción *Web Service Client.*

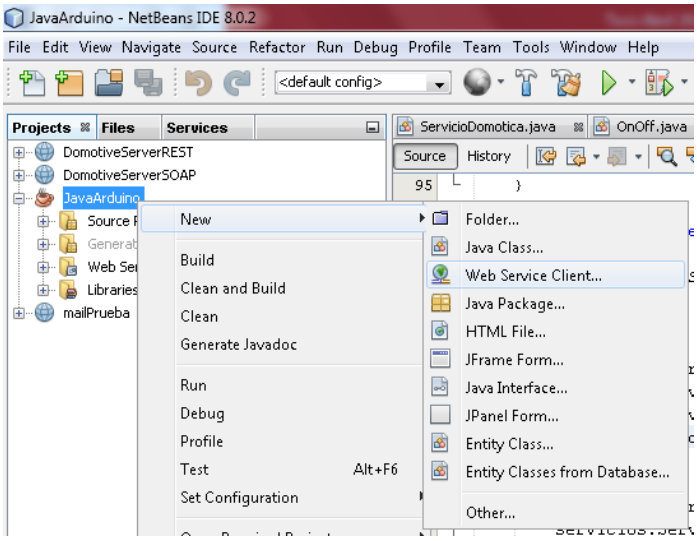


Fig.3.9.7 Cliente Web Service

Fuente: Los Autores

4. Seleccionar la opción *WSDL URL*, y escribir la dirección del archivo WSDL del proyecto creado con anterioridad, además se deberá escoger el paquete de trabajo en el que se generará el código del servicio web.

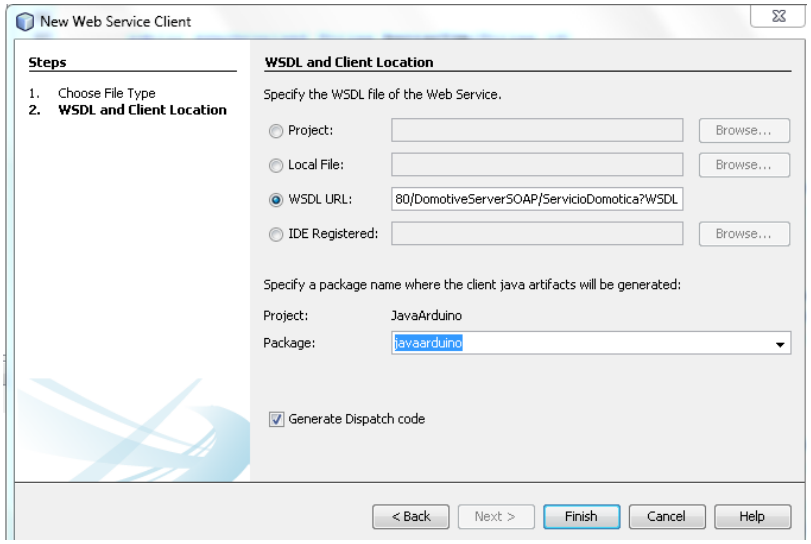


Fig.3.9.8 WSDL URL

Fuente: Los Autores

5. Automáticamente se genera una instancia dentro del proyecto, observar dentro de la carpeta *Web Service Reference*, que se ha creado dicha instancia con el nombre del servicio web creado annteriormente, al desplegar el mismo se pueden observar todos los métodos disponibles para consumir.

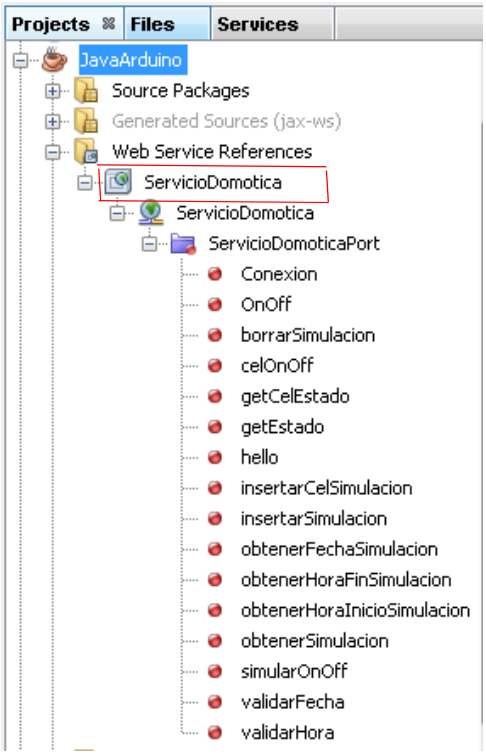


Fig.3.9.9 Referencias Web Service

Fuente: Los Autores

Codificación del proyecto JavaArduino

Código fuente *App.java*

Clase principal del proyecto, encargada de crear la conexión con los servidores.

```
public class App {

public Session session;
public Session session2;
  private static final Servidor SERVIDOR =
  new Servidor();
private static final Ard arduino = new
Ard();

protected void start()
   {
              WebSocketContainer container
  =
  ContainerProvider.getWebSocketContainer()
  ;
```

```
            String uri =
  "ws://localhost:8080/DomotiveServerSOAP/b
  roadcast";
        System.out.println("Connecting to "
  + uri);

            String uri2 =
  "ws://localhost:8080/DomotiveServerREST/b
  roadcast";
        System.out.println("Connecting to "
  + uri2);

        try
        {
            SERVIDOR.conectar();
                  cliente cl = new
  cliente(SERVIDOR, arduino);
            session =
  container.connectToServer(cl,
  URI.create(uri));
            session2 =
  container.connectToServer(cl,
  URI.create(uri2));
        }
        catch (DeploymentException e)
        {
            e.printStackTrace();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }

}
public static void main(String args[]){
    App client = new App();
    client.start();

          ProcesoLectura leer = new
  ProcesoLectura(true);
    leer.setArduino(arduino);
    leer.setConexionServidor(SERVIDOR);
    leer.start();

          BufferedReader br = new
  BufferedReader(new
  InputStreamReader(System.in));
    String input = "";
    try {
        do{
            input = br.readLine();
            if(!input.equals("exit"))

        client.session.getBasicRemote().sen
        dText(input);

        client.session2.getBasicRemote().se
        ndText(input);

        }while(!input.equals("exit"));
```

```
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Código fuente *ProcesoLectura.java*

Clase encargada de leer constantemente el estado de la lámpara y notificar en el momento que existen cambios en el mismo.

```
public class ProcesoLectura extends Thread{

    private Ard arduino;
    private Servidor SERVIDOR;
    private boolean lectura = true;

    public ProcesoLectura(boolean leer)
    {
        super();
    }

    @Override
    public void run()
    {
        System.out.println("result: ");
        //if(SERVIDOR.conectar())
        //{
            System.out.println("se conecto");
            String estadoAnt = "";
            while (lectura)
            {
                String estado = arduino.leer();
                            if
            (estado.equals("encendido") &&
            (estadoAnt.equalsIgnoreCase(estado)
            ==false))
                {

                  SERVIDOR.encender("on", "1");
                    estadoAnt = estado;

                  System.out.println("ENCENDIO"
                  );
                }
                            else if
            (estado.equals("apagado")&&
            (estadoAnt.equalsIgnoreCase(estado)
            ==false))
                {

                  SERVIDOR.apagar("off", "1");
                    estadoAnt = estado;
```

```
                System.out.println("APAGO");
              }
            }
        //}
    }

    public void setLeer(boolean l)
    {
        lectura = l;
    }

    public void setArduino(Ard ard)
    {
        arduino = ard;
    }

      public void setConexionServidor(Servidor
      serv)
    {
        SERVIDOR = serv;
    }
}
```

Código fuente *Notificaciones.java*

Clase que se encarga de recibir las notificaciones que vienen desde el servidor, con el fin de actualizar el estado de la lámpara (encendida o apagada) según sea el caso.

```
public class Notificaciones {
    private String tipo;
    private String mensaje;

    public Notificaciones(String notificacion)
    {
       try
       {
            int indice =
notificacion.indexOf(",");
            tipo = notificacion.substring(0,
indice);

            mensaje =
notificacion.substring(indice+1,
notificacion.length());
       }
       catch(Exception ex)
       {
           System.out.println(ex);
       }
    }

    public Notificaciones(String tipo, String
mensaje)
    {
        this.tipo = tipo;
```

```
        this.mensaje = mensaje;
    }

    public String getTipo()
    {
        return tipo;
    }

    public void setTipo(String tipo)
    {
        this.tipo = tipo;    }

    public String getMensaje()
    {
        return mensaje;
    }

    public void setMensaje(String mensaje)
    {
        this.mensaje = mensaje;
    }

    public void realizarOperacion(Servidor
serv, Ard arduino)
    {
        if(tipo.equalsIgnoreCase("lamp"))
        {
            if (mensaje.equalsIgnoreCase("on"))
            {
                try
                {
                    arduino.encender();
                    System.out.println("Se
prendio la luz");
                }
                catch (Exception ex)
                {
                    System.out.println("No se
puede ejecutar la operación");

Logger.getLogger(Window.class.getName()).log(Le
vel.SEVERE, null, ex);
                }
            }
            else if
(mensaje.equalsIgnoreCase("off"))
            {
                try
                {
                    if (serv.conectar())
                    {
                        arduino.apagar();
                        System.out.println("Se
apago la luz");
                    }
                    else
                    {
                        System.out.println("No
se puede ejecutar la operación");
                    }
                }
```

```
                catch (Exception ex)
                {
Logger.getLogger(Window.class.getName()).log(Le
vel.SEVERE, null, ex);
                }
            }
        }
        else if(tipo.equalsIgnoreCase("sim"))
        {
            System.out.println("Esta en
simulacion");
        }
    }
}
```

Conclusiones del capítulo

Una vez que se ha encontrado una solución para el dispositivo electrónico que controlará el encendido y apagado de la lámpara, para la base de datos que gestionará el sistema y para la implementación de cada uno de los servicios que ofrece el servidor a sus clientes para controlar el sistema de iluminación, se podrá consumir dichos servicios desde las aplicaciones móviles además de configurar y poner en marcha el sistema de vigilancia junto con el sistema de iluminación ya desarrollado, esto se describirá en los próximos capítulos.

4. CAPÍTULO 4: Sistemas de Control

Introducción

En este capítulo se describirá la implementación del sistema de vigilancia, se indicará la cámara IP que se va a utilizar, su proceso de instalación y configuración para poder observar la imagen que captura en tiempo real, además se describirán los módulos del sistema de iluminación con los que el usuario podrá interactuar.

4.1.Sistema de Vigilancia

El sistema de vigilancia del sistema domótico que se está implementando permitirá observar en tiempo real lo que sucede en el interior del hogar a través de la utilización de una cámara IP.

Una cámara IP, también conocida como cámara de red, ha sido utilizada desde su aparición en 1996 hasta el día de hoy dentro de múltiples escenarios, como soluciones de seguridad en industrias, laboratorios, bancos, aeropuertos, oficinas, casinos, etc. (RNDS)

Para entender de manera más clara la funcionalidad de esta cámara dentro del sistema de vigilancia, a continuación se cita un concepto de cámara IP:

“Una cámara IP o también conocida como cámara de red puede ser descrita como la combinación de una cámara y una computadora en una sola unidad, la cual captura y transmite imágenes en vivo a través de una

red IP, habilitando a usuarios autorizados a ver, almacenar y administrar el video sobre una infraestructura de red estándar basada en el protocolo IP, una cámara de red tiene su propia dirección IP, se conecta a la red, tiene enlazadas una serie de aplicaciones, funciones y servicios como un servidor web, un servidor FTP, cliente de correos, administración de alarmas y muchos otros que en su conjunto permiten inclusive realizar programación directamente en la cámara.” (Ycezalaya)

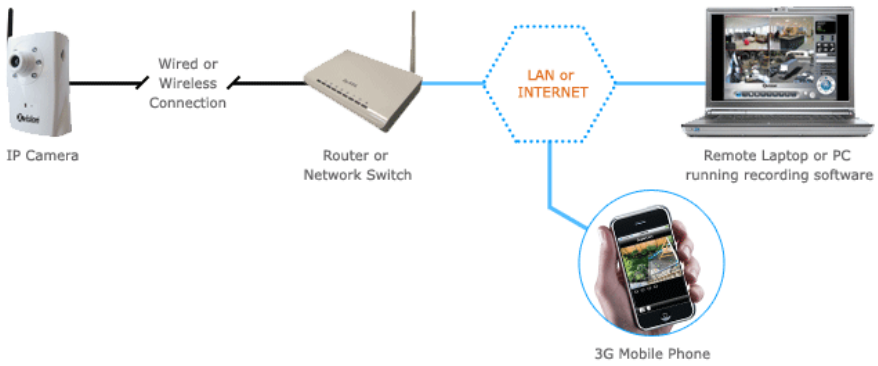


Fig.4.1.1 Concepto Cámara IP

Fuente: http://www.ipcctv.com/article.php?xArt=13

Algunas de las principales características de las cámaras IP son:

- **Tecnología digital:** utilizan tecnología digital para la captura, transmisión y almacenamiento de la imagen de video en lugar de tecnología analógica como es el caso de las cámaras de CCTV.
- **Acceso Remoto:** permite acceso en todo momento y en todo lugar desde cualquier computador que tenga acceso a Internet.
- **Independencia:** no necesita estar conectada a ningún ordenador que actúe como servidor, posee un sistema operativo y un

servidor web propio que le permite conectarse a cualquier punto de red para empezar a transmitir.

- **Wi-fi:** permite realizar la transmisión de la imagen inalámbricamente sin necesidad de un cable de red.
- **Funcionalidades extras:** algunas cámaras IP poseen funcionalidades especiales tales como detección de sonido y movimiento, actuadores de alarma, extensor de la señal de internet, etc.
- **Escalabilidad:** en caso de que un sistema de vigilancia requiera de varias cámaras IP, simplemente se debe conectar dichas cámaras a un nuevo punto de red (LAN router / Switch). Ver fig.7.7.
- **Transmisión Múltiple:** debido a la utilización del protocolo TCP/IP para la transmisión de la imagen, una cámara IP puede transmitir a más de 10 ordenadores al mismo tiempo, ya sea a través de una red LAN(Local Area Network) o a través de Internet. .

(Dlink, 2013)

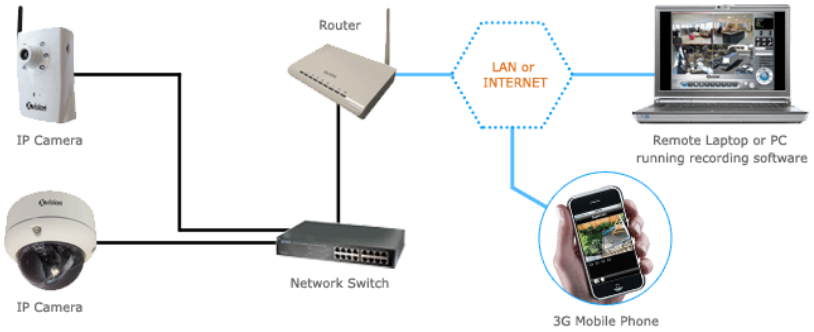


Fig.4.1.2. Concepto Múltiples Cámaras IP

Fuente: http://www.ipcctv.com/article.php?xArt=13

Otra característica a tener en cuenta es el formato de compresión que se va a utilizar para la transmisión de la imagen, ya que mientras más detallada sea ésta, mayor ancho de banda se requerirá para su

transmisión. Para que el tamaño de la imagen comprimida disminuya se debe transmitir una menor cantidad de *frames* por segundo, teniendo en cuenta que al disminuir el número de *frames* se obtendrá una imagen de menor calidad.

A continuación se explicará paso a paso el proceso de instalación y configuración de la cámara IP para la posterior implementación del sistema de vigilancia:

Instalación

La cámara IP a utilizar para la implementación es la cámara Dlink: *DCS-931L Wireless N H.264 Network Camera* que se muestra a continuación:

Fig.4.1.3 DCS-931L Network Camera / Vista Frontal

Fuente: ftp.dlinkla.com

Fig.4.1.4. DCS-931L Network Camera / Vista Posterior

Fuente: ftp.dlinkla.com

1. Seleccionar el idioma de configuración de la cámara IP y hacer click en Iniciar.

Fig.4.1.5 Idioma Instalación

Fuente: Los Autores

2. Aceptar las condiciones de la licencia y hacer click en Siguiente.

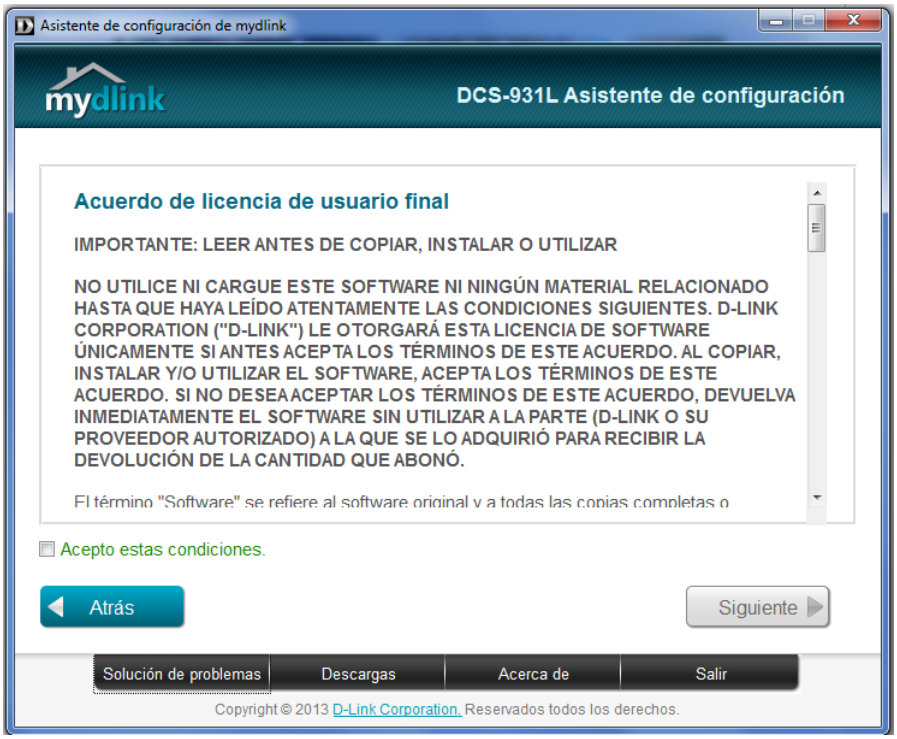


Fig.4.1.6 Acuerdo Licencia

Fuente: Los Autores

3. Conectar el cable Ethernet en la parte posterior de la cámara y hacer click en Siguiente.

Fig.4.1.7 Cable Ethernet

Fuente: Los Autores

4. Conectar el otro extremo del cable Ethernet en un puerto LAN del router.

Fig.4.1.8 Puerto LAN

Fuente: Los Autores

5. Conectar el cable de alimentación de la cámara IP.

Fig.4.1.9 Adaptador Cámara

Fuente: Los Autores

6. Verificar que el LED de la parte posterior de la cámara IP haya cambiado de color rojo a verde.

Fig.4.2.1 LED Verde

Fuente: Los Autores

7. Esperar que el asistente de configuración encuentre la cámara IP conectada anteriormente.

Fig.4.2.2Buscar Cámaras

Fuente: Los Autores

8. Una vez que la cámara IP haya sido encontrada y reconocida, le asignamos un nombre de administrador y contraseña.

Fig.4.2.3 Cámara Encontrada

Fuente: Los Autores

9. Elegir si la transmisión de la imagen se va a realizar a través de cable o inalámbricamente.

Fig.4.2.4 Opciones Conexión

Fuente: Los Autores

10. Crear una cuenta de usuario en mydlink.com.

Fig.4.2.5 Servicios mydlink

Fuente: Los Autores

11. Una vez que se observe la imagen que está transmitiendo la cámara en tiempo real, hacer click en finalizar.

Fig.4.2.6 Configuración Finalizada

Fuente: Los Autores

12. Activar la cuenta de mydlink.com por medio del email de confirmación.

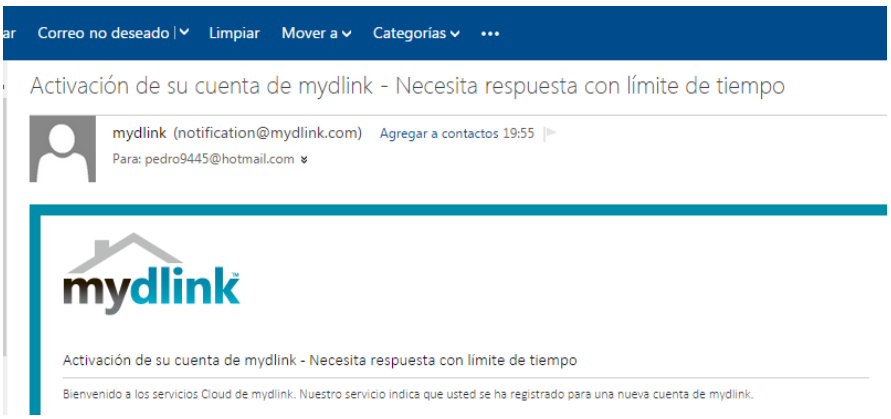


Fig.4.2.7 Activación mydlink

Fuente: Los Autores

Una vez finalizada la instalación y configuración de la cámara IP se puede ver la transmisión de la misma de 2 formas distintas:

Ingresando a mydlink.com con el nombre de administrador y la contraseña creada anteriormente.

Fig.4.2.8 Inicio Sesión mydlink

Fuente: Los Autores

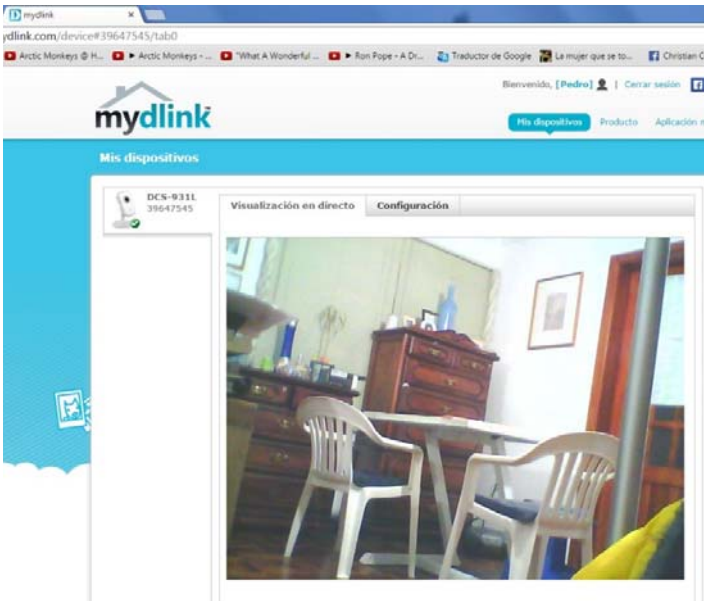


Fig.4.2.9 Imagen Transmición

Fuente: Los Autores

Digitando la dirección IP de la cámara en la barra de búsqueda de un navegador web.

Para obtener la dirección IP de la cámara se debe acceder a la web de configuración del router y buscar la lista de clientes DHCP conectados al mismo, en dicha lista se podrá encontrar la dirección correspondiente a la cámara como se muestra a continuación:

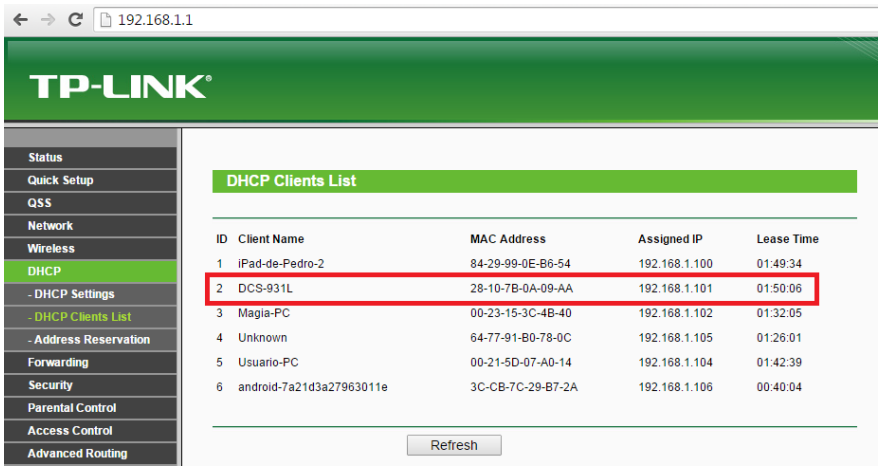


Fig.4.3.1 Dirección IP Cámara

Fuente: Los Autores

Una vez que se conoce la dirección IP de la cámara se puede acceder a una transmisión en vivo de la imagen capturada sin

necesidad de ingresar al sitio mydlink.com como se puede observar en la siguiente imagen.

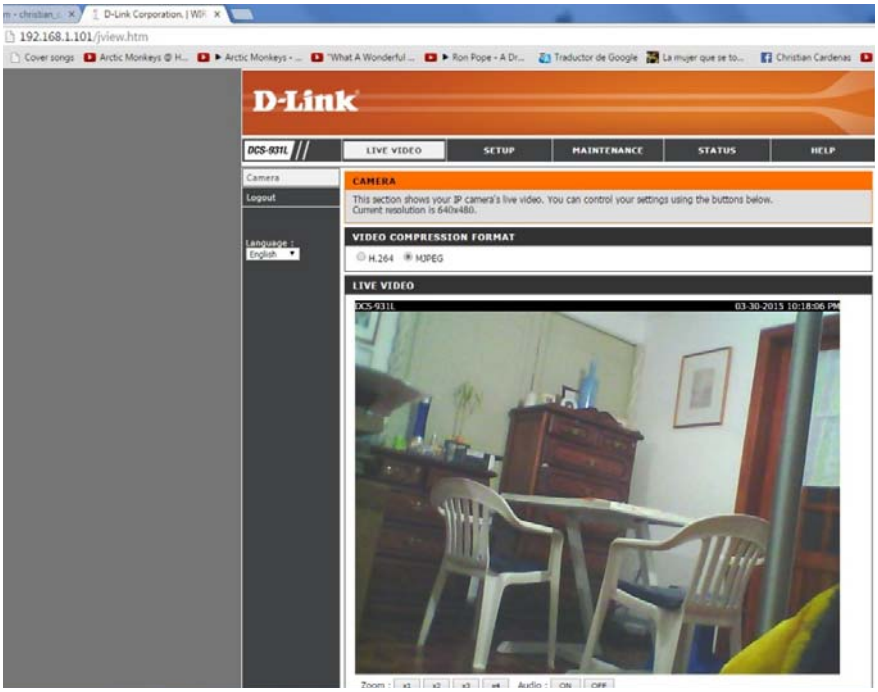


Fig.4.3.2 Video Cámara IP

Fuente: Los Autores

Además dentro de este sitio web se puede obtener la información completa acerca de la configuración de la cámara, dentro de la pestaña *Status.*

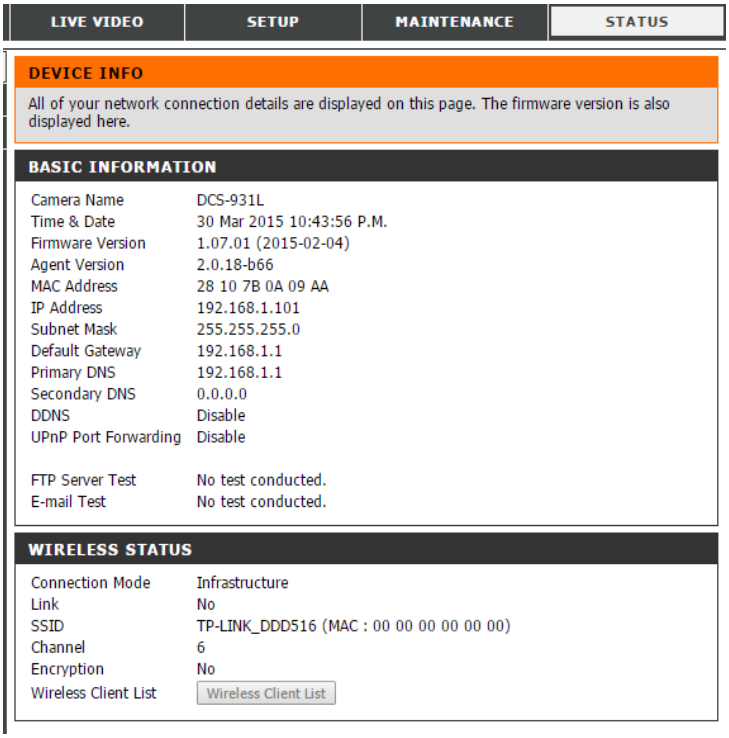


Fig.4.3.3 Información Cámara IP

Fuente: Los Autores

Finalmente se necesita elegir la resolución y la cantidad de *frames* por segundo que se van a transmitir según las necesidades que se presenten más adelante en la codificación de las aplicaciones móviles. Para configurar estas características se debe acceder a las pestaña *Setup.*

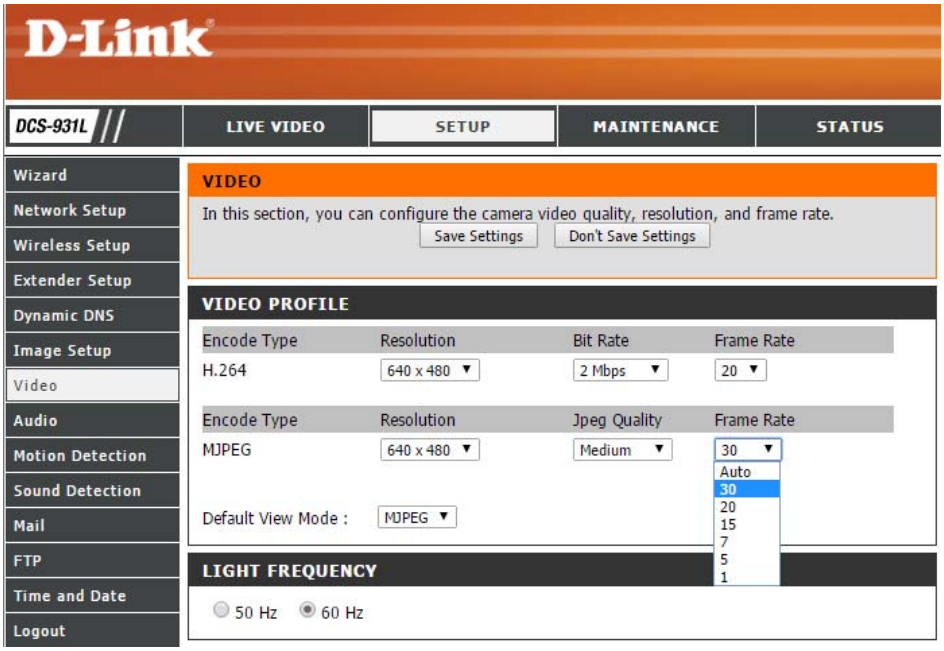


Fig.4.3.4 Perfil de video

Fuente: Los Autores

Con todos lo mencionado anteriormente, la imagen que transmite la cámara IP será capturada y mostrada en los dispositivos móviles por las aplicaciones cliente alojadas en los mismos, el procedimiento para la captura de la imagen en tiempo real se explicará a detalle en el siguiente capítulo.

4.2.Sistema de Iluminación

El sistema de Iluminación implementado en el capítulo 3 será visto por el usuario como un sistema compuesto por 3 módulos que se describen a continuación:

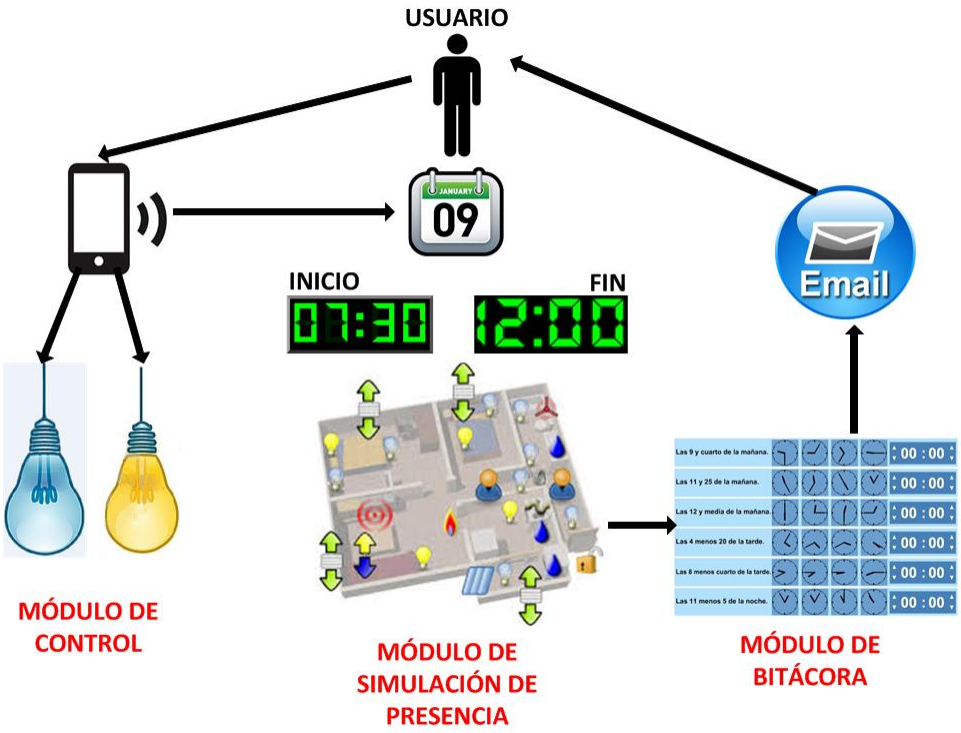


Fig.4.3.5. Módulos Sistema Iluminación

Fuente: Los Autores.

Módulo de Control

Este módulo es el que permite al sistema domótico encender o apagar la lámpara, el usuario a través de la aplicación alojada en los dispositivos móviles encenderá o apagará la lámpara en el momento que lo desee llamando a los servicios que se encuentran disponibles en el servidor.

Módulo de Simulación de Presencia (Modo Automático):

Este módulo permite al usuario programar tantas simulaciones de presencia como desee, es decir, cuando necesite abandonar su hogar y no exista ninguna persona dentro del mismo, puede hacer que la lámpara se encienda y se apague de manera automática a las horas que él desee para que se produzca la sensación de que alguien se encuentra en el interior del hogar. Para esto el usuario deberá indicar al sistema la fecha, la hora de inicio y la hora de finalización de cada simulación que desea ingresar.

Módulo de Bitácora:

Este módulo es el encargado de almacenar los distintos eventos que se producen en la lámpara y generar una bitácora (historial) de los mismos, dichos eventos corresponden al encendido y apagado de la lámpara, además de almacenar la bitácora en la base de datos mencionada en el capítulo 3.3, este módulo envía al usuario notificaciones mediante correo electrónico indicándole que la lámpara ha sido encendida en ese instante. A continuación se muestra la estructura de la bitácora.

id Clave identificadora de cada lámpara	operacion Indica la operación que se realiza sobre la lámpara	fecha Fecha y hora en la que se realizó la operación sobre la lámpara.
1	on	2015-03-26 11:46:25
1	off	2015-03-26 11:46:29
1	on	2015-03-26 11:46:33
1	off	2015-03-26 11:46:36
1	on	2015-03-26 11:46:39
1	off	2015-03-26 11:46:42
1	off	2015-03-26 11:47:42
1	on	2015-03-26 11:51:12
1	off	2015-03-26 11:51:13
1	on	2015-03-26 11:53:49
1	off	2015-03-26 11:53:49
1	on	2015-03-26 11:54:15

Fig.4.3.6 Bitácora Sistema Iluminación

Fuente: Los Autores.

El formato de las notificaciones enviadas al usuario mediane correo electrónico se observa a continuación:

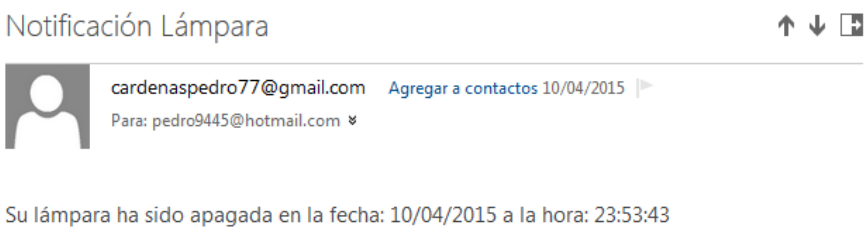


Fig.4.3.7 Notificación correo electrónico

Fuente: Los Autores

Conclusiones del capítulo

Ahora que el sistema de iluminación y el sistema de vigilancia han sido implementados, y se conoce su estructura interna, su funcionamiento, las tecnologías que utilizan y los protocolos de comunicación a través de los que se comunicarán, ambos sistemas estarán disponibles y podrán ser consumidos por las aplicaciones móviles, en el siguiente capítulo se explicará a detalle la codificación de dichas aplicaciones.

5. CAPÍTULO 5: Aplicaciones Usuario:

Introducción

En este capítulo se explicará el desarrollo y la codificación de las aplicaciones móviles, se describirá las tecnologías utilizadas por las mismas para consumir los servicios del servidor, en el caso de la aplicación iOS, se explicará la tecnología JSON mientras que en la aplicación Android, la tecnología SOAP. Se ilustrará los elementos que conforman la interfaz de usuario (GUI) de cada una de las pantallas de dichas aplicaciones, además se explicará cómo llamar a los servicios ofertados por el servidor desde los métodos que se implementarán.

5.1.Aplicación para iOS.

Para la codificación de la aplicación móvil iOS se utilizará el entorno de desarrollo XCode 6 junto con el lenguaje de programación objective c.

Fig.5.1.1. Entorno de Desarrollo XCode

Fuente: http://softpixelengine.sourceforge.net

Antes de observar la codificación de la aplicación, se explicará la tecnología utilizada para realizar la comunicación entre la aplicación cliente y el servidor, en este caso se utilizó la tecnología JSON (JavaScript Object Notation) ya que trabaja con un formato simple que puede ser generado e interpretado por varios lenguajes de programación:

Primero se necesita construir la cadena que va llamar al servidor para consumir sus servicios con el siguiente formato:

```
NSString *dir =

@"http://IPdelServidor/nombreDelServidor/nombreDelRecurso/ nombreDelMetodo?nombreParametro=";
```

Se tiene que convertir la cadena creada en una dirección del tipo

URL:

```
NSString* strUrl = [NSString
stringWithFormat:@"%@%@", dir,idLuz];

NSURL *url = [NSURL URLWithString:strUrl];
```

Agregar la dirección URL a un objeto del tipo NSURLRequest, para que pueda ser enviada como petición (request) al servidor:

```
NSURLRequest *request = [NSURLRequest
requestWithURL:url];
```

Finalmente se debe crear la conexión asíncrona (*sendAsynchronousRequest*) para comunicarse con el servidor, dentro de esta conexión se debe especificar la variable en donde se recibirá el valor retornado *(NSURLResponse*) por el servidor, y la llave (*ObjectForKey*) por la que el servicio sabrá el objeto JSON (*JSONObjectWithData*) que debe retornar:

```
[NSURLConnection sendAsynchronousRequest:request
queue:[NSOperationQueue mainQueue]
completionHandler:^(NSURLResponse *response,
NSData *data, NSError *connectionError)
{
  if (data.length > 0 && connectionError == nil)
  {
        NSDictionary *greeting =
        [NSJSONSerialization
        JSONObjectWithData:data
        options:0
        error:NULL];
        self.lbEstadoLuz.text = [greeting
        objectForKey:@"estado"];
```

Ahora se procede a desarrollar la aplicación cliente, al abrir el entorno de desarrollo Xcode se mostrará la siguiente pantalla donde se tiene que elegir el tipo de plantilla que tendrá el proyecto que se va a crear, en este caso se elige la opción *Empty Porject* para no generar ninguna opción predefinida o por defecto:

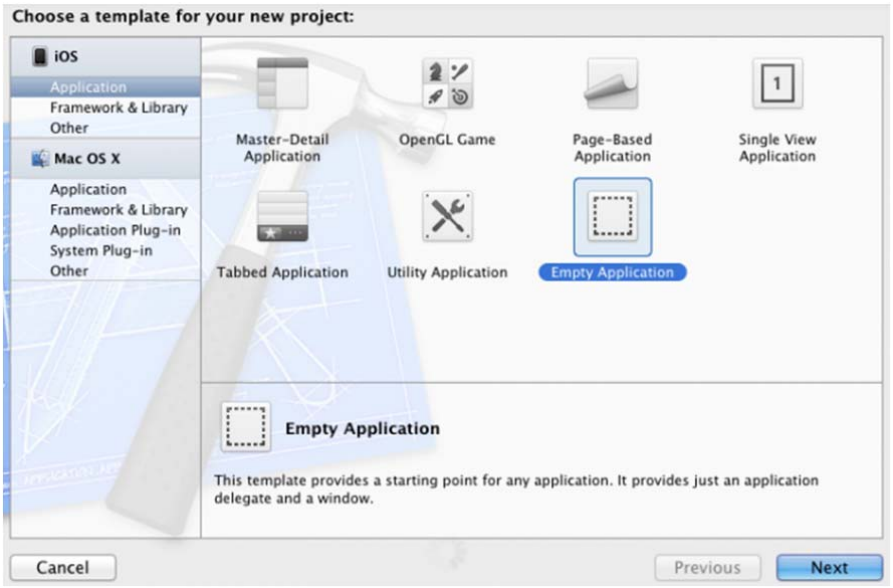


Fig.5.1.2 Aplicación Vacía

Fuente: Los Autores

En la siguiente pantalla se le asigna el nombre que va a tener el proyecto:

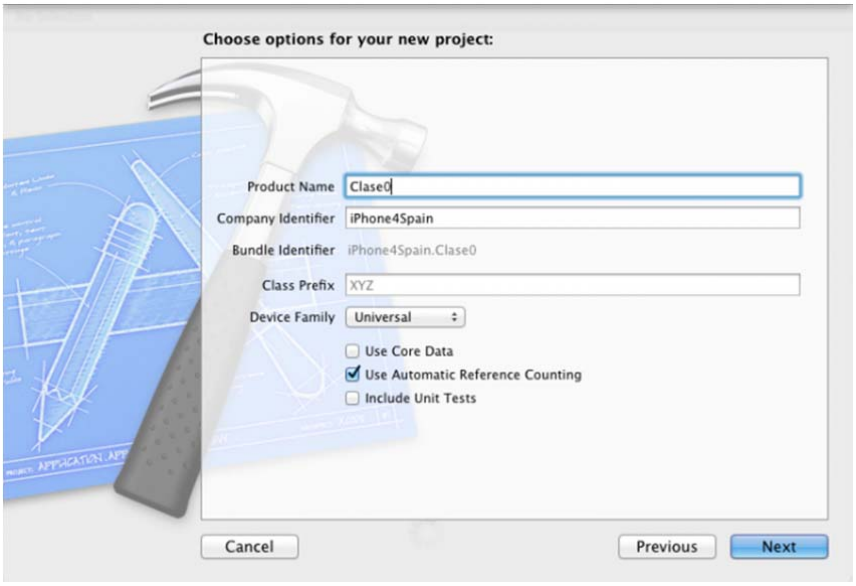


Fig.5.1.3 Nombre Proyecto

Fuente: Los Autores

A continuación se deben elegir las características de desarrollo del proyecto, en la opción *Devices* se debe indicar si la aplicación que se va a a desarrollar será para ejecutarse en un iPhone o en un iPad, además en la opción *Deployment Target* se debe indicar la versión iOS que sea compatible con la versión del sistema operativo que tiene el dispositivo móvil donde se va a ejecutar la aplicación:

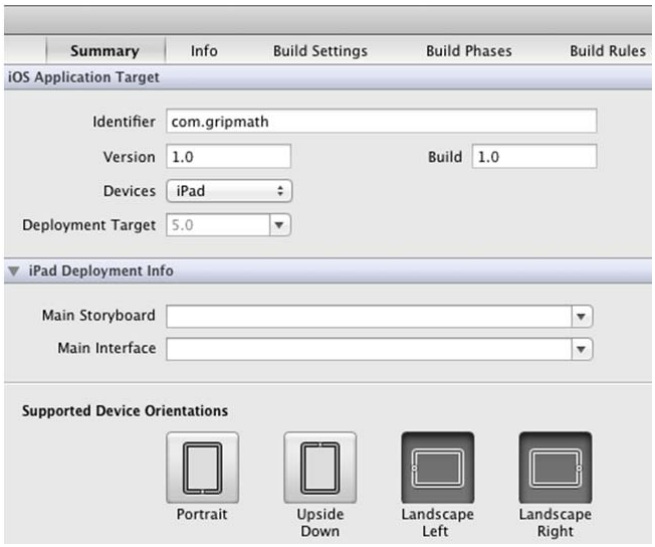


Fig.5.1.4 Versión IOs

Fuente: Los Autores

Cada vez que Xcode crea una clase se generan 2 archivos con el mismo nombre de la clase pero con distinta extensión, el uno con extensión .h y el otro con extensión .m, en el archivo .h se debe declarar todos los atributos y métodos que deben ser públicos, mientras que en el archivo .m se debe declarar los atributos que deben ser privados y realizar la implementación de los métodos declarados en el archivo .h. (Dept. Ciencia de la computación e IA, 2012-2013)

Ahora se mostrará cada pantalla que conforma la interfaz de la aplicación y se explicará su respectivo funcionamiento junto con el código correspondiente a los arhivos (.h y .m) que permiten consumir los servicios ofertados por el servidor:

Pantalla Módulo de Control (Sistema Iluminación)

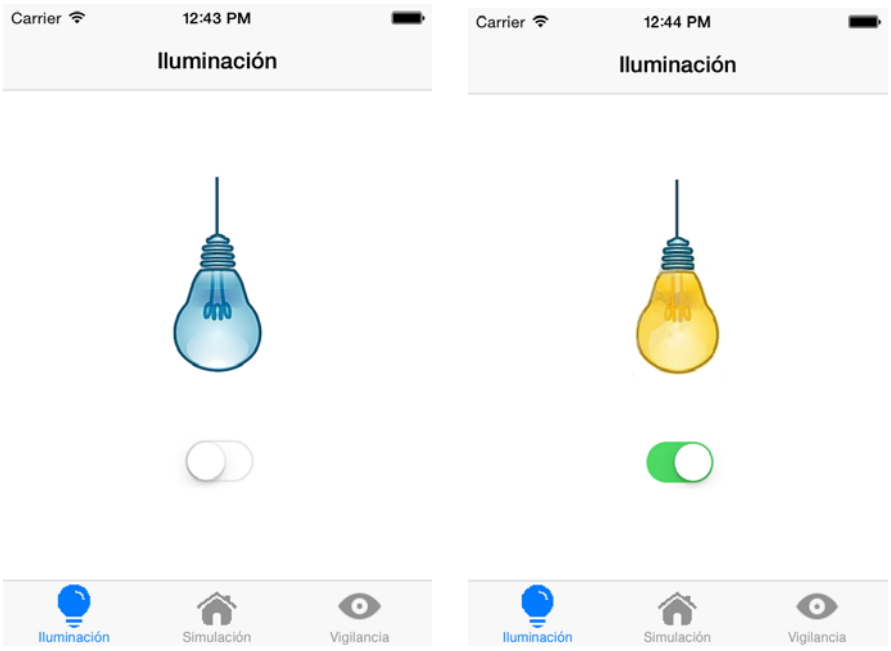


Fig.5.1.5 Pantalla Módulo de Control

Fuente: Los Autores

Funcionamiento:

En el instante que el usuario abre la aplicación, se realiza una llamada al servidor para leer el estado en que se encuentra la lámpara, es decir, si está encendida o apagada, al recibir la respuesta del servidor, después de haber consultado la base de datos, la imagen de la lámpara de la interfaz se mostrará como encendida o apagada según el estado de respuesta del servidor.

Una vez que el estado en el que se encuentra la lámpara física y el estado que muestra la imagen de la interfaz estén correctamente sincronizados, el usuario podrá pulsar el botón para encender o apagar la lámpara.

Codificación:

En el archivo .h correspondiente a esta pantalla se declaran las variables de cada uno de los elementos que se observan en la interfaz:

Para la imagen de la lámpara encendida y apagada se declara un elemento del tipo *UIImageView* con el nombre *miImagen*:

```
@property (weak, nonatomic) IBOutlet
UIImageView *miImagen;
```

Para el botón que generará el evento de encender o apagar la lámpara se declara un elemento tipo *Switch* con el nombre *miSwitch*:

```
@property (weak, nonatomic) IBOutlet UISwitch
*miSwitch;
```

Además se declara el método que se implementará en el archivo .m para ejecutar el evento que llamará al servidor con el nombre *moverSwitch*.

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

En el archivo .m correspondiente a esta pantalla se realiza la sincronización de la lámpara y la imagen mencionada anteriormente, además de la implementación del método *moverSwitch*:

Se declaran las variables necesarias:

```
@property NSString *soapMessage;
@property NSString *currentElement;
@property NSMutableData *webResponseData;
@synthesize soapMessage, webResponseData,
currentElement;
@synthesize miImagen;
@synthesize miSwitch;
```

```
@synthesize lbEstadoLuz;
```

El método *viewDidLoad* siempre se ejecuta automáticamente al abrir la aplicación por lo que aquí se realiza la llamada al servicio *obtenerEstadoLuz* implementado en el servidor en el capítulo 3.4.3 que realizará la sincronización entre la lámpara y la imagen:

```
- (void)viewDidLoad {
    [super viewDidLoad];
    NSString *idLuz = @"1";
    NSString *dir =
@"http://10.10.26.221:8080/DomotiveServerREST/w
ebresources/obtenerEstadoLuz?id=";
    NSString* strUrl = [NSString
stringWithFormat:@"%@%@", dir,idLuz];
    NSURL *url = [NSURL URLWithString:strUrl];
    NSURLRequest *request = [NSURLRequest
requestWithURL:url];
    [NSURLConnection
sendAsynchronousRequest:request

queue:[NSOperationQueue mainQueue]

completionHandler:^(NSURLResponse *response,



NSData *data, NSError *connectionError)
     {
         if (data.length > 0 && connectionError
== nil)
         {
             NSDictionary *greeting =
[NSJSONSerialization JSONObjectWithData:data

options:0

error:NULL];
             self.lbEstadoLuz.text = [greeting
objectForKey:@"estado"];
             self.lbEstadoLuz.hidden = TRUE;
             if([self.lbEstadoLuz.text
isEqualToString:@"on"])
             {
                 UIImage *imagen = [UIImage
imageNamed:@"prendido.gif"];
                 [miImagen setImage:imagen];
                 [miSwitch setOn:YES
animated:YES];
             }
             else
             {
                 UIImage *imagen = [UIImage
imageNamed:@"apagada.gif"];
                 [miImagen setImage:imagen];
```

```
                [miSwitch setOn:NO
animated:YES];
            }
        }
     }];
}
```

Implementación del método *moverSwitch* que llama al servicio *lampOnOff* implementado en el servidor en el capítulo 3.4.3, si se desea apagar la lámpara se tiene que enviar como parámetro la cadena “off”, caso contrario si se desea encender la misma se enviará la cadena “on”:

```
- (IBAction)moverSwitch:(id)sender
{

    if(miSwitch.on)
    {
        NSString *estado = @"on";
        NSString *dir =
@"http://10.10.26.221:8080/DomotiveServerREST/w
ebresources/lampOnOff?estado=";
        NSString* strUrl = [NSString
stringWithFormat:@"%@%@", dir,estado];
        NSURL *url = [NSURL
URLWithString:strUrl];
        NSURLRequest *request = [NSURLRequest
requestWithURL:url];
        [NSURLConnection
sendAsynchronousRequest:request

queue:[NSOperationQueue mainQueue]

completionHandler:^(NSURLResponse *response,

NSData *data, NSError *connectionError)
         {
             if (data.length > 0 &&
connectionError == nil)
             {
                 NSDictionary *greeting =
[NSJSONSerialization JSONObjectWithData:data

options:0

error:NULL];
                 self.lbEstadoLuz.text =
[greeting objectForKey:@"estado"];
                 UIImage *imagen = [UIImage
imageNamed:@"prendido.gif"];
                 [miImagen setImage:imagen];
             }
         }];
    }
    else
    {
        NSString *estado = @"off";
```

```
        NSString *dir =
@"http://10.10.26.221:8080/DomotiveServerREST/w
ebresources/hola?estado=";
        NSString* strUrl = [NSString
stringWithFormat:@"%@%@", dir,estado];
        NSURL *url = [NSURL
URLWithString:strUrl];
        NSURLRequest *request = [NSURLRequest
requestWithURL:url];
        [NSURLConnection
sendAsynchronousRequest:request

queue:[NSOperationQueue mainQueue]

completionHandler:^(NSURLResponse *response,

NSData *data, NSError *connectionError)
         {

             if (data.length > 0 &&
connectionError == nil)
             {
                 NSDictionary *greeting =
[NSJSONSerialization JSONObjectWithData:data

options:0

error:NULL];

                 self.lbEstadoLuz.text =
[greeting objectForKey:@"estado"];
                 UIImage *imagen = [UIImage
imageNamed:@"apagada.gif"];
                 [miImagen setImage:imagen];
             }
         }];
    }
}
@end
```

Pantalla Simulaciones (Módulo de Simulación de Presencia)

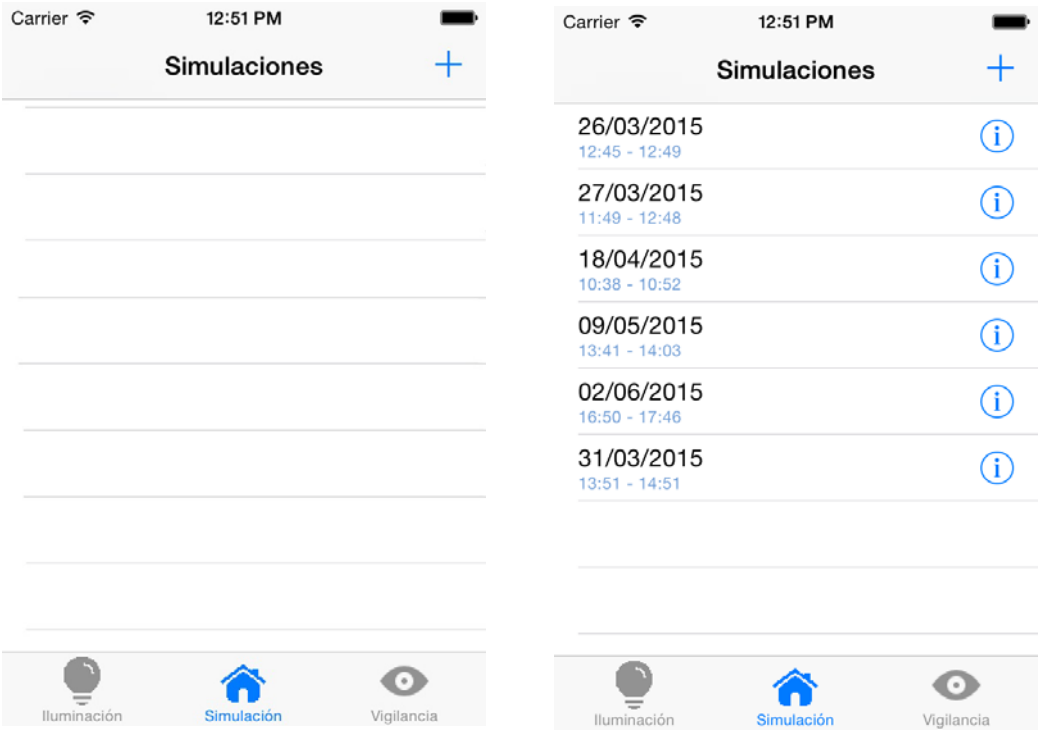


Fig.5.1.6 Pantalla Simulaciones

Fuente: Los Autores

Funcionamiento:

Al navegar hasta esta pantalla pulsando la opción *Simulación* en el menú inferior de la aplicación, se observarán todas las simulaciones que han sido ingresadas por el usuario y que todavía no han sido ejecutadas por el sistema domótico debido a que su fecha de activación es mayor a la fecha actual. En caso de no existir simulaciones ingresadas la lista se mostrará vacía.

Codificación:

En el archivo .h correspondiente a esta pantalla se declaran las variables necesarias para mostrar la lista de simulaciones que se observa en la interfaz:

Para que las simulaciones se muestren en forma de lista se declara un elemento tabla del tipo *UITableViewController* con el nombre *ToDoListTableViewController*:

```
@interface ToDoListTableViewController :
UITableViewController
```

Para que cada simulación se muestre en una fila de la tabla se declara un elemento del tipo *NSObject* con su respectivo identificador de tipo *NSString* llamado *itemName:*

```
@interface ToDoItem : NSObject;
@property NSString *itemName;
```

Para que se pueda observar en la tabla la fecha, hora de inicio y hora de fin de cada simulación se declaran las siguientes variables:

```
@property NSString *horaIn;
@property NSString *horaFin;
@property (readonly) NSDate *creationDate;
```

Para que el contenido de la tabla se actualice con las simulaciones se declara el método *unwindToList* que se implementará en el archivo .m:

```
- (IBAction)unwindToList:(UIStoryboardSegue
*)segue;
```

En el archivo .m correspondiente a esta pantalla se realiza la consulta de la base de datos de las simulaciones ingresadas por el usuario, además de la implementación del método *unwindToList*:

Se declaran las variables necesarias:

```
@interface ToDoListTableViewController ()
@property NSMutableArray *toDoItems;
@implementation ToDoListTableViewController
```

El método *viewDidLoad* siempre se ejecuta automáticamente al movernos hasta esta pantalla, por lo que aquí se invoca a otro método llamado *loadInitialData* que se encargará de cargar la tabla con las simulaciones obtenidas del servidor:

```
- (void)viewDidLoad {
        [self loadInitialData];
         //[super viewDidLoad];
            self.toDoItems = [[NSMutableArray
            alloc]init];
            self.toDoItems = [[NSMutableArray
            alloc]init];
            [self.tableView reloadData];
}
```

El método *loadInitialData* realiza la llamada al servicio *obtenerSimulacion* implementado en el servidor en el capítulo 3.4.3, se debe enviar como parámetro el *id* de la lámpara para que retorne las simulaciones pendientes de dicha lámpara:

```
- (void)loadInitialData
{
    NSString *idLuz = @"1";
    NSString *dir =
@"http://10.10.26.221:8080/DomotiveServerREST/w
ebresources/obtenerSimulacion?id=";
    NSString* strUrl = [NSString
stringWithFormat:@"%@%@", dir,idLuz];
    NSURL *url = [NSURL URLWithString:strUrl];
    NSURLRequest *request = [NSURLRequest
requestWithURL:url];
    [NSURLConnection
sendAsynchronousRequest:request

queue:[NSOperationQueue mainQueue]

completionHandler:^(NSURLResponse *response,

NSData *data, NSError *connectionError)
     {


         if (data.length > 0 && connectionError
         == nil)
         {
             NSString *jSON = [[NSString
alloc]initWithData:data
encoding:NSUTF8StringEncoding];
             while(jSON.length > 65)
             {
                 int cont = 67;
```

```
                    simulacion *sim =
            [[simulacion alloc]init];
                 ToDoItem *item1 = [[ToDoItem
            alloc]init];
                 item1.itemName = [sim
            getFecha:jSON];
                 item1.horaIn = [sim
            getHoraIn:jSON];
                 item1.horaFin = [sim
            getHoraFin:jSON];
                 [self.toDoItems
            addObject:item1];

                 if(jSON.length < 67){
                     cont = cont - 2;
                 }

                                  jSON = [jSON
                  substringWithRange:NSMakeRang
                  e(cont, jSON.length-cont)];
              }
              [self.tableView reloadData];
          }
      }];
}
```

Implementación del método *unwindToList* que se encarga de añadir los objetos a la vista de la tabla, es decir, las simulaciones, además de recargar su contenido:

```
- (IBAction)unwindToList:(UIStoryboardSegue
*)segue
{
    VSViewController *source = [segue
sourceViewController];
    ToDoItem *item = source.toDoItem;
    if(item.itemName != nil)
    {
        [self.toDoItems addObject:item];
        [self.tableView reloadData];
    }

}
```

Métodos para dar formato a la fecha y horas que devolvió el servidor y así puedan ser añadidas a la tabla de simulaciones:

```
@property NSString *idLuz;
@property NSString *fecha;
@property NSString *horaIn;
```

```
@property NSString *horaFin;

- (NSString *)getFecha:(NSString*)simulacion;
- (NSString *)getHoraIn:(NSString*)simulacion;
- (NSString *)getHoraFin:(NSString*)simulacion;
@end


- (NSString *)getFecha:(NSString*)simulacion {
    NSString *fecha = [simulacion
substringWithRange:NSMakeRange(54, 10)];
    return fecha;
}

- (NSString *)getHoraIn:(NSString*)simulacion;
{
    NSString *horaIn = [simulacion
substringWithRange:NSMakeRange(20, 5)];
    return horaIn;
}

- (NSString *)getHoraFin:(NSString*)simulacion;
{
    NSString *horaFin = [simulacion
substringWithRange:NSMakeRange(38, 5)];
    return horaFin;
}
@end
```

Los siguientes métodos permiten cambiar el comportamiento de la tabla cuando se agrega una nueva simulación a la misma, son métodos que vienen implementados por defecto junto con la tabla por lo que no se deben modificar a menos que sea necesario para satisfacer una necesidad específica:

```
(NSInteger)numberOfSectionsInTableView:(UITable
View *)tableView {
    // Return the number of sections.
    return 1;
}


- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {
    // Return the number of rows in the
section.
    return [self.toDoItems count];
}
```

```
- (UITableViewCell *)tableView:(UITableView
*)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath {
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"ListPrototy
peCell" forIndexPath:indexPath];

    ToDoItem *toDoItem = [self.toDoItems
objectAtIndex:indexPath.row];
    cell.textLabel.text = toDoItem.itemName;

    NSString* horas = [NSString
stringWithFormat:@"%@%@%@", toDoItem.horaIn,@"
- ",toDoItem.horaFin];
    cell.detailTextLabel.text = horas;
    return cell;
}
```

Pantalla Añadir Simulación (Módulo de Simulación de Presencia)

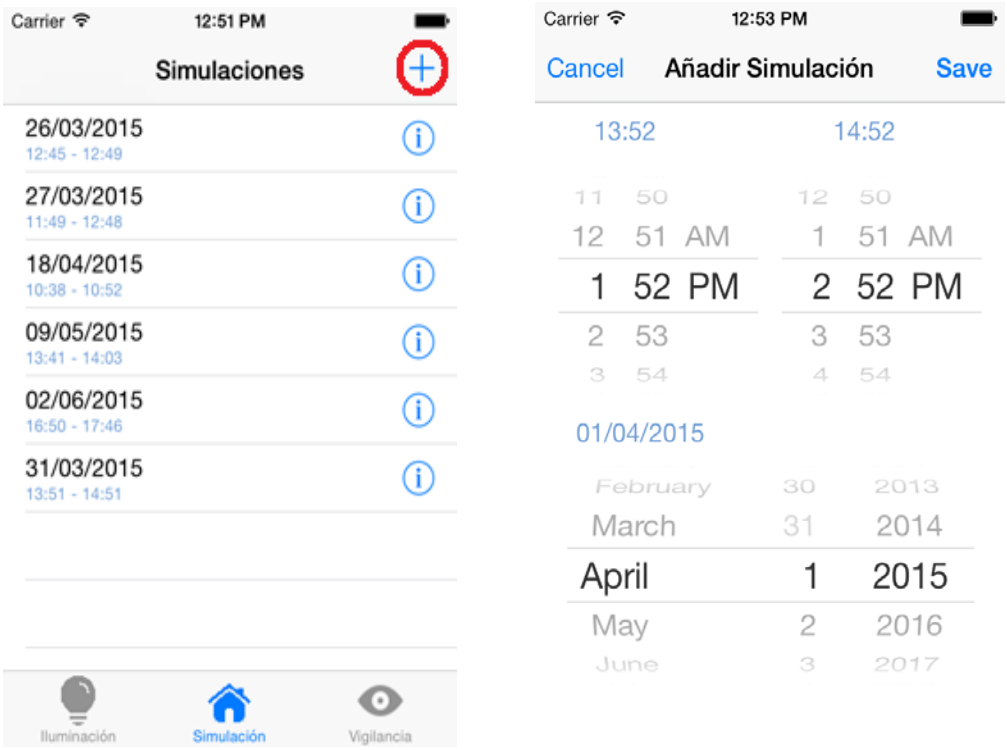


Fig.5.1.7 Pantalla Agregar Simulación

Fuente: Los Autores

Funcionamiento:

Para agregar una nueva simulación se debe pulsar el signo “+” que se encuentra en la esquina superior derecha de la pantalla de *Simulaciones*. Con esta acción se abrirá la nueva pantalla *Añadir Simulación* que permitirá al usuario seleccionar la hora de inicio, la hora de finalización y la fecha en la que desea programar una nueva simulación, una vez escogidos estos datos se deberá pulsar el botón *Save* para guardar la simulación, la cual automáticamente aparecerá en la lista de simulaciones ingresadas.

Codificación:

En el archivo .h correspondiente a esta pantalla se declaran las variables necesarias para elegir la fecha y horas de la simulación:

Los elementos que permiten al usuario deslizar el dedo hacia arriba o hacia abajo para escoger una hora o fecha específica de manera rápida y eficiente tienen el nombre de *UIDatePicker*.

Se necesita declarar una variable distinta del tipo *UIDatePicker* para la selección de la hora de inicio, otra para la hora de finalización y otra para la fecha:

```
@property (weak, nonatomic) IBOutlet
UIDatePicker *myDatePicker;
@property (weak, nonatomic) IBOutlet
UIDatePicker *myDatePicker2;
@property (weak, nonatomic) IBOutlet
UIDatePicker *myDatePicker3;
```

Se declara los *labels* en los que se mostrarán las horas y la fecha seleccionada en sus respectivos DatePickers.

```
@property (weak, nonatomic) IBOutlet UILabel
*selectedDate;
@property (weak, nonatomic) IBOutlet UILabel
*selectedDate2;
@property (weak, nonatomic) IBOutlet UILabel
*selectedDate3;
```

Se declara el botón *Save* situado en la parte superior derecha como un elemento del tipo *UIBarButtonItem* con el nombre *saveButton:*

```
@property (weak, nonatomic) IBOutlet
UIBarButtonItem *saveButton;
```

Definir el tiempo de duración que girará el *DatePicker* al deslizar el dedo:

```
#define kPickerAnimationDuration    0.40
```

Definir el número de columnas en las que se va a mostrar la información del *DatePicker*:

```
#define kDateStartRow   1
#define kDateEndRow     2
```

En el archivo .m correspondiente a esta pantalla se realiza la transferencia de la fecha y las horas seleccionadas a los campos de texto correspondientes situados encima de cada DatePicker, aquí se muestran los valores escogidos para que luego sean ingresados a la base de datos a través de la llamada al servicio *modoAutomatico* implementado en el servidor en el capítulo 3.4.3, se debe enviar como parámetros el id de la lámpara, la hora de inicio, la hora de fin y la fecha de la simulación, la llamada a este servicio se realiza dentro del método *prepareForSegue*:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
```

```
    [self.myDatePicker addTarget:self
action:@selector (datePickerChanged:)
forControlEvents:UIControlEventValueChanged];

    [self.myDatePicker2 addTarget:self
action:@selector (datePickerChanged2:)
forControlEvents:UIControlEventValueChanged];

    [self.myDatePicker3 addTarget:self
action:@selector (datePickerChanged3:)
forControlEvents:UIControlEventValueChanged];

}

-(void)prepareForSegue:(UIStoryboardSegue *)segue
sender:(id)sender
{

    if(sender != self.saveButton)return;

    if(self.fecha.text.length > 5)
    {

        NSString *SidLuz = @"id=";
        NSString *idLuz = @"1";
        NSString *Sfecha = @"&fecha=";
        NSString *fecha = self.fecha.text;
        NSString *ShoraIn = @"&horaIn=";
        NSString *horaIn= self.selectedDate.text;
        NSString *ShoraFin = @"&horaFin=";
        NSString *horaFin = self.selectedDate2.text;

        NSString *dir =
@"http://10.10.26.221:8080/DomotiveServerREST/webreso
urces/modoAutomatico?";
        NSString* strUrl = [NSString
stringWithFormat:@"%@%@%@%@%@%@%@%@",
dir,SidLuz,idLuz,Sfecha,fecha,ShoraIn,horaIn,ShoraFin
,horaFin];
        NSURL *url = [NSURL URLWithString:strUrl];
        NSURLRequest *request = [NSURLRequest
requestWithURL:url];
        [NSURLConnection
sendAsynchronousRequest:request

queue:[NSOperationQueue mainQueue]

completionHandler:^(NSURLResponse *response,

NSData *data, NSError *connectionError)
         {
             if (data.length > 0 && connectionError
== nil)
             {
                 NSLog(@"paso");
            }

         }];
        self.toDoItem = [[ToDoItem alloc]init];
        self.toDoItem.itemName = fecha;
        self.toDoItem.horaIn = horaIn;
```

```
        self.toDoItem.horaFin = horaFin;

    }
    else
    {
        NSDateFormatter *dateFormatter =
[[NSDateFormatter alloc]init];
        [dateFormatter setDateFormat:@"dd/MM/yyyy"];
        NSString *strDate = [dateFormatter
stringFromDate:self.myDatePicker3.date];
        self.fecha.text = strDate;

        NSString *SidLuz = @"id=";
        NSString *idLuz = @"1";
        NSString *Sfecha = @"&fecha=";
        NSString *fecha = self.fecha.text;
        NSString *ShoraIn = @"&horaIn=";
        NSString *horaIn= self.selectedDate.text;
        NSString *ShoraFin = @"&horaFin=";
        NSString *horaFin = self.selectedDate2.text;


        NSString *dir =
@"http://10.10.26.221:8080/DomotiveServerREST/webreso
urces/modoAutomatico?";
        NSString* strUrl = [NSString
stringWithFormat:@"%@%@%@%@%@%@%@%@",
dir,SidLuz,idLuz,Sfecha,fecha,ShoraIn,horaIn,ShoraFin
,horaFin];
        NSURL *url = [NSURL URLWithString:strUrl];
        NSURLRequest *request = [NSURLRequest
requestWithURL:url];
        [NSURLConnection
sendAsynchronousRequest:request

queue:[NSOperationQueue mainQueue]

completionHandler:^(NSURLResponse *response,

NSData *data, NSError *connectionError)
         {
             if (data.length > 0 && connectionError
== nil)
             {
                 NSDictionary *greeting =
[NSJSONSerialization JSONObjectWithData:data

options:0

error:NULL];

             }

         }];
        self.toDoItem = [[ToDoItem alloc]init];
        self.toDoItem.itemName = self.fecha.text;
        self.toDoItem.horaIn =
self.selectedDate.text;
        self.toDoItem.horaFin =
self.selectedDate2.text;
    }
```

```
}

- (void) datePickerChanged: (UIDatePicker
*)datePicker{
    NSDateFormatter *dateFormatter =
[[NSDateFormatter alloc]init];
    [dateFormatter setDateFormat:@"HH:mm"];
    NSString *strDate = [dateFormatter
stringFromDate:datePicker.date];
    self.selectedDate.text = strDate;
}

- (void) datePickerChanged2: (UIDatePicker
*)datePicker{
    NSDateFormatter *dateFormatter2 =
[[NSDateFormatter alloc]init];
    [dateFormatter2 setDateFormat:@"HH:mm"];
    NSString *strDate2 = [dateFormatter2
stringFromDate:datePicker.date];
    self.selectedDate2.text = strDate2;
}

- (void) datePickerChanged3: (UIDatePicker
*)datePicker{
    NSDateFormatter *dateFormatter =
[[NSDateFormatter alloc]init];
    [dateFormatter setDateFormat:@"dd/MM/yyyy"];
    NSString *strDate = [dateFormatter
stringFromDate:datePicker.date];
    self.fecha.text = strDate;
}
```

Si el usuario desea eliminar una de las simulaciones ingresadas deberá deslizar el dedo sobre dicha simulación como se muestra en la siguiente figura, en inglés a este gesto se le conoce como *swipe.*

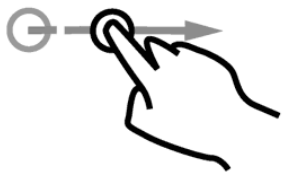


Fig.5.1.8 *Swipe Motion*

Fuente: http://answers.unity3d.com

Una vez ejecutado este movimiento, aparecerá un botón de color rojo con

el nombre *Delete* a la derecha de la simulación que queremos eliminar. Al pulsarlo, dicha simulación desaparecerá de la lista de simulaciones.

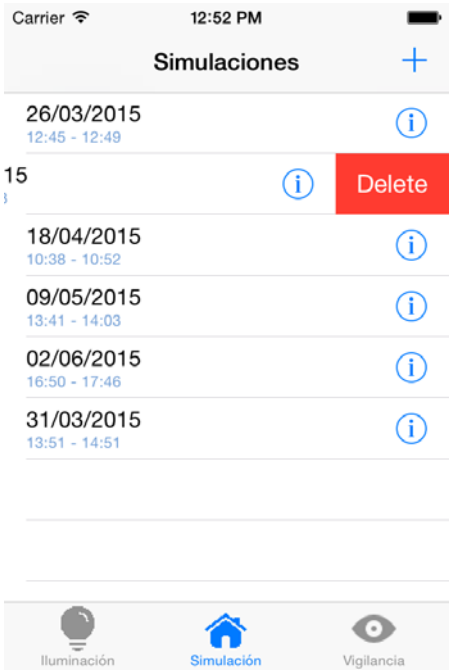


Fig.5.1.9 Eliminar Simulación

Fuente: Los Autores

Codificación:

Para la eliminación se debe modificar las opciones *removeObjectAtIndex* y *deleteRowsAtIndexPaths* del método de edición de la tabla que viene por defecto en el elemento *UITableViewController*, de esta manera el proceso de eliminación antes explicado se realiza automáticamente. Además se debe realizar la llamada al servicio *eliminarSimulacion* implementado en el servidor en el capítulo 3.4.3 para que dicha simulación sea eliminada de la base de datos, se debe enviar como parámetros el id de la lámpara, la fecha, la hora de inicio y la hora de finalización de la simulación:

```
- (void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle
)editingStyle forRowAtIndexPath:(NSIndexPath
*)indexPath {
    if (editingStyle ==
UITableViewCellEditingStyleDelete) {
        // Delete the row from the data source
        ToDoItem *toDoItem = [self.toDoItems
objectAtIndex:indexPath.row];

        NSString *SidLuz = @"id=";
        NSString *idLuz = @"1";
        NSString *Sfecha = @"&fecha=";
        NSString *fecha = toDoItem.itemName;
        NSString *ShoraIn = @"&horaIn=";
        NSString *horaIn= toDoItem.horaIn;
        NSString *ShoraFin = @"&horaFin=";
        NSString *horaFin = toDoItem.horaFin;

        NSString *dir =
@"http://10.10.26.221:8080/DomotiveServerREST/w
ebresources/eliminarSimulacion?";
        NSString* strUrl = [NSString
stringWithFormat:@"%@%@%@%@%@%@%@%@%@",
dir,SidLuz,idLuz,Sfecha,fecha,ShoraIn,horaIn,Sh
oraFin,horaFin];
        NSURL *url = [NSURL
URLWithString:strUrl];
        NSURLRequest *request = [NSURLRequest
requestWithURL:url];
        [NSURLConnection
sendAsynchronousRequest:request

queue:[NSOperationQueue mainQueue]

completionHandler:^(NSURLResponse *response,

NSData *data, NSError *connectionError)
         {
             if (data.length > 0 &&
connectionError == nil)
             {
                 NSDictionary *greeting =
[NSJSONSerialization JSONObjectWithData:data
options:0
error:NULL];
                 [self.toDoItems
removeObjectAtIndex:indexPath.row];
                 [tableView
deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationAutomat
ic];
             }
         }];
    } else if (editingStyle ==
UITableViewCellEditingStyleInsert) {
```

```
@end
```

Pantalla Sistema de Vigilancia

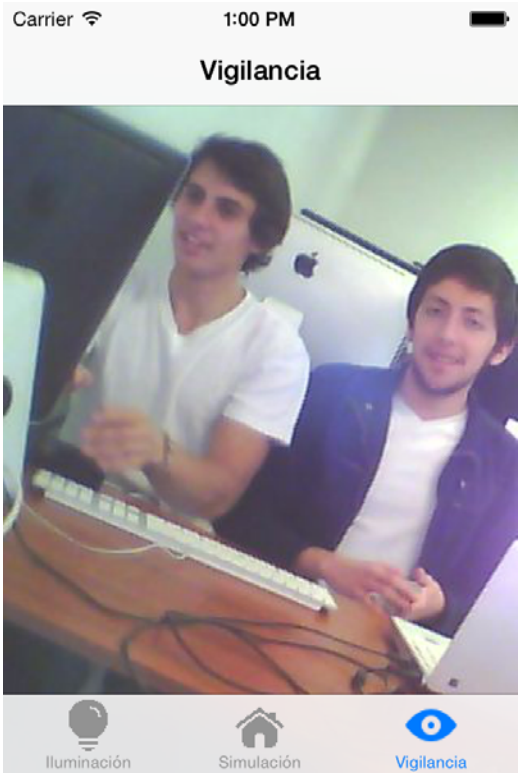


Fig.5.2.1 Pantalla Vigilancia

Fuente: Los Autores

Funcionamiento:

Al navegar hasta esta pantalla pulsando la opción *Vigilancia* en el menú inferior de la aplicación, se observará la imagen transmitida por la cámara IP utilizando un *web view*.

La resolución escogida para la imagen a transmitir es de 640 pixeles x 480 píxeles.

La dirección de la cámara IP a utilizar es la siguiente **http://192.168.1.101/image/jpeg.cgi**.

La extensión cgi hace referencia al estándar *Common Gateway Interface* que permite comunicar aplicaciones externas con servidores web, se utiliza este formato debido a que este tipo de archivos son ejecutados en tiempo real generando información dinámica a cada instante a diferencia de un archivo HTML plano que solo proporciona información de carácter estático.

Por esta razón, la extensión jpeg.cgi que se observa en la dirección IP de la cámara permite capturar la imagen que en ese preciso instante está observado el lente de la misma, la cual envía de manera dinámica una imagen actualizada en tiempo real con contenido diferente cada vez que este archivo es ejecutado. (w3, 2009) (Beth).

Codificación:

En el archivo .h correspondiente a esta pantalla se declara una variable del tipo *UIWebView* con el nombre *web*:

```
@interface webViewViewController : UIViewController{
IBOutlet UIWebView *web;
}

@property (nonatomic, retain) UIWebView *web;
@end
```

En el archivo .m correspondiente a esta pantalla se realiza la transmisión de la imagen de la cámara IP dentro de un *loop* que captura un *frame* cada 0.8 segundos, para lograr así la transmisión continua de los *frames* y producir el efecto de video en tiempo real.

```
@implementation webViewViewController
@synthesize web;
```

```
- (void)viewDidLoad {
    [web loadRequest:[NSURLRequest
requestWithURL:[NSURL
URLWithString:@"http://192.168.1.101/image/jpeg.cgi"]
]];


    [NSTimer scheduledTimerWithTimeInterval:0.8
target:self selector:@selector(WebViewLoad:)
userInfo:nil repeats:YES];
    [super viewDidLoad];
}


-(void)WebViewLoad:(NSTimer *)theTimer
{
    [web reload];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be
recreated.
}
```

Para observar la estructura final y el funcionamiento completo de la aplicación debemos acceder al archivo llamado *Main.storyboard* ubicado en el navegador de proyectos xcode, como se muestra a continuación:

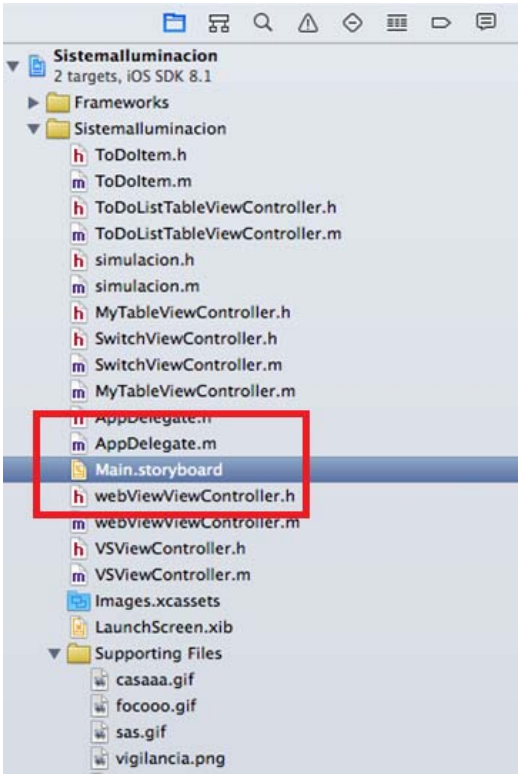


Fig.5.2.2 Storyboard

Fuente: Los Autores

El archivo Main.storyboard muestra las pantallas utilizadas en el proyecto, las transiciones que se producen entre ellas y el curso de navegación de la aplicación. (Apple.Inc, 2013)

Cada pantalla creada debe contener su propio *navigation controller,* este elemento contiene una vista que indica el elemento del menú o barra de navegación que deberá ser pulsado para observar la pantalla indicada. (Apple.Inc, 2013)

Los elementos que conectan las pantallas entre sí, que tienen el aspecto de una flecha indican el sentido en que se produce la relación entre una pantalla y la siguiente, a estas estructuras se les conoce como *segue*, la pantalla desde la que se realiza la transición es conocida como *source* y la pantalla a la que se llega luego de realizada la transición se denomina *destination*. (Apple.Inc, 2013)

El elemento *TabBarController* permite crear las opciones de menú de la barra de navegación, está debe ser un elemento del tipo *source* conectado hacia cada *navigation controller* utilizado. A continuación se observa el *storyboard* correspondiente a la aplicación desarrollada anteriormente:

Fig.5.2.3 Main.storyboard

Fuente: Los Autores

5.2.Aplicación para Android.

Para la codificación de la aplicación móvil Android se utilizará el entorno de desarrollo Eclipse y el lenguaje de programación Java.

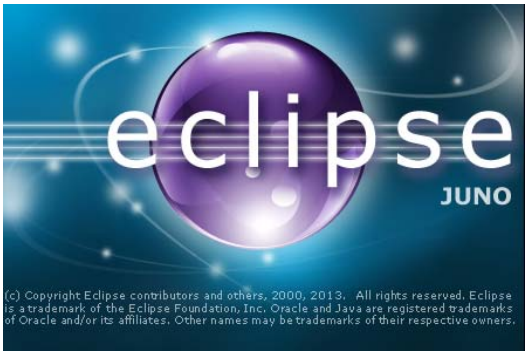


Fig.5.2.4. Entorno de Desarrollo Eclipse

Fuente: http://www.ics.uci.edu

Antes de observar la codificación de la aplicación, se explicará la tecnología utilizada para realizar la comunicación entre la aplicación cliente y el servidor, en este caso se utilizará la librería *ksoap2* como se muestra a continuación:

Primero se necesita importar los paquetes de la librería ksoap2 que se van a necesitar más adelante:

```
import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.PropertyInfo;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapPrimitive;
importorg.ksoap2.serialization.SoapSerializatio
nEnvelope;
import org.ksoap2.transport.HttpTransportSE;
```

Se necesita crear una tarea asíncrona para realizar la llamada al servidor en segundo plano, es decir, una operación en *background* que no interfiera en el hilo principal de la aplicación, para esto se utilizará la clase *AsyncTask* que permitirá publicar el resultado de la operación en *background* en el hilo principal conocido como *UIThread*.

A continuación se explican los 4 métodos de la clase *AsyncTask* que se pueden sobrescribir con su respectiva codificación:

```
AsyncCallWS task = new AsyncCallWS();
 task.execute();

private class AsyncCallWS extends
AsyncTask<String, Void, Void> {
```

onPreExecute( ): en este método se añaden todas las operaciones a realizar antes de ejecutar la llamada al servidor, se utiliza para inicializar las variables necesarias.

```
@Override
protected void onPreExecute()
{

}
```

doInBackground ( ): en este método se debe añadir la llamada al servidor, ya que todas las operaciones que se requieran ejecutar fuera del hilo principal de la aplicación se deben añadir aquí.

```
@Override
protected Void doInBackground(String...
params)
{
      return null;
}
```

onProgressUpdate ( ): este método se utiliza para informar al usuario el progreso en el que se encuentra la tarea ejecutada en segundo plano, normalmente se utiliza una barra de color verde que indica el porcentaje de la tarea completada.

```
@Override
protected void onProgressUpdate(Void...
values) {
}
```

```
}
```

onPostExecute (Result): este método muestra el resultado que ha sido retornado por el servidor luego de completada la tarea ejecutada en *background.*

```
@Override
protected void onPostExecute(Void result)
{

}
```

Ahora se explicará los pasos a seguir para la utilización de la librería ksoap2 que permitirá la comunicación con el servidor:

Se necesitarán variables globales que contengan los siguientes datos del servidor que se encuentran en el archivo WSDL mencionado en el capítulo 3.4.2:

private static String *NAMESPACE* = Aquí se debe colocar el valor correspondiente al atributo *targetNameSpace* del archivo WSDL.

private static String URL = Aquí se debe colocar la dirección URL del archivo WSDL mencionada en el capítulo 3.4.2.

private static String SOAP_ACTION = Aquí se debe colocar el valor correspondiente al atributo *operationName* del archivo WSDL.

Una vez obtenidos los datos necesarios para la comunicación se define la solicitud (*request*) que se va a enviar al servidor de la siguiente manera:

Se crea un objeto tipo *SoapObject:*

```
SoapObject request = new
SoapObject(NAMESPACE, operationName);
```

Se añade los parámetros a enviar como una propiedad del objeto creado anteriormente, se debe especificar su nombre, valor y tipo:

```
// Property which holds input parameters
PropertyInfo parametro1 = new
PropertyInfo();
// Set Name
parametro1.setName("param1");
// Set Value
parametro1.setValue(valorParam1);
// Set dataType
parametro1.setType(String.class);
// Add the property to request object
request.addProperty(parametro1);
```

Se guarda la solicitud (*request*) antes creada en un sobre (*envelope*) para que pueda ser enviada:

```
// Create envelope
SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(
SoapEnvelope.VER11);
// Set output SOAP object
envelope.setOutputSoapObject(request);
```

Se indica el medio de transporte a utilizar para enviar el sobre, en este caso se usará el protocolo HTTP:

```
// Create HTTP call object
HttpTransportSE androidHttpTransport =
new HttpTransportSE(URL);
```

Se invoca al servicio ofrecido por el servidor con el nombre del método correspondiente y el sobre creado anteriormente:

```
try {
      // Invoke web service
      androidHttpTransport.call
      (SOAP_ACTION + nombreMetodo,
      envelope);
}
```

Finalmente se recuperan los datos retornados por el servidor luego de la llamada a sus servicios:

```
try {
      // Get the response
      SoapPrimitive response =
      (SoapPrimitive)
      envelope.getResponse();

      // Assign it to resTxt variable
      static variable
      resTxt = response.toString();

} catch (Exception e) {
      //Print error
      e.printStackTrace();
      //Assign error message to resTxt
      resTxt = "Error occured";
}
//Return resTxt to calling object
return resTxt;
}
```

A continuación se mostrará cada pantalla que conforma la interfaz de la aplicación y se explicará su respectivo funcionamiento junto con el código correspondiente que permite consumir los servicios ofertados por el servidor:

Pantalla Módulo de Control (Sistema Iluminación)

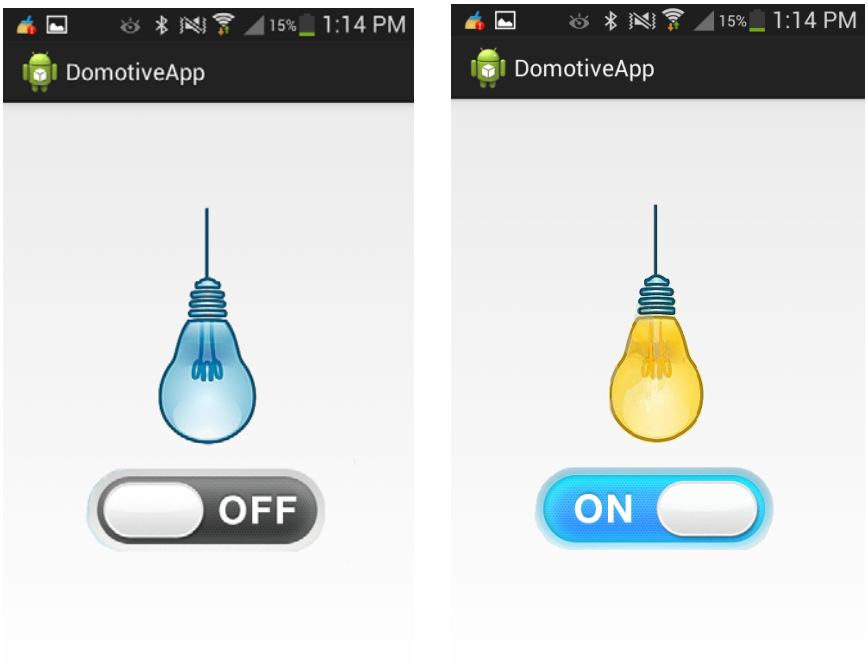


Fig.5.2.5 Módulo de Control

Fuente: Los Autores

Funcionamiento:

En el instante que el usuario abre la aplicación, se realiza una llamada al servidor para leer el estado en que se encuentra la lámpara, es decir, si está encendida o apagada, al recibir la respuesta del servidor, después de haber consultado la base de datos, la imagen de la lámpara de la interfaz se mostrará como encendida o apagada según el estado de respuesta del servidor.

Una vez que el estado en el que se encuentra la lámpara física y el estado que muestra la imagen de la interfaz estén correctamente sincronizados, el usuario podrá pulsar el botón para encender o apagar la lámpara.

Codificación:

Para la imagen de la lámpara encendida y apagada se declara un elemento del tipo *ImageView* con el nombre *image*:

```
image = (ImageView)
findViewById(R.id.imageViewFoco);
```

Para el botón que generará el evento de encender o apagar la lámpara se declaran 2 elementos tipo ImageView, una cuando el botón muestra el estado *ON* y otra para el estado *OFF*:

```
on = (ImageView)findViewById(R.id.imageViewOn);
off = (ImageView)findViewById(R.id.imageViewOff);
```

Al abrir la aplicación se neceista generar una tarea asíncrona inmediatamente para realizar la llamada al servicio *getCelEstado* implementado en el servidor en el capítulo 3.4.3 que realizará la sincronización entre la lámpara y la imagen, enviando como parámetro el *id* de la misma:

```
private class AsyncCallWS3 extends
AsyncTask<String, Void, Void> {
@Override
protected Void doInBackground(String... params)
{
      displayText =
      WebService.invokeEstado("1","getCelEstado
      ");
      return null;
}
```

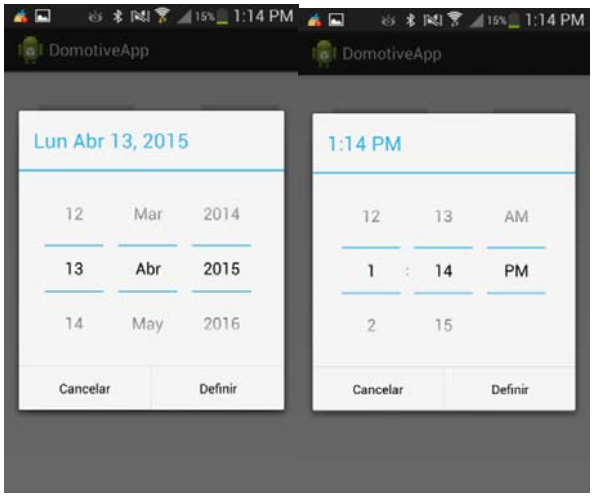
Además se agregará el método *onClick* que generará otra llamada a una tarea asíncrona, esta tarea realizará la llamada al servicio *celOnOff* implementado en el servidor en el capítulo 3.4.3 que realizará el encendido o pagado de la lámpara, si el usuario desea apagarla tiene que enviar como parámetro la cadena “off”, caso contrario si se desea encender la misma se enviará la cadena “on”:

```
public void onClick(View v) {
  AsyncCallWS task = new AsyncCallWS();
   task.execute();
}
});
private class AsyncCallWS extends
AsyncTask<String, Void, Void> {
@Override
protected Void doInBackground(String... params)
{
      displayText =
      WebService.invokeHelloWorldWS("on","1","c
      elOnOff");
      //prueba.setText(displayText);
      return null;
}
```

Pantalla Añadir Simulación (Módulo de Simulación de Presencia)

Fig.5.2.6 Pantalla Agregar Simulación

Fuente: Los Autores



`

Fig.5.2.7 Seleccionar Fecha y Hora

Fuente: Los Autores

Funcionamiento:

Al navegar hasta esta pantalla pulsando la opción *Simulación* en el menú de la aplicación, se observarán los elementos necesarios para que el usuario seleccione la fecha y las horas de inicio y fin para la simulación que desea ingresar.

Para agregar una nueva simulación se deben pulsar los botones “Hora Inicio”, “Hora Fin” y “Seleccionar Fecha” que se muestran en la interfaz, con lo que se desplegarán los elementos TimePicker y DatePicker. Una vez escogida la fecha y las horas de dichos elementos se deberá marcar el *checkbox Automático* para que la simulación sea ingresada al sistema.

DatePicker TimePicker

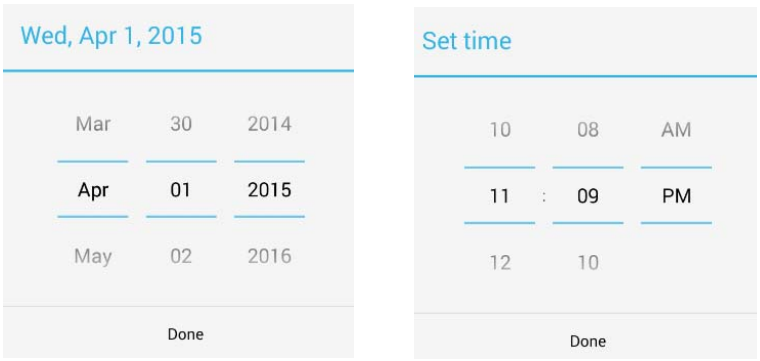


Fig.5.2.8 Date Picker / Time Picker

Fuente: Los Autores

Codificación:

Los elementos que permiten al usuario deslizar el dedo hacia arriba o hacia abajo para escoger una hora o fecha específica de manera rápida y eficiente tienen el nombre de TimePicker y DatePicker respectivamente:

Para el correcto funcionamiento del elemento DatePicker se debe enlazar con la clase *DatePickerFragment.java*:

```
public class DatePickerFragment extends
DialogFragment implements
DatePickerDialog.OnDateSetListener{

TextView txtDate;

public DatePickerFragment(TextView txtDate) {
super();
this.txtDate = txtDate;
}

@Override
    public Dialog
    onCreateDialog(BundlesavedInstanceState) {
    // Use the current date as the default date
    in the picker
    final Calendar c = Calendar.getInstance();
    int year = c.get(Calendar.YEAR);
    int month = c.get(Calendar.MONTH);
    int day = c.get(Calendar.DAY_OF_MONTH);

      // Create a new instance of
      DatePickerDialog and return it
```

```
            return new
            DatePickerDialog(getActivity(), this,
            year, month, day);
            }

           public void onDateSet(DatePicker view, int
           year, int month, int day) {
           txtDate.setText(new
           StringBuilder().append(day)
          .append("/").append(month +
          1).append("/").append(year)
          .append(" "));
          }
      }
```

Para el correcto funcionamiento del elemento TimePicker se debe enlazar con la clase *TimePickerFragment.java*:

```
public class TimePickerFragment extends
DialogFragment  implements
TimePickerDialog.OnTimeSetListener{

TextView txtTime;

public TimePickerFragment(TextView txtTime) {
      super();
      this.txtTime = txtTime;
}
```

```
@Override
public Dialog onCreateDialog(Bundle
savedInstanceState) {
      // Use the current time as the default values
      for the picker
      final Calendar c = Calendar.getInstance();
      int hour = c.get(Calendar.HOUR_OF_DAY);
      int minute = c.get(Calendar.MINUTE);
      // Create a new instance of TimePickerDialog
      and return it
      return new TimePickerDialog(getActivity(),
      this, hour, minute,
       DateFormat.is24HourFormat(getActivity()));
}

public void onTimeSet(TimePicker view, int hourOfDay,
int minute) {
       txtTime.setText(hourOfDay+":"+minute);
      }

}
```

Se necesita declarar las variables que guardaran los datos seleccionados en el *DatePicker* y *TimePicker*, además de la variable que controlará el evento del *checkbox*:

```
txtFecha = (TextView) findViewById(R.id.txtDate);
txtHoraInicio = (TextView)
findViewById(R.id.txtHoraInicio);
txtHoraFin = (TextView)
findViewById(R.id.txtHoraFin);
chk = (CheckBox) findViewById(R.id.checkBox1);
```

Para que la hora actual se muestre automáticamente en el *TimePicker* de la hora de incio, de la hora de finalización y en los *labels* colocados en la parte inferior de los mismos se utiliza el siguiente método:

```
public void showCurrentTimeOnView() {

      txtDisplayTime = (TextView)
      findViewById(R.id.txtTime);
      txtDisplayEndTime = (TextView)
      findViewById(R.id.txtEndTime);
      timePicker = (TimePicker)
      findViewById(R.id.timePicker1);
      timePicker.setVisibility(View.GONE);

      final Calendar c = Calendar.getInstance();
      hour = c.get(Calendar.HOUR_OF_DAY);
      minute = c.get(Calendar.MINUTE);



      // set current time into textview
      txtDisplayTime.setText(
      new StringBuilder().append(hour)
      .append(":").append(minute));

      txtDisplayEndTime.setText(
      new StringBuilder().append(hour)
       .append(":").append(minute));

      // set current time into timepicker
      timePicker.setCurrentHour(hour);
      timePicker.setCurrentMinute(minute);
}
```

Para que la fecha actual se muestre automáticamente en el *DatePicker* y en el *label* colocado en la parte inferior del mismo se utiliza el siguiente método:

```
public void showCurrentDateOnView() {
```

```
        txtDisplayDate = (TextView)
        findViewById(R.id.txtDate);
        datePicker = (DatePicker)
        findViewById(R.id.datePicker1);
        datePicker.setVisibility(View.GONE);

        final Calendar c = Calendar.getInstance();
        year = c.get(Calendar.YEAR);
        month = c.get(Calendar.MONTH);
        day = c.get(Calendar.DAY_OF_MONTH);

        // set current date into textview
        txtDisplayDate.setText(new StringBuilder()
        // Month is 0 based, just add 1
        .append(day).append("/").append(month +
        1).append("/")
        .append(year).append(" "));

        fecha= txtDisplayDate.getText().toString();

        // set current date into datepicker
        datePicker.init(year, month, day, null);

        //fecha =day + "/" + month + "/" + year;
    }
```

Además se agregará el método *onClick*, en el cual, una vez que se compruebe que el *checkbox Automático* ha sido seleccionado generará otra llamada a una tarea asíncrona, esta tarea realizará la llamada al servicio *insertarCelSimulacion* implementado en el servidor en el capítulo 3.4.3, se tiene que enviar como parámetros la fecha, la hora de inicio y la hora de finalización correspondiente a la simulación:

```
@Override
public void onClick(View v) {
    if (((CheckBox) v).isChecked()) {
          AsyncCallWS task = new AsyncCallWS();
           task.execute();
    }
}

private class AsyncCallWS extends AsyncTask<String,
Void, Void> {

@Override
protected Void doInBackground(String... params) {

    fecha = fechaCadena();
    horaInicio = horaInicio();
    horaFin = horaFin();
    displayText=WebService.
```

```
        invokeSimulacion("1",fecha,horaInicio,horaFin,"
        insertarCelSimulacion");

        return null;
    }
```

Pantalla Sistema de Vigilancia

Fig.5.2.9 Pantalla Vigilancia

Fuente: Los Autores

Funcionamiento:

Al navegar hasta esta pantalla pulsando la opción *Vigilancia* en el menú de la aplicación, se observará la imagen transmitida por la cámara IP utilizando un *web view*.

La resolución escogida para la imagen a transmitir es de 640 pixeles x 480 píxeles.

La dirección de la cámara IP a utilizar es la siguiente **http://192.168.1.101/image/jpeg.cgi**.

La extensión cgi hace referencia al estándar *Common Gateway Interface* que permite comunicar aplicaciones externas con servidores web, se utiliza este formato debido a que este tipo de archivos son ejecutados en tiempo real generando información dinámica a cada instante a diferencia de un archivo HTML plano que solo proporciona información de carácter estático. Por esta razón, la extensión jpeg.cgi que se observa en la dirección IP de la cámara permite capturar la imagen que en ese preciso instante está observado el lente de la misma, la cual envía de manera dinámica una imagen actualizada en tiempo real con contenido diferente cada vez que este archivo es ejecutado. (w3, 2009) (Beth).

Codificación:

Para la transmisión de la imagen se necesita una clase que recargue el *web view* cada cierto tiempo, en donde la captura de los *frames* se realice dentro de un *loop* cada 900 milisegundos, para lograr así la transmisión continua de los *frames* y producir el efecto de video en tiempo real, los parámetros que necesita el método constructor de esta clase son el contexto de la actividad correspondiente, el número de segundos para el *loop* y la instancia de un *web view*:

```
public class ReloadWebView extends TimerTask {

      Activity context;
      Timer timer;
      WebView wv;

      public ReloadWebView(Activity context, int
      seconds, WebView wv) {
      this.context = context;
      this.wv = wv;

      timer = new Timer();
      /* execute the first task after seconds */
      //timer.schedule(this,
      //  seconds * 1000,  // initial delay
      //  seconds * 1000); // subsequent rate
```

```
        //if you want to execute the first task
        immediatly */

         timer.schedule(this,
         0,               // initial delay null
         seconds * 300); // subsequent rate

         }

         @Override
         public void run() {
               if(context == null ||
               context.isFinishing()) {
               // Activity killed
               this.cancel();

               return;
         }

         context.runOnUiThread(new Runnable() {
         @Override
         public void run() {
             wv.reload();
         }
         });
         }
}
```

A continuación se asigna la dirección IP de la cámara a un elemento *WebView*, y se llama a la clase *ReloadWebView* creada anteriormente para conseguir la transmisión del video en tiempo real.

```
public class Vigilancia extends Activity {

      final static String RTSP_URL =
      http://192.168.1.101/image/jpeg.cgi

      @Override
      protected void onCreate(Bundle
      savedInstanceState) {

      super.onCreate(savedInstanceState);
      this.setContentView(R.layout.vigilancia);

      WebView myWebView = (WebView)
      this.findViewById(R.id.webView1);
```

```
                myWebView.loadUrl(RTSP_URL);

                ReloadWebView refresh = new
                ReloadWebView(this, 3, myWebView);

                }
        }
```

Pantalla AcercaDe

Fig.5.3.1 Pantalla Acerca De

Fuente: Los Autores

Funcionamiento:

Al navegar hasta esta pantalla pulsando la opción *Acerca de* en el menú de la aplicación, se observará la información de contacto de los desarrolladores de la aplicación.

Codificación:

Para mostrar los datos de contacto que se observan en la interfaz se utilizará un elemento del tipo *StringBuffer* que permite concatenar varias variables tipo *String* para ser mostrados en un solo *TextView:*

```
public class AcercaDe extends Activity {
      public TextView outputText;
      public String nombre2 = "Pedro Cárdenas";
      public String nombre1 = "Pedro Corral";
      public String telefono1 = "0984784972";
      public String telefono2 = "0983820444";
      public String email1 =
      "pedro9445@hotmail.com";
      public String email2 =
      "perico_corral09@hotmail.com";

      StringBuffer output = new StringBuffer();

      @Override
      public void onCreate(Bundle
      savedInstanceState) {

      super.onCreate(savedInstanceState);

      setContentView(R.layout.acercade);
      outputText = (TextView)
      findViewById(R.id.textView1);
      output.append("\n");

      output.append("\nNombre: ");
      output.append(nombre1);
      output.append("\nTeléfono: ");
      output.append(telefono1);
      output.append("\nEmail: ");
      output.append(email1);
      output.append("\nNombre: ");
      output.append(nombre2);
      output.append("\nTeléfono: ");
      output.append(telefono2);
      output.append("\nEmail: ");
      output.append(email2);
      output.append("\n");
      outputText.setText(output);
      }
```

Finalmente, para que la aplicación sea compilada y ejecutada correctamente se debe agregar etiquetas y permisos al archivo *AndroidManifest.xml* que se encuentra en la raíz del proyecto:

Etiquetas, se debe agregar una etiqueta *activity* para cada pantalla que creamos a excepción de la pantalla del módulo de control debido a que esta es creada automáticamente por ser la primera en mostrarse al abrir la aplicación:

```
<activity
      android:name=".Vigilancia"
      android:label="@string/app_name" >
</activity>


<activity
      android:name=".AcercaDe"
      android:label="@string/app_name" >
</activity>


<activity
      android:name=".SimulacionPresencia"
      android:label="@string/app_name" >
</activity>
```

Permisos: se necesita que la aplicación tenga permiso de acceder y conectarse a internet para que pueda consumir los servicios ofertados por el servidor, para eso se agrega la siguiente líneas de código:

```
<uses-permission
android:name="android.permission.INTERNET" />
```

A continuación se muestra el contenido completo que debe tener el archivo *AndroidManifest.xml* para verificar como debe verse al finalizar la aplicación:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.domotiveapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />

    <uses-permission
android:name="android.permission.INTERNET" />


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>


        <activity
            android:name=".Vigilancia"
            android:label="@string/app_name" >
        </activity>


            <activity
            android:name=".AcercaDe"
            android:label="@string/app_name" >
        </activity>


            <activity
            android:name=".SimulacionPresencia"
            android:label="@string/app_name" >
        </activity>



    </application>

</manifest>
```

Conclusiones del capítulo

Una vez implementadas y codificadas las aplicaciones móviles sobre las plataformas de desarrollo mencionadas, se observó que la sintáxis, la estructura de los proyectos, la configuración y la tecnología utilizada en cada plataforma para consumir servicios web externos a la aplicación, se realizan de manera completamente diferente en iOS y en Android. En el siguiente capítulo se realizarán varias pruebas del funcionamiento de todo el sistema, en donde interactuén todos sus módulos sobre un entorno real.

6. CAPÍTULO 6: Aplicación de la solución

Introducción

En este capítulo se explicará cómo se va a realizar el acoplamiento del sistema domótico sobre un entorno real, además se realizarán pruebas sobre dicho sistema para verificar el funcionamiento de cada uno de sus módulos y la interacción entre ellos, de esta manera se observará si la solución que se ha desarrollado en los capítulos anteriores satisface las necesidades del mismo.

6.1.Acoplamiento del prototipo.

El acoplamiento de la solución correspondiente al sistema domótico implementado en los capítulos anteriores se realizará en la habitación de un hogar, dentro de la cual se conectará el circuito controlador a una de sus lámparas junto con el ordenador que actuará como servidor, además se instalará la cámara IP dentro de la misma habitación.

Los elementos a utilizar para dicho acoplamiento se muestran a continuación:

Lámpara:

Fig.6.1.1. Lámpara

Fuente: Los Autores

Router:

Fig.6.1.2 Router

Fuente: Los Autores

Circuito Controlador / Arduino:

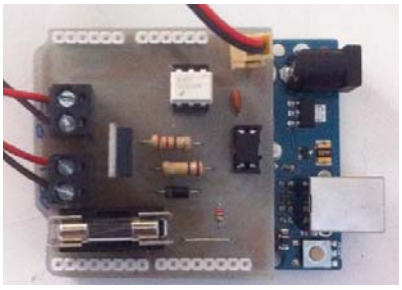


Fig.6.1.3 Circuito Controlador

Fuente: Los Autores

Ordenador (Servidor)

Fig.6.1.4 Ordenados (Servidor**)**

Fuente: Los Autores

Cámara IP:

Fig.6.1.5 Cámara IP

Fuente: Los Autores

Ahora se describirá paso a paso el proceso de instalación del sistema domótico:

1. Conectar el circuito controlador a la lámpara del hogar como se muestra en la siguiente figura.

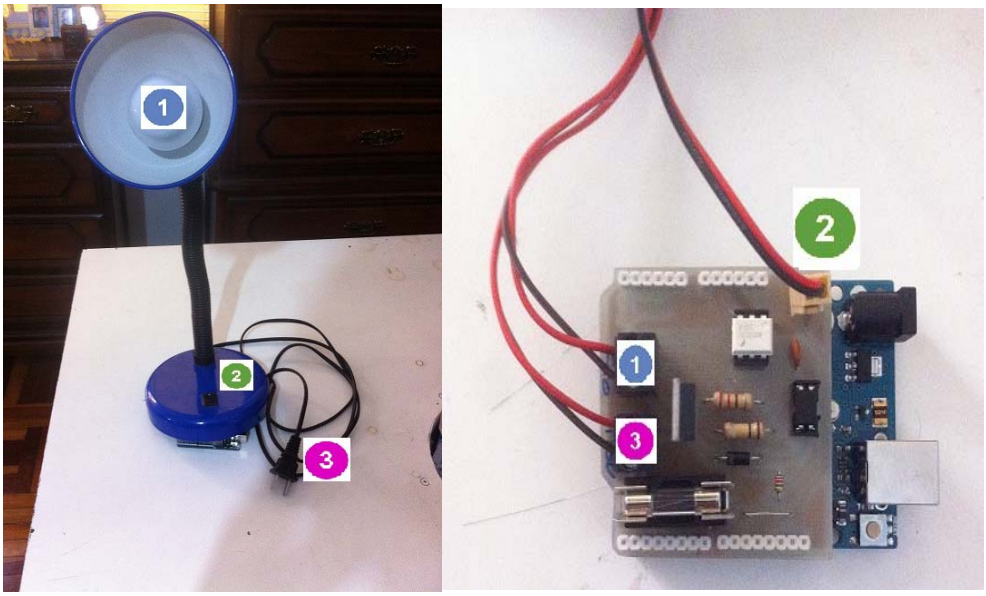


Fig.6.1.6. Conexión Lámpara/Circuito

Fuente: Los Autores

1: la bornera etiquetada con el número 1 de color azúl debe ser conectada directamente al foco de la lámpara.

2: la bornera etiquetada con el número 2 de color verde debe ser conectada al switch de la lámpara.

3: la bornera etiquetada con el número 3 de color rosado debe ser conectada a los 110 v.

2. Conectar el un extremo del cable USB al puerto USB de la placa Arduino.

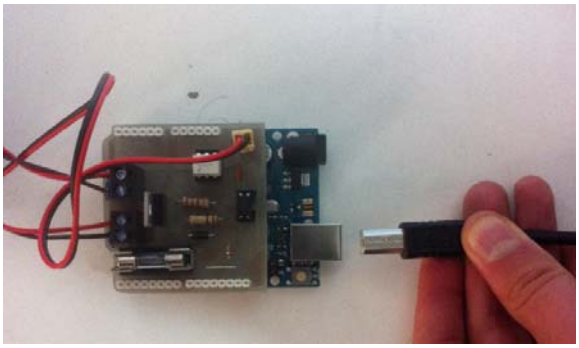


Fig.6.1.7. Cable USB Arduino

Fuente: Los Autores

3. Conectar el otro extremo del cable USB a un puerto USB del ordenador (Servidor).

Fig.6.1.8. Cable USB Ordenador

Fuente: Los Autores

4. Conectar el cable de alimentación de la lámpara a la línea eléctrica (110 v).

Fig.6.1.9. Lámpara

Fuente: Los Autores

5. Conectar el cable de alimentación del *router*.

Fig.6.2.1 Router

Fuente: Los Autores

6. Conectar el un extremo del cable de alimentación de la cámara a la parte posterior de la misma y el otro extremo a 110v.

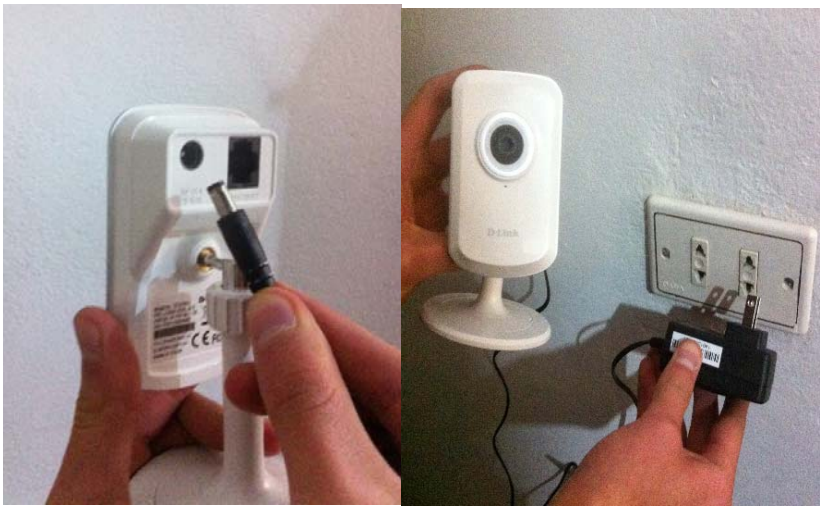


Fig.6.2.2. Conexión Cámara IP

Fuente: Los Autores

7. Conectar el un extremo de un cable de red al puerto Ethernet de la cámara IP y el otro extremo a un puerto LAN del router.

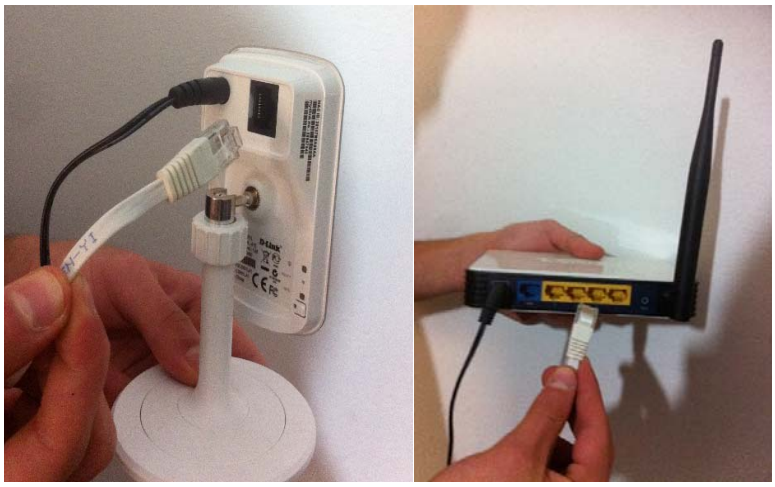


Fig.6.2.3. Conexión Cámara IP/Router

Fuente: Los Autores

Una vez que todos los elementos del sistema domótico han sido acoplados como se observa en la siguiente figura, se realizarán las pruebas descritas en el siguiente subcapítulo para verificar su funcionamiento:

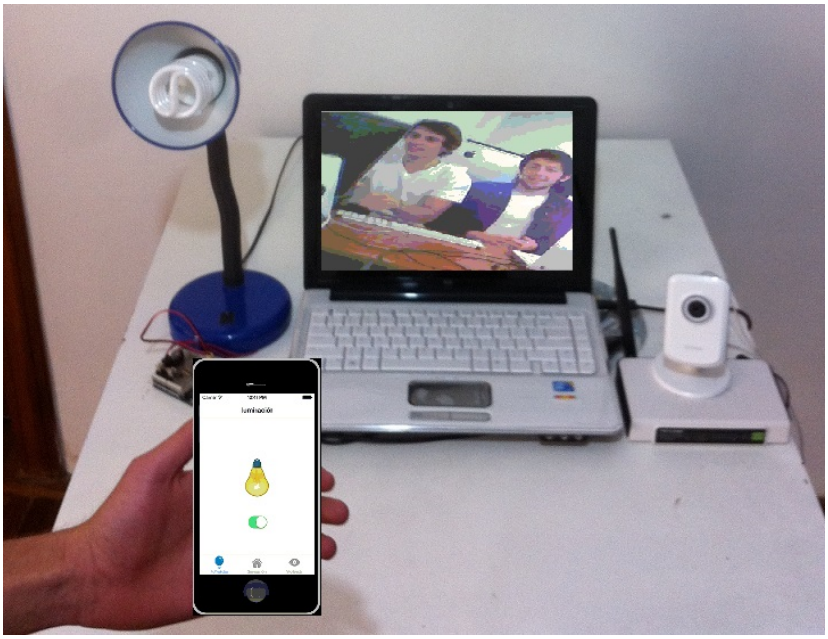


Fig.6.2.4. Sistema Domótico Acoplado

Fuente: Los Autores

6.2.Funcionamiento y Pruebas.

Para verificar el correcto funcionamiento del sistema domótico se realizaron las siguientes pruebas sobre cada uno de sus módulos:

Módulo de control: se encendió la lámpara desde la aplicación iOS y en seguida se apagó la misma desde la aplicación Android, se repitió este proceso 10 veces, apagando y encendiendo la lámpara desde ambas aplicaciones.

Módulo Simulación de Presencia: las pruebas realizadas sobre este módulo se realizaron en 3 fases:

1. Primera fase: se ingresó una simulación con una duración de 1 minuto, la cual se inició y finalizó exactamente a las horas ingresadas.

2. Segunda fase: se ingresaron 10 simulaciones, en un solo día, con una duración entre 1 y 2 minutos como se muestra a continuación:

Fecha	Hora Inicio	Hora Fin	Se ejecutó exitosamente?
01/04/2105	19:35	19:36	SI
01/04/2105	19:39	19:41	SI
01/04/2105	19:43	19:45	SI
01/04/2105	19:48	19:49	SI
01/04/2105	19:52	19:54	SI
01/04/2105	19:55	19:57	SI
01/04/2105	20:00	20:01	SI
01/04/2105	20:05	20:07	SI
01/04/2105	20:10	20:12	SI
01/04/2105	20:20	20:22	SI

Fig.6.2.5 Resultados Segunda Fase

Fuente: Los Autores

3. Tercera fase: se ingresaron 10 simulaciones, en días distintos, con diferentes tiempos de duración como se muestra a continuación:

Fecha	Hora Inicio	Hora Fin	Se ejecutó exitosamente?
01/04/2105	23:35	23:40	SI
02/04/2105	00:05	00:30	SI
03/04/2105	10:45	11:00	SI
03/04/2105	11:02	12:00	SI
03/04/2105	16:00	16:30	SI
03/04/2105	21:55	22:00	SI
04/04/2105	11:00	11:02	SI
04/04/2105	21:00	21:17	SI
05/04/2105	13:25	14:12	SI
05/04/2105	21:21	22:22	SI

Fig.6.2.6 Resultados Tercera Fase

Fuente: Los Autores

Módulo de Bitácora: se verificó exitosamente que cada una de las simulaciones ingresadas en las pruebas realizadas anteriormente fue almacenada correctamente en la base de datos (bitácora) además de ser notificada al usuario mediante correo electrónico.

Sistema de vigilancia: se verificó que la imagen de la cámara IP mostrada en el dispositivo móvil es de una calidad muy aceptable, además la velocidad de la transmisión a pesar de no ser exactamente en tiempo real, tiene un retraso mínimo que no interfiere en su visualización.

Conclusiones del capítulo

Una vez que el sistema dómotico fue implementado, acoplado sobre un entorno real (hogar) y su funcionamiento fue verificado cumpliendo exitosamente cada una de las pruebas realizadas, se puede decir que la solución implementada a lo largo de este trabajo satisface los requerimientos del sistema domótico establecidos al inicio del mismo.

7. Conclusiones Generales

Una vez finalizado el trabajo de titulación se obtuvieron las siguientes conclusiones:

La domótica apoya y brinda una solución para minimizar la inseguridad a la que se encuentra expuesto un hogar, ya que el sistema de iluminación permite simular la presencia de personas en casos en que nadie se encuentre dentro del mismo, además, el sistema de vigilancia permite que el usuario pueda observar en tiempo real lo que está sucediendo dentro de su hogar con la posibilidad de reaccionar inmediatamente en caso de ver alguna situación de peligro. Incluso en ocasiones en que se encuentre fuera de su hogar y no esté observando el interior del mismo, el usuario recibirá una notificación en su correo electrónico, la cual indicará que una lámpara de su hogar ha sido encendida.

La domótica también ofrece mayor confortabilidad al usuario, ya que el sistema de iluminación permite que el usuario encienda o apague las lámparas de su hogar en el momento que él desee, además puede encender o apagar las lámparas de cualquier habitación sin estar presente en ella, por ejemplo, puede apagar la lámpara del garaje mientras está descansando en su dormitorio.

El usuario podrá obtener mayor seguridad y confortabilidad en la palma de su mano, con solo realizar un clic, ya que podrá acceder tanto al sistema de iluminación como al sistema de vigilancia desde cualquier dispositivo móvil con conexión a internet, siempre y cuando éstos tengan un sistema operativo con tecnología iOS y/o Android.

La implementación de dómotica dentro de un hogar cada vez puede ser más amplia gracias a las posibilidades que permite la integración de diferentes ramas tecnológicas como la electrónica, las telecomunicaciones, la programación web y el desarrollo móvil dentro de un solo sistema.

Los resultados esperados fueron completados exitosamente, ya que cada capa del sistema domótico implementado se acopló a la siguiente de manera correcta, es decir, se logró realizar la comunicación entre la lámpara y el circuito controlador, a su vez el circuito controlador se comunicó con la placa Arduino, la cual permtió la conexión con el ordenador (servidor), el servidor logró enlazarse con la base de datos además de ser una instancia para poder conectarse al middleware y finalmente el middleware permitió que las aplicaciones móviles consuman los servicios ofertados por el servidor, de esta manera el sistema dómotico implementado funcionó correctamente y se puede decir que la solución que se desarrolló a lo largo de este trabajo satisface los requerimientos establecidos al inicio del mismo.

8. Recomendaciones

Los resultados obtenidos durante la investigación y realización de este trabajo de titulación, se espera que sirvan como inicio para investigaciones posteriores que pueden abordar temas como:

Implementar una VPN al sistema domótico para que pueda ser controlado remotamente desde cualquier parte del mundo.

Agregar la posibilidad de controlar todas las lámparas de un hogar, implementado un circuito en cada lámpara, así como tener acceso a varias cámaras IP colocadas en distintas partes del hogar.

Realizar la comunicación con el circuito controlador de manera inalámbrica sin necesidad de utilizar el cable USB, de esta manera se evitará realizar el cableado desde el punto de instalación de cada circuito hasta el lugar donde se encuentra el servidor.

Ampliar las funcionalidades del sistema domótico utilizando sensores para controlar la temperatura, la dimerización de la iluminación, las cortinas/persianas, puertas, televisores, etc.

Realizar un prototipo que demuestre la nueva tecnología conocida como el Internet de las cosas, en donde no solo las lámparas, sino varios artefactos electrónicos del hogar se comuniquen entre sí.

9. Bibliografía

(s.f.). Obtenido de Puerto USB: http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/alvarez_v_ce/apendiceA.pdf

Apple Inc. (18 de Julio de 2014). *iOS Dev Center*. Obtenido de https://developer.apple.com/devcenter/ios/index.action

alvarez. (13 de 8 de 2004). Obtenido de http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/alvarez_v_ce/apendiceA.pdf

Anton, J. (s.f.). *Domótica e Inmótica.* Espana: Automation Products S.A. Obtenido de http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf

Apple.Inc. (2013). *Your Second iOS app: Storyboard.*

Arduino. (2015). *Arduino*. Obtenido de http://www.arduino.cc/

Báez, M. (s.f.). *Introducción a Android.* E.M.E Editorial.

Berson, A. (2007). *Client/Server Architecture.* McGraw Hill.

Beth, A. (s.f.). *Common Gateway Interface CGI.* Texas.

CadSoft Computer. (2011). *CS Eagle.* Obtenido de http://www.cadsoftusa.com/eagle-pcb-design-software/schematic-editor/

Christensen, E. (15 de Marzo de 2001). *Web Services Description Language (WSDL) 1.1*. Obtenido de http://www.w3.org/TR/wsdl

Dept. Ciencia de la computación e IA. (2012-2013). Introducción a Xcode y Objective-C.

Dlink. (04 de 01 de 2013). *D-Link DCS-931L User Manual.* Obtenido de ftp://ftp.dlinkla.com/pub/DCS-931L/DCS-931L_A1_Manual_v1.10(WW).pdf

D-Link Corporation/D-Link Systems, Inc. (2014). Obtenido de Dlink Building Networks for People: http://www.dlinkla.com/dcs-931l

EasyTutz. (2015). *EasyTutz*. Obtenido de http://www.eazytutz.com/android/android-architecture/

FAIRCHILD. (2014). *1n4007.* Obtenido de http://arduino.cc/documents/datasheets/1n4007.pdf

Hostinger Colombia. (2012). *Hosting Gratuito con PHP Y mySQL.* Obtenido de http://www.hostinger.co

Ibertrónica. (2014). Obtenido de Conectores USB:Los diferentes tipos: http://www.ibertronica.es/blog/tutoriales/conectores-usb-tipos/

Inc, M. T. (2009). *PIC18F2455/2550/4455/4550 DataSheet.*

Juliá, A. B. (s.f.). *Automation Products S.A.* Obtenido de http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf

Kioskea. (s.f.). *USB Bus de serie universal.* Obtenido de Kioskea.net: http://es.kioskea.net/contents/407-usb-bus-de-serie-universal

Lopéz, J. (s.f.). Arquitectura Cliente Servidor. Obtenido de http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r99011.PDF

Madero, G. (2012 de 04 de 29). *Qué es IOS*. Obtenido de http://www.osupiita.com/index.php/proyectos/ios/que-es-ios

Manchado, D. S. (2010). *Estudio del Servidor de Aplicaciones Glassfish y de las aplicaciones J2EE.* Obtenido de http://www.recercat.net/bitstream/handle/2072/206748/SerraManchadoDavidR-ETISa2009-10.pdf?sequence=1

Marset, R. (2006). *Modelado, Diseño e Implenetación de Servicios Web.* Obtenido de http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf

Pérochon, S. (2012). *Android Guía de desarrollo de aplicaciones para Smartphones y Tabletas.* Barcelona: Ediciones ENI.

RNDS. (s.f.). Calidad de Imagen y Transmición a Bajo Costo: Cámaras IP. 6.

Rodriguez, S. I. (s.f.). Obtenido de Universal Serial Bus: http://www.iuma.ulpgc.es/~avega/int_equipos/trab9899/usb_1/index.html

Rouse, M. (12 de 2014). *SOAP (Simple Object Access Protocol)*. Obtenido de http://searchsoa.techtarget.com/definition/SOAP

SHARP. (s.f.). Obtenido de Datasheet PIC817X: http://www.sharpsme.com/download/pc817x-eJpdf

Sommerville, I. (2005). *Ingeniería del Software* (7 ed.). Madrid: Pearson Educación.

Tarkoma, S. (2009). *Mobile Middleware.* John Wiley.

Texas Instrument. (2015). Obtenido de Datasheet Catalog: http://www.datasheetcatalog.com/info_redirect/datasheet_pdf/texas-instruments/MOC3020_to_MOC3023.pdf.shtml

Torrente, Ó. (2013). *ARDUINO. Curso práctico de formación.* Madrid, España: RC Libros.

VISHAY. (2012). Obtenido de Datasheet 14n001 thru 1n4007: http://www.vishay.com/docs/88503/1n4001.pdf

w3. (2009). *http://www.w3.org/CGI/*.

Ycezalaya, J. P. (s.f.). Gerente Comercial NetPoint.

Báez, Borrego, Cordero, Cruz, González, Hernández, Palomero, Rodríguez de Llera, DanieSanz, Saucedo, Torralbo, Zapata. Introducción a Android. Obtenido de: http://pendientedemigracion.ucm.es/info/tecnomovil/documentos/android.pdf

WordPress. (2015). Definición de USB. Obtenido de: http://definicion.de/usb/

Definición ABC. (2015). Definición de USB. Obtenido de:
http://www.definicionabc.com/acerca-de

Informaticamoderna.com. El puerto USB 1/2.0 y 3.0. Obtenido de:

http://www.informaticamoderna.com/El_puerto_USB.htm

redusers.com.2005.Introducción a Proteus. Obtenido en:
http://img.redusers.com/imagenes/ebook/lpcu239/notagratis.pdf

Arduino. (2015). Arduino UNO. Obtenido en:
http://arduino.cc/en/Main/ArduinoBoardUno

10. ANEXOS

Anexo A: Diseño del Trabajo de Titulación

Universidad del Azuay

Facultad de Ciencias de la Administración

Escuela de Ingeniería de Sistemas y Telemática.

Trabajo de Titulación: Prototipo de un Sistema Domótico para controlar iluminación y cámaras de seguridad mediante dispositivos móviles.

Trabajo de Titulación, previo a la obtención del título de Ingeniero de Sistemas y Telemática

Autores:

Pedro Andrés Cárdenas Vásquez.
Pedro Sebastián Corral Jaramillo.

Director Sugerido:

Ing. Esteban Crespo Martínez

Cuenca – Ecuador

2014

1. **Datos Generales**

1.1. Nombre del Estudiante:

Nombre: Cárdenas Vásquez Pedro Andrés

Código: ua049498

Contacto: 072817875, 0984784972, pedro9445@hotmail.com

Nombre: Corral Jaramillo Pedro Sebastián

Código: ua049217

Contacto: 072880583, 0983820444, perico_corral09@hotmail.com

1.2. Director Sugerido: Crespo Martínez Paúl Esteban. Ingeniero.

1.2.1. Contacto: 4092109, 0996804562, ecrespo@uazuay.edu.ec.

1.3. Co-director sugerido: Ing. Diego Chacón.

1.3.1. Contacto: dpchacont@yahoo.es

1.4. Asesor Metodológico: Ing. Francisco Salgado.

1.5. Tribunal Designado.

1.6. Aprobación: Fecha de Junta Académica y fecha de consejo de facultad.

1.7. Línea de Investigación de la Carrera

1.7.1. UNESCO: Código 1203.99: Telemática

1.7.2. Tipo de Trabajo:

a) Está dentro de Ciencia de Los Ordenadores, cuya subdisciplina corresponde a la rama de la Telemática, dentro del campo de desarrollo de la comunicación entre distintos dispositivos computacionales y electrónicos.

b) Investigación Formativa.

1.8. Área de Estudio:

Las áreas involucradas dentro del proyecto son: programación, electrónica digital, microcontroladores, ingeniería de software y domótica.

1.9. Título Propuesto:

Prototipo de un Sistema Domótico para controlar iluminación y cámaras de seguridad mediante dispositivos móviles.

1.10. Subtítulo:

Sistema de Seguridad y Automatización de luces de una vivienda.

1.11. Estado del Proyecto.

Al momento el proyecto tiene desarrollado un sistema de iluminación manejado a través de dispositivos móviles Android. Este sistema, no ha sido puesto a prueba de forma rigurosa, por lo que se pretende con este proyecto de grado realizar todos los cambios y mejoras en el mismo para su óptimo funcionamiento.

2. CONTENIDO

2.1. Motivación:

La falta de comodidad, confort y seguridad en los hogares motivó a mejorar esta situación integrando distintas tecnologías móviles y de telecomunicaciones actuales, para de esta manera acoplar el hogar al mundo tecnológico de hoy.

2.2. Problemática:

La delincuencia e inseguridad ha obligado a la población cuencana a buscar métodos de protección para sus familias y hogares, tales como sistemas de alarma, cercados eléctricos, aseguradoras, entre otros. Pero lamentablemente estos sistemas no cumplen con la eficacia necesaria para garantizar seguridad debido a que no existen soluciones integradas, es decir, no se pueden manejar los distintos sistemas de iluminación y vigilancia desde un solo dispositivo.

Además las personas prefieren facilidad y comodidad en el desarrollo de sus actividades diarias, es por ello que gracias a la evolución de la tecnología se podría automatizar varios de los procesos enunciados.

Actualmente existen tecnologías domóticas como el X-10 Home System, que permite el control del hogar a través de su línea eléctrica, siendo su limitante el funcionamiento dependiente de comandos y parámetros preestablecidos. Es por ello que el control mediante un dispositivo móvil facilitará el acceso a las cámaras IP y por ende ahorrará tiempo al usuario final en la interacción con la vivienda.

En la ciudad de Cuenca se ha podido observar la falencia en soluciones innovadoras en cuanto a tecnología domótica. Esto se debe, a que no existe un gran número de empresas preocupadas por el confort dentro de un hogar, lo que ha creado una brecha tecnológica en cuanto a actualización y desarrollo de sistemas para el bienestar y comodidad familiar.

.

2.3. Pregunta de Investigación:

¿La domótica apoya a la solución de inseguridad y falta de confortabilidad de un domicilio dentro la ciudad de Cuenca?

2.4. Resumen:

En este proyecto se expone una solución a la inseguridad y a la falta de confortabilidad de los hogares de la ciudad de Cuenca, a través de un sistema

domótico que permitirá el control de cámaras de seguridad e iluminación mediante el uso de dispositivos móviles (celulares y tablets), sobre los sistemas operativos más utilizados, iOS y Android. Al finalizar este trabajo se probará y demostrará el funcionamiento del sistema sobre el prototipo construido por medio de las aplicaciones desarrolladas.

2.5. Estado del Arte y marco teórico:

Concepto de domótica

El término domótica proviene de dos palabras:

domus que significa casa (en latín).

tica del término informática.

La domótica es el conjunto de sistemas que automatizan las diferentes instalaciones de la vivienda, permitiendo la integración y aplicación de las nuevas tecnologías informáticas que simplificarán tareas y producirán confort. Incluye principalmente el uso de electricidad, dispositivos electrónicos, sistemas informáticos y diferentes protocolos de comunicación, incorporando la tecnología móvil e internet.

"Huidobro J.M. y Millán R. (2004) recogen que el origen de la Domótica se remonta a los años setenta, cuando en Estados Unidos aparecieron los primeros dispositivos de automatización de edificios basados en la aún hoy exitosa tecnología X-10."(Domótica: Un Enfoque Sociotécnico, pag. 15)

"Estas incursiones primerizas se alternaron con la llegada de nuevos sistemas de calefacción y climatización orientados al ahorro de energía, en clara sintonía con las crisis del petróleo. Los primeros equipos comerciales se limitaban a la colocación de sensores y termostatos que regulaban la temperatura ambiente. La disponibilidad y proliferación de la electrónica de bajo coste favoreció la

expansión de este tipo de sistemas, despertando así el interés de la comunidad internacional por la búsqueda de la casa ideal. Los ensayos con electrodomésticos avanzados y otros dispositivos automáticos condujeron a comienzos de los años noventa, junto con el desarrollo de los PC y los sistemas de cableado estructurado, al nacimiento de aplicaciones de control, seguridad, comunicaciones que son el germen de la Domótica actual." (Domótica: Un Enfoque Sociotécnico, pag. 15).

"Los sistemas domóticos, también conocidos como sistemas de automatización de hogares, han estado dentro del mercado durante varios años y gracias al incremento de disponibilidad y disminución de costos de dispositivos y guiados por las nacientes necesidades de confort en hogares, además del ahorro energético, seguridad, comunicación y servicios multimedia, iniciaron a expandirse dentro de las construcciones residenciales, hace muy poco tiempo." (Ontology Modeling for Intelligent Domotic Environments, pag.790).

Actualmente las principales aplicaciones que ofrece un sistema domótico son: control de iluminación, confort térmico, elementos de cierre y protección, mobiliario y electrodomésticos.

La arquitectura del sistema domótico

El sistema domótico basa su estructura en un modelo cliente/servidor que interactúan en el envío y recepción de mensajes. Con servidor hacemos referencia a un proveedor de servicios domóticos y el cliente es el consumidor de dichos servicios.

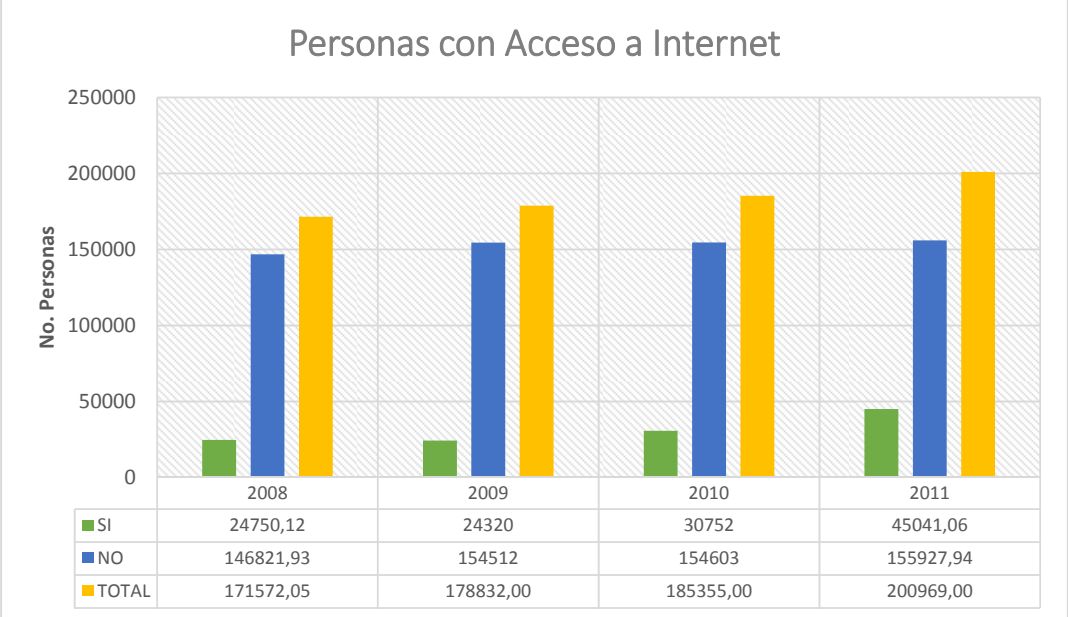
"El modelo cliente - servidor se basa en un protocolo solicitud / respuesta. Es sencillo y sin conexión. El cliente envía un mensaje de solicitud al servidor pidiendo cierto servicio. El servidor ejecuta el requerimiento, regresa los datos solicitados o un código de error si no pudo ejecutarlo correctamente. (Martínez, 2001 pág. 275)

A pesar de que la domótica en el Ecuador es una rama tecnológica muy poco explotada debido a los altos costos del hardware necesario para su implementación, existen varias empresas dedicadas a la venta de equipos de automatización de viviendas, asesoría técnica y distribución e implementación de sistemas domóticos.

En el caso de la ciudad de Cuenca se tiene a la empresa SODEL (Soluciones Domóticas y Electrónicas), la cual proporciona a clientes locales y nacionales soluciones domóticas y electrónicas enfocadas a la venta y aprovisionamiento de equipos tecnológicos para la automatización de viviendas además de la implementación de sistemas domóticos completos.

En los siguientes gráficos se puede observar información estadística en la parte de ciencia y tecnología correspondiente a la provincia del Azuay que influye directamente en la evolución de la domótica dentro de la ciudad de Cuenca:

Acceso a Internet

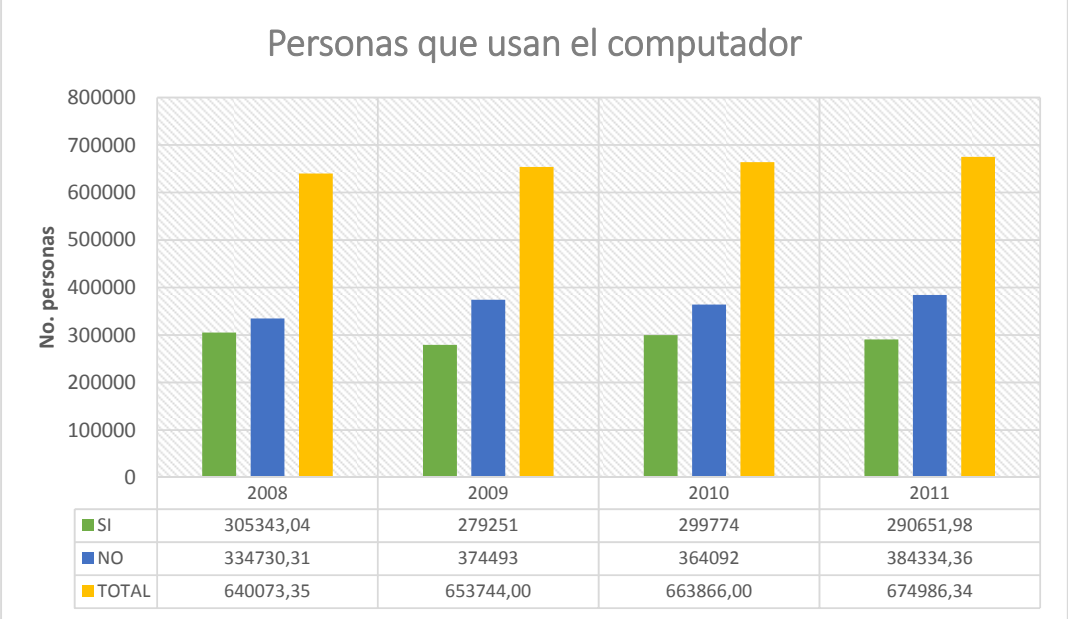


	2008	2009	2010	2011
SI	24750,12	24320	30752	45041,06
NO	146821,93	154512	154603	155927,94
TOTAL	171572,05	178832,00	185355,00	200969,00

Fuente: INEC

Elaborado por: Los autores

Uso de Computadora



Fuente: INEC

Elaborado por: Los autores

2.6. Objetivo General

Desarrollar un sistema domótico prototipo para el control de cámaras de seguridad e iluminación de un hogar mediante dispositivos móviles con tecnología iOS y Android.

2.7. Objetivos Específicos

- Fundamentar teóricamente la domótica y la arquitectura cliente – servidor.

- Analizar la situación actual de la domótica en la ciudad de Cuenca.
- Realizar el análisis y diseño del sistema domótico, de acuerdo a los requisitos.
- Desarrollar la aplicación móvil para controlar los sistemas de iluminación y vigilancia.
- Implementar el sistema sobre un hogar prototipo (maqueta).

2.8. Metodología

Se utilizará el método cualitativo, debido a que se requiere comprender el comportamiento humano de la sociedad actual hacia el avance tecnológico en el área de la domótica. Se utilizará el método de análisis orientado a objetos para el modelado del problema y su posterior solución. En base a la investigación realizada se elaborará un diagrama de casos de uso que represente el comportamiento entre el usuario y la aplicación domótica para dispositivos móviles.

El Web Service estará implementado en el lenguaje de programación Java, mientras que las aplicaciones clientes estarán desarrollados en Objective C (xCode) y Java (Eclipse) para los dispositivos iOS y Android respectivamente.

El funcionamiento de las aplicaciones se pondrá a prueba sobre la maqueta construida, la misma que constará de circuitos electrónicos comandados por medio de un Microcontrolador PIC16F877A, la comunicación entre el circuito y el servidor se realizará a través del puerto paralelo.

2.9. Alcances y resultados esperados

Al finalizar el trabajo de titulación se espera obtener un sistema domótico centralizado y controlado a través de dispositivos móviles tales como celulares y tablets con sistemas operativos de última generación (IOs, Android).

El sistema permitirá el control de la iluminación (encendido y apagado de las lámparas) y manejo de cámaras de seguridad, facilitando la visualización del interior del hogar en tiempo real. Este se dividirá en dos partes, el cliente (usuarios con dispositivos móviles) y servidor o web service. Estos dos actores del sistema se comunicarán entre sí para envió y recepción de peticiones.

Al finalizar se probará y demostrará el funcionamiento del sistema sobre el prototipo construido (maqueta).

2.10. Supuestos y riesgos

El punto crítico dentro de la realización del proyecto que puede retrasar su desarrollo radica en el tiempo que tomará en llegar al país el hardware necesario desde los Estados Unidos. También podría traer problemas el desabastecimiento de los componentes electrónicos en los locales de la ciudad.

2.11. Presupuesto

Rubro-Denominación	**Costo USD**	**Justificación**
Sistema de Iluminación	200,00	Componentes electrónicos necesarios y focos.
Sistema de Vigilancia	180,00	Cámaras IP
Aplicaciones Usuario	600,00	Computador de marca Mac para el desarrollo de las aplicaciones

Derechos de certificación	134,00	Para cumplir con requisitos de la universidad.
TOTAL	1.114,00	

2.12. Financiamiento

El proyecto será autofinanciado.

2.13. Esquema tentativo

Abstract
Introducción
Objetivos

11. **CAPÍTULO 1: Marco Teórico**
    Introducción
    - 11.1. Conceptos principales
    - 11.2. Arquitectura Cliente Servidor
    - 11.3. iOS
    - 11.4. Android

12. **CAPÍTULO 2: Análisis de situación actual**
    Introducción
    - 12.1. Estructura del sistema
    - 12.2. Funcionalidad
    - 12.3. Domótica en Cuenca

    Conclusiones del capítulo

13. **CAPÍTULO 3: Propuesta tecnológica**
    Introducción
    - **13.1. Análisis del sistema:**
      - **13.1.1.** Levantamiento de Información y Requisitos para el Sistema Domótico.
      - **13.1.2.** Análisis Orientado a Objetos del problema (AOO).
      - **13.1.3.** Estudio de alternativas de microcontroladores y dispositivos electrónicos.
      - **13.1.4.** Análisis de Protocolos de Comunicación.
      - **13.1.5.** Modelo del Sistema Domótico
    - **13.2. Hardware y Dispositivos Electrónicos:**

13.2.1. Microcontrolador.
13.2.2. Diseño del Circuito Controlador.
13.2.3. Implementación y Construcción.

13.3. Base de Datos:

13.3.1. Modelo Entidad – Relación.
13.3.2. Creación de la Base de Datos.
13.3.3. Documentación.

13.4. Web Service

13.4.1. Arquitectura.
13.4.2. Funcionamiento.
13.4.3. Implementación.

Conclusiones del capítulo

14. CAPÍTULO 4: Sistemas de Control:

Introducción

14.1. Sistema de Vigilancia
14.2. Sistema de Iluminación

Conclusiones del capítulo

15. CAPÍTULO 5: Aplicaciones Usuario:

Introducción

15.1. Diseño de Interfaz.
15.2. Aplicación para iOS.
15.3. Aplicación para Android.

Conclusiones del capítulo

16. CAPÍTULO 6: Aplicación de la solución

Introducción

16.1. Acoplamiento en el prototipo construido (maqueta).
16.2. Funcionamiento.
16.3. Pruebas.

Conclusiones del capítulo

Conclusiones y recomendaciones generales
Bibliografía

2.14. Cronograma

Objetivo Específico	Actividad	Resultado	Tiempo(Semanas)

		esperado	
Establecer los requisitos del sistema.	Recopilar Requisitos.	Obtener todos los posibles requisitos para la creación e implementación del sistema.	1
	Especificación de requisitos.	Determinar qué requisitos son los que se utilizarán.	1
	Validación de Requisitos.	Comprobar que los requisitos escogidos sean los correctos.	1
	Aprobación de Requisitos.	Se aprobará los requisitos finales.	1
	Modelado del Sistema.	Establecer la distribución y estructura que tendrá el sistema.	1
Construir un prototipo de una vivienda (maqueta de madera) en donde se implementará el sistema domótico para su prueba.	Construcción de la maqueta.	Obtener la maqueta para la implementación del sistema domótico.	1
Diseñar y construir los circuitos electrónicos para el control del sistema de iluminación.	Diseño de los circuitos.	Establecer el diseño más eficiente para el circuito de control del sistema de iluminación.	1
	Construcción de	Implementar	2

	los circuitos.	físicamente los circuitos previamente diseñados.	
Instalar y configurar un *Web Service* y codificar procedimientos para la comunicación entre los circuitos electrónicos y las aplicaciones móviles clientes.	Instalación y configuración del sistema cliente-servidor.	Establecer un medio de comunicación entre el servidor y los usuarios cliente.	1
Implementar el sistema de seguridad con cámaras IP.	Instalación de las cámaras IP.	Obtener el control de cada una de las cámaras IP.	1
Crear las aplicaciones clientes (iOS y Android) en dispositivos móviles que interactuarán con los usuarios.	Diseño de las aplicaciones cliente.	Establecer la mejor estructura y diseño para la aplicación.	1
	Diseño de Interfaces de las aplicaciones cliente.	Establecer una interfaz amigable para que interactúe con el usuario final.	1
	Codificación de la Aplicación para dispositivos iOS.	Aplicación real funcionando para dispositivos iOS.	2
	Codificación de la Aplicación para dispositivos	Aplicación real funcionando para dispositivos Android.	2

	Android.		
Implementar el sistema domótico sobre el prototipo construido (maqueta).	Instalación del sistema en la maqueta.	Poner el sistema en pleno funcionamiento.	1
	Etapa de pruebas.	Determinar posibles fallos del sistema.	1
	Corrección de errores.	Corregir fallos detectados en la etapa de pruebas.	1

2.15. Referencias

- Campo, Domingo. (2005). *Las nuevas tecnologías al servicio de los mayores, Domótica.* Universitat Jaume-I. Castellano de la Plana. pág 34
  Recuperado en: http://mayores.uji.es/proyectos/proyectos2005/domotica.pdf
- Kendall, Kenneth E; Kendall, Julie E. (2005). *Análisis y diseño de sistemas.* México. Prentice Hall. pág 268
- Martínez, David Luis la Red. (2001). *Sistemas Operativos.* pág 275
  Recuperado en: http://sistop.gwolf.org/biblio/Sistemas_Operativos_-_Luis_La_Red_Martinez.pdf
- Novel, Beatriz. (2006) *Clasificación de los Sistemas Domóticos y Normalización en el área domótica*. Asociación de Fabricantes de Material Eléctrico. pág 25
  Recuperado en:
  http://www.cedom.es/fitxers/documents/normativa/Sistemas%20Domoticos%20y%20Normalizacion.pdf
- ThyssenKrupp. (2013). *Accesibilidad.* MLDM Estudio
  Recuperado en: www.mldm.es/BA/x12.shtml
- Valle, Guillermo José; GUTIERREZ James. (2005). *Definición Arquitectura Cliente Servidor.* Tecnología en informática.

2.16. Firma de responsabilidad

_ _ _ _ _ _ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _ _ _ _ _ _
Pedro Andrés Cárdenas	Pedro Sebastián Corral
49498	49217
ESTUDIANTE	**ESTUDIANTE**

2.17. Firma de responsabilidad

_ _ _ _ _ _ _ _ _ _ _ _ _

Ing. Esteban Crespo Martínez

DIRECTOR SUGERIDO

2.18. Fecha de entrega:

Anexo B: Diccionario de Datos

Anexo C: Encuestas para Levantamiento de Requisitos

Formato Encuestas:

UNIVERSIDAD DEL AZUAY

ESCUELA DE INGENIERÍA DE SISTEMAS Y TELEMÁTICA

Encuesta para el levantamiento de requisitos para la realización del trabajo de titulación: "PROTOTIPO DE UN SISTEMA DOMÓTICO PARA CONTROLAR ILUMINACIÓN Y CÁMARAS DE SEGURIDAD MEDIANTE DISPOSITIVOS MÓVILES"

Nombre de la Empresa:

Nombre del Propietario:

Preguntas

1. ¿Qué productos y/o servicios ofrece su empresa?
2. ¿Qué productos y/o servicios son los más solicitados?
3. ¿Su empresa dispone de productos y/o servicios que ofrecen seguridad?
4. ¿Qué ventajas obtendrá el usuario al obtener sus productos y/o servicios?
5. ¿Cuál es el producto y/o servicio más básico que ofrece y cuál es su costo?
6. ¿En cuanto a versatilidad y facilidad de acoplamiento de sus productos y/o servicios qué soluciones puede escoger el usuario?
7. ¿Qué nivel de satisfacción ha obtenido por parte de los usuarios que han consumido sus productos y/o servicios?
8. ¿Cuál es el proceso de instalación de las soluciones domóticas que ofrece su empresa?

Gracias por su tiempo y colaboración.