UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Máster en Ingeniería de Computadores

GESTIÓN DE APARCAMIENTO PARA IPHONE INTEGRADA CON CLOUD COMPUTING

Alumno: Jesús Cano Albir

Director: Enrique Hernández Orallo

Departamento de Informática de Sistema y Computadores

Fecha: Julio 2012

1. INTRODUCCIÓN

El objetivo perseguido en la realización de la presente tesis de máster, ha sido la realización de una aplicación móvil destinada a realizar estacionamientos de vehículos de una forma más eficiente y de un modo colaborativo, orientado a aplicar dicha aplicación de un modo práctico.

El propósito de la aplicación , de un modo resumido, es almacenar información sobre posibles emplazamientos para estacionar nuestro vehículo (basándonos en la coordenadas de los individuos en un momento determinado) con la finalidad de que el conductor disponga de una sencilla herramienta que le pueda facilitar el estacionamiento del mismo. De este modo, un usuario podrá acceder a las facilidades aportadas por la aplicación mediante las coordenadas que proporcione su localizador GPS integrado en el dispositivo móvil.

Revisando posibles opciones e implementaciones para poder llevar a cabo, se observa que en la actualidad los problemas de estacionamiento son habituales en la ciudades, por lo que hemos optado por destinar las capacidades y herramientas que proporciona el geoposicionamiento de los dispositivos móviles, así como, las ventajas que nos ofrecen los servicios de “Cloud Computing” para poder diseñar una aplicación que facilite y agilice la tarea de estacionamiento de vehículos.

En concreto, el dispositivo escogido para la realización del mencionado proyecto ha sido el **iPhone** de Apple y la tecnología escogida para llevar a cabo las tareas de “Cloud Computing” son las ofrecidas por Google, mediante su servicio **Google App Engine**.

Una vez hemos puesto en situación en lo referente al contenido de la memoria, podemos pasar a la descripción más detallada de la misma y relatar de un modo más clarividente el aspecto y funcionamiento de las herramientas empleadas.

MOTIVACIÓN

Los motivos que me han impulsado a decantarme por este tipo de tecnologías para el desarrollo de mi aplicación son por un lado la posibilidad de poder desarrollar para una plataforma móvil, en concreto iPhone debido al auge y popularidad de la que goza en la actualidad, así como aprender su lenguaje y metodología, totalmente desconocidas para mi, para poder ampliar mis conocimientos y por otro lado la posibilidad de poder combinar dicha aplicación con los servicios de "Cloud Computing" que también están gozando de una gran popularidad ya que proporcionan gran flexibilidad al desarrollador para poder llevar a cabo sus tareas.

OBJETIVO

El objetivo que persigo en la realización de este proyecto es aprender sobre un nuevo e interesante campo de la programación para dispositivos móviles (dispositivos iPhone y compatibles en concreto) y basándome en dicho aprendizaje, realizar una aplicación en la que pueda extrapolar los conocimientos adquiridos combinados con la utilización de Google App Engine para poder hacer que la aplicación sea más flexible y móvil haciendo que la información de la que queremos disponer se obtenga en tiempo real, algo parecido como lo que ocurre en otro tipo de plataformas muy populares como pueden ser Facebook o Twitter.

Todo esto además, se pretende llevar a cabo con un diseño que sea sencillo de cara al usuario para que puede utilizar la aplicación sin acciones complejas, de una forma intuitiva, pues como usuario habitual de la plataforma móvil que se ha escogido, mi preferencia a la hora de poder instalar una aplicación en mi dispositivo, es que la misma, sea sencilla, fácil e intuitiva.

ALCANCE

El proyecto pretende realizar una aplicación destinada a dispositivos iPhone y compatibles (iPad, iPod Touch) destinada a aquellos usuarios que dispongan de dicha tecnología (puesto en la parte referente a Google App Engine es totalmente transparente al usuario) con el fin de facilitar en la medida de lo posible las tareas de estacionamiento de vehículos, pues la misma se ha convertido en una ardua tarea en emplazamientos de gran volumen de tráfico. De este modo se presenta información

(calle y ciudad) de un modo colaborativo sobre ciertos lugares en los que se encuentra un vehículo estacionado y que previamente ha utilizada dicha aplicación para poder notificar la reserva/estacionamiento del vehículo en la una posición determinada. Del mismo modo ese mismo usuario cuando fuese a desalojar su plaza de estacionamiento, puede llevar a cabo la liberación/notificación de dicha plaza para que aparezca en un listado y que otro usuario potencial de la aplicación pueda observar los lugares más cercanos a él en los que podrías estacionar su vehículo y poder realizar la pertinente reserva/estacionamiento de su vehículo.

La aplicación en sí se compone de diferentes tareas que se llevan a cabo en cada una de las tres diferentes sub-vistas que la componen:

- Por un lado tendríamos la vista **Home**. Esta sería la vista principal de la aplicación y la cual se ejecuta en el momento que la aplicación arranca desde el dispositivo. En dicha vista podríamos realizar la reserva de la posición donde se va a estacionar, en el caso de que no se encontrase en la lista de las libres previamente establecidas, mediante un botón destinado a tales efectos. Mientras que por otro lado podríamos realizar la liberación de ese hueco reservado, la cual, nos llevará a otra vista en la que podremos observa un listado de todos los huecos de aparcamiento que se encuentran ocupados en la actualidad. Para llevar a cabo la liberación simplemente bastaría con editar la tabla y borrar la celda correspondiente a nuestro hueco de estacionamiento y la aplicación se encargará del resto.

- La segunda vista, está denominada con el nombre **Ubicación**. En la misma y tal como se puede intuir por su nombre, aparece una vista de un mapa, el cual nos muestra la posición actual en la cual nos encontramos para tener una visión general de la zona por la que nos estamos moviendo.

- Por ultimo tenemos la vista **Resumen**, la cual nos ofrece un listado de las posibles plazas de estacionamiento que podríamos ocupar y que se encuentran libres, previa liberación de los usuarios en la vista Home mencionada anteriormente. El mecanismos para llevar a cabo la reserva del lugar de estacionamiento es análoga a la que utilizábamos en la vista Home, por lo que solo tendremos que editar la vista de tabla que se nos presenta, eliminando la celda de la misma correspondiente al lugar que queremos reservar y automáticamente, éste desaparecerá de listado para aparecer en el listado en el que se muestran los huecos que se encuentran ocupados en la actualidad.

De este modo lo que nos permite esta aplicación es poder agilizar las tareas de estacionamiento de vehículos así y por consiguiente poder mejorar las condiciones de circulación de las calles por las que transitamos. Esta tarea resulta bastante costosa y desesperante en ocasiones, por lo que el carácter colaborativo que tiene esta aplicación, hace que dicha búsqueda se lleve a cabo de una forma más sencilla y directa.

También ofrece ventajas en cuanto al ahorro de combustible pues con el uso de esta aplicación conseguimos ir de un modo directo al lugar donde vamos a estacionar el vehículo sin necesidad de tener que estar dando vueltas constantes para poder localizar el lugar.

Y aunando los dos puntos de vista anteriores, podemos también decir que el uso de la aplicación incorpora ventajas ecológicas, puesto que cuanto menos tiempo estemos circulando con nuestro vehículo, menos gases estaremos emitiendo a la atmósfera

2. ESTADO DEL ARTE

Es aquí es donde procederemos a realizar un pequeño repaso en cuanto a las tecnologías destinadas a la realización de la tarea que nos ocupa para tener una idea general del estado actual respecto a lo que podemos encontrar en el mercado en la actualidad.

Hay que tener en cuenta a la hora de realización del proyecto, las características propias de cada uno de los productos disponibles en el mercado para observar cual es el que mejor se adapta a nuestras características y necesidades. Más concretamente nos estamos refiriendo al hecho de que al centrar las posibilidades y habilidades de nuestra aplicación a la aplicación de términos de posicionamiento con un fin determinado, deberemos escoger un dispositivo o tecnología que presente características fiables a tales efectos, para que el funcionamiento de la misma sea lo más preciso y fidedigno posible.

PLATAFORMAS MÓVILES

Como decíamos anteriormente, existen diversas plataformas móviles sobre las que desarrollar nuestro proyecto, decantándose por uno por otro dependiendo de las circunstancias citadas. Principalmente podemos distinguir, de entre todas las posibilidades, cuatro grandes plataformas que copan el panorama móvil actual:

Se trata de un sistema operativo móvil desarrollado por Microsoft y que fue lanzado en 2010, concebido para reemplazar la plataforma “Windows mobile”.

A diferencia de su antecesor está destinado al sector de uso cotidiano y generalista, no con fines comerciales y empresariales como su antecesor.

La compañía con sede en Redmon, decidió dar un vuelco a la tecnología móvil que venía desarrollando hasta entonces y esa revolución les llevó a desarrollar un nuevo sistema operativo móvil objeto de nuestra descripción.

Fruto de esa revolución y cambio de rumbo por parte de Microsoft, obtenemos un sistema operativo que presenta una nueva interfaz (“Metro”) que enfatiza el contenido y

la tipografía haciendo un mayor énfasis en la utilización del teléfono con los dedos comparado con “Windows Mobile”.

Presenta además una fuerte integración con la nube y redes sociales y también establece una serie de requisitos mínimos de hardware para que los fabricantes de móviles puedan hacer uso de este sistema operativo. Este hecho hace que aparezcan ciertas ventajas para los desarrolladores de aplicaciones para esta plataforma, puesto que en cierto modo, se reduce la fragmentación de la plataforma al no tener diferentes tamaños, formas, etc. sobre las que desarrollar dichas aplicaciones.

Como resultado tenemos un sistema operativo que está siendo introducido en numerosos dispositivos basándose en el uso de una interfaz de fácil manejo con un aspecto más fluido y mucho más optimizada para estar a la altura de las demás compañías y poder compartir cuota de mercado.

Sin embargo la tardía llegada de este sistema al mercado ha hecho que los sistemas contra los que pretende competir le lleven mucha ventaja en cuanto a usuarios consumidores se refiere, lo que hace que su cuota de mercado sea insignificante comparado con el resto.

El catálogo de aplicaciones disponibles para esta plataforma es bastante reducido, limitándose a las aplicaciones más populares por lo que nos hace llevar a pensar que todavía le queda un largo camino por recorrer para equipararse con el amplísimo catálogo de aplicaciones disponibles para sus mas inmediatos competidores.

Como conclusión, podemos extraer que pese a la tardía implantación del sistema y su reducido catálogo de aplicaciones disponibles, hubiese sido una posibilidad viable poder haber desarrollado la aplicación para este sistema operativo, sin embargo, las posibilidades de mercado y desarrollo para esta plataforma se presentan bastante reducidas si tenemos en cuenta la aceptación de otro sistemas operativos.

A día de hoy, podemos decir que es el principal competidor para el sistema iOS de Apple, alcanzando cuotas de mercado superiores a las de la compañía de Cupertino tal y como muestra el dato que revela que durante el cuarto trimestre de 2011, llegó a alcanzar una cuota de mercado de un 50,9% (más del doble del segundo sistema operativo más utilizado que fue iOS), pues goza de la confianza de los fabricantes debido a que fue el único sistema operativo abierto disponible para hacer frente al sistema propietario de Apple.

Se trata de un sistema operativo basado en Linux y que es desarrollado por la "Open Handset Alliance" (consorcio de 78 compañías de software, hardware y compañías móviles destinadas al desarrollo de estándares abiertos y liderada por Google) lo que le confiere una virtud imprescindible que no es otra que la de ser una plataforma libre (se puede acceder tanto al código fuente como al listado de incidencias , pudiéndose reportar nuevos problemas).

Esto nos lleva a un sistema que tiene una gran aceptación, basado en la arquitectura ARM, multitarea y que puede ser instalado en cualquier terminal. Todo aunado ha hecho que este sistema se encuentre en numerosos dispositivos en el mercado y que goce de una amplísimo catálogo de aplicaciones destinadas a él, la mayoría de ellas gratuitas, teniendo en cuenta únicamente el "Android Market" oficial del sistema, aunque existen otros lugares donde también podríamos encontrar aplicaciones destinas a este sistema como bien puede ser el App Store de Amazon.

El hecho de que se trate de un sistema operativo abierto en el que cualquiera puede aportar ideas e implementaciones, constituye en cierto modo un arma de doble filo, pues esto puede acarrear ciertas desventajas en el sistema.

La rápida evolución de los dispositivos, de un tiempo a esta parte, conlleva que los sistemas deben adaptarse a dichos cambios para poder sacar el mayor rendimiento de los mismos a la misma vez que deben funcionar en los antiguos terminales que ya disponían de dicha tecnología por lo que se debe establecer una especie de compendio para poder garantizar el correcto funcionamiento en todos los terminales.

Por otro lado encontramos la tendencia que existe en la actualidad respecto al aumento de las pantallas de los terminales en cuestión. Esto convierte la tarea de programar para este tipo de sistemas en una tarea complicada por el hecho de tener que adaptar el desarrollo a diferentes configuraciones (cosa que no ocurría en Windows Phone como comentábamos al disponer de estándares hardware). Si a este problema le añadimos el anteriormente mencionado del rendimiento, observamos que obtener el compendio de consumo y productividad de los terminales se antoja una tarea que en ningún caso es trivial.

Existen también otros factores que han conllevado a que este sistema no le haya ganado la partida al afamado sistema de Cupertino como bien pueden ser factores de calidad de los dispositivos, suavidad en el manejo de los mismo, el uso poco intuitivo de la interfaz o el hecho de ciertos problemas con la multitarea que llevan a los dispositivos a tener que utilizar ciertas aplicaciones que trabajen sobre otras aplicaciones con el fin de poder cerrar la tarea por completo y conseguir un uso más eficiente de las baterías de los terminales.

Esta tendencia ha experimentado un cambio en la actualidad con la aparición de terminales de alta calidad como pueden ser los pertenecientes a la saga "Galaxy" de

Samsung que se equiparan a nivel de calidad de los dispositivos de su principal competidor (Apple).

Por último podemos citar las posibilidades comerciales que puede conllevar desarrollar para este tipo de sistema. Como citábamos al comienzo de la exposición de este sistema, podemos encontrar un amplio catálogo de aplicaciones disponibles para el mismo, siendo la gran mayoría de ellas de carácter gratuito, los que nos lleva a deducir que desarrollar para el mismo puede suponer un escaparate antes la gran cantidad de usuarios potenciales del sistema.

Como resumen podemos llegar a la conclusión de que por los avances realizados por este sistema, unido al respaldo de las compañías telefónicas y a su carácter abierto, que finalmente este sistema operativo terminará imponiéndose en todas las facetas a sus inmediatos competidores, debido al respaldo que tiene por parte de compañías y usuarios y que hace que las ventajas acaben imponiéndose a las desventajas que puedan surgir del mismo.

BlackBerry

Cuando hablamos de Blackberry nos estamos refiriendo a una línea de teléfonos inteligentes desarrollados por la compañía RIM (Research In Motion) que integra el servicio de correo electrónico móvil además de incluir ciertas aplicaciones típicas en los smartphones. Sus dispositivos son mundialmente conocidos por su teclado integrado (QWERTY), utilizando como sistema operativo el denominado Blackberry OS.

Sin duda, se trata de un tipo de dispositivo que experimento su mayor auge hacia principios de los años 2000, más concretamente en 2002, cuando apareció en el mercado un modelo que soportaba push e-mail, telefonía móvil, mensajería de texto, faxes por Internet, navegación web y otros servicios informáticos inalámbricos, lo cual, hizo que gozase de una maravillosa acogida sobre todo en el mundo ejecutivo.

Sin embargo se trata de una tecnología que se encuentra en una clara decadencia. La competencia en el mundo de los smartphones es cada días mayor y el principal punto fuerte del que disponían los dispositivos de RIM (mensajería por correo eléctronico) se ha visto amenazada por otras tecnologías (Android, Iphone, etc) que manejan aceptablemente el uso del correo electrónico, además de presentar un amplio catálogo de aplicaciones disponibles para instalar en dichos dispositivos y, por lo que las tecnologías táctiles actuales están comiendo prácticamente todo el terreno y cuota de mercado.

Teniendo en cuenta estos aspectos podemos observar los motivos por los cuales esta tecnología ha ido perdiendo cuota de mercado respecto al resto. Además incidentes como el protagonizado recientemente, el cual, presentó problemas en las

comunicaciones de mensajería instantánea para usuarios de Blackberry en todo el mundo no ayudan precisamente a que esta tecnología pueda presentar una posible recuperación.

Todo y con eso todavía siguen contando con un grupo fiel de seguidores y continúan sacando al mercado nuevos dispositivos de gama baja para poder ir recuperando un poco de terreno respecto al resto.

La solución que han adoptado desde las altas esferas de RIM ha sido el lanzamiento de su nuevo sistema operativo QNX, con el que pretenden ser competitivos frente al resto de compañías y en el cual tienen depositadas grandes esperanzas para poder revertir la situación.

En cuanto al catálogo de aplicaciones disponibles para este tipo de dispositivos, podemos decir que no goza de un amplio número de las mismas y más teniendo en cuenta el hecho de que la mayoría de aplicaciones que se desarrollan en la actualidad van destinadas a dispositivos en los que se puedan realizar acciones de carácter táctil, por lo que los desarrolladores están dando cada vez mas la espalda al desarrollo para dicha plataforma para poder centrarse en tecnologías mucho mas competitivas, aunque como decía en el párrafo anterior, los directivos de RIM tienes depositadas sus esperanzas en el sistema operativo QNX y confían en poder revertir la situación.

*** NOTA: En este breve repaso he obviado mencionar otro tipo de tecnologías también disponibles en el mercado (Symbian, Bada, etc) para centrarme en las de mayor volumen de cuota de mercado y por tanto las que constituye el bloque principal de demanda de los usuarios.**

Por último concluiremos con la tecnología por la cual he optado para el desarrollo de mi proyecto que no es otro que la basada en los dispositivos iOS de Apple, por los motivos previamente especificados.

Se trata de un sistema operativo para dispositivos móviles propios de la compañía Apple tales como iPhone, iPad e iPod, basado en el Mac Kernel de Mac OS X. Los principales motivos que han provocado el auge de esta tecnología los podemos relatar a continuación:

- Hardware independiente

  Apple establece un hardware independiente para sus dispositivos, por lo que al estar todos bajo un estándar de diseño, permite al desarrollador poder realizar

aplicaciones que aprovechen al máximo las capacidades de los terminales sin detenerse en establecer concordancias de rendimiento provocadas por los diferentes diseños y modelos existentes en otro tipo de tecnologías como bien puede ser Android.

- Distribución de aplicaciones

  Apple dispone de su propia plataforma en la cual poder distribuir de un modo sencillo el amplio número de aplicaciones disponibles para sus dispositivos. Además el auge experimentado por el uso de sus terminales ha hecho que el software propietario propio de la compañía y el lenguaje de programación destinado a ello haya experimentado un auge importante llegando incluso a doblar tasas del año anterior, obteniendo un 6,9%, cifras que aúpan al lenguaje a situarse entre los más utilizados por los desarrolladores.
  Enlazando con el tema del desarrollo para iOS, es conveniente mencionar que la estrategia ideada por los de Cupertino se basa en centralizar el filtro de aplicaciones de tal modo, que todas pasan por ellos para cerciorarse de que el software proporcionado cumple con los requisitos imprescindibles para poder ser aceptado para su distribución.

- Funcionamiento

  Todo lo contado anteriormente aunado con su fácil manejo y una experiencia de usuario bastante depurada, nos llevan a aupar a los dispositivos Apple como uno de los mejores existentes en el mercado actual.

Como mencionábamos más arriba, la programación para los dispositivos de Apple utilizan un lenguaje denominado "Objective-C" y que es compatible para aquellos dispositivos que pertenecen a la compañía salvo con pequeñas diferencias y con esto me refiero a que programar para iPhone es totalmente compatible con iPod, sin necesidad de realizar ningún ajuste previo y la única diferencia puede residir en la ausencia del último de GPS y de cámara fotográfica, mientras que programar para iPad requiere de ciertos ajustes debido a la mayor de la pantalla de este último para poder rentabilizar al máximo todo el espacio que se nos ofrece.

Es precisamente el hecho de poder disponer de una pantalla de mayores dimensiones en el iPad la que nos puede condicionar a la hora de decantarnos por un dispositivo y otro ya que lógicamente, a mayor espacio de diseño, podremos establecer mayor numero de acciones que deberá realizar la aplicación.

El hecho de decantarnos por uno u otro dispositivo reside lógicamente en la finalidad de la aplicación. Con esto nos estamos refiriendo al hecho de que si el fin de nuestra aplicación es más bien mostrar todo tipo de documentación en la que el objetivo primordial es la buena visibilidad de los parámetros y acciones establecidas, sin duda, elegiríamos iPad como nuestro dispositivo de desarrollo. Sin embargo si consideramos que el objetivo de nuestra aplicación es meramente de consulta, podemos decantarnos por el dispositivo iPhone para llevar a cabo nuestra tarea.

Dicho todo esto, cabe destacar, que por la pruebas realizadas en el simulador, la aplicación llevada a cabo funciona del mismo modo en ambos dispositivos, el problema reside en que tal y como comentábamos, si ejecutásemos la misma en un dispositivo iPad no estaríamos aprovechando todas las posibilidades que el mismo nos ofrece en cuanto a tamaño de la pantalla.

Por lo tanto después de haber establecido las bondades de cada uno de los dispositivos Apple sobre los que desarrollar nuestra aplicación, podemos realizar un repaso de las características propias del dispositivo elegido, iPhone.

IPHONE

Basándonos en el libro **Desarrollo de Aplicaciones para iPhone - SAMS**, podemos describir el funcionamiento del dispositivo estableciendo cuatro puntos principales:

1) Pantalla y gráficos

La pantalla del dispositivo tiene una resolucion de 960x640 píxeles con la llegada de la pantalla retina en los iPhone 4/4S (frente a los 480x320 de su predecesor, el iPhone 3/3Gs), por lo que nos ofrece un espacio acotado a estas dimensiones para presentar la interfaz y el contenido de nuestras aplicaciones. Las aplicaciones de iPhone también eliminan la noción de las ventanas múltiples, disponiendo de una única ventana con la que poder trabajar. En cualquiera de las versiones del dispositivo el tamaño de la pantalla sigue estando acotado a 3,5 pulgadas.

Si bien decíamos que el espacio suponía una limitación a la hora de poder presentar nuestras aplicaciones sobre la pantalla del dispositivo, podemos utilizar herramientas propias de diseño que nos lleven a poder realizar aplicaciones tan complejas como puede resultar un software de escritorio.

Las opciones a la hora de mostrar gráficos pueden incluir complejas visualizaciones animadas en 2D y 3D gracias a la implementación OpenGL ES (estándar industrial orientado a la definición y manipulación de imágenes, ampliamente utilizado en la creación de juegos, principalmente) disponible en todos los modelos de iPhone y que son mejorados mediante un chipset 3D actualizado.

2) Limitaciones en los recursos de aplicaciones

Apple se ha esforzado mucho para hacer que el iPhone responda correctamente sin importar lo que estamos haciendo. Sin embargo, desafortunadamente, esto ha contribuido a una de las limitaciones más grandes en la plataforma: sólo puede ejecutarse al mismo tiempo una aplicación de terceros. Esto significa que el programa debe proporcionar las funciones necesarios para el usuario sin que éste tenga que acceder a otras aplicaciones. Del mismo modo, también quiere decir que la aplicación debe permanecer activa para comunicarse con el usuario.

Otra de las limitaciones que debemos tener el cuenta a la hora de realizar nuestros desarrollos para esta plataforma es la limitación de memoria RAM, pues sólo disponemos de de 512 MB en el caso de los dispositivos iPhone 4/4s y iPods de nueva generación y de 256 MB en el caso de iPhone 3Gs y 128 en el 3G. Por lo que esta observación deberá ser tenida a la hora del desarrollo para libera aquellos recursos de los cuales no se vaya a volver a hacer uso, para que otros ocupen su lugar en memoria y tener un óptimo rendimiento.

3) Conectividad

Se trata precisamente de una de las áreas en las que este dispositivo brilla, debido a su capacidad para mantenerse conectado permanentemente a través de la conexión por medio de un proveedor móvil, viéndose esta capacidad mejorada en la medida que van apareciendo en el mercado nuevas versiones del dispositivo.

Además, este acceso a Internet se ve complementado mediante el acceso a las tecnologías Wifi (puede ofrecer velocidades similares a las que podríamos obtener en nuestro propio ordenador) y Bluetooth (usado para tener conectados varios dispositivos a nuestro terminal).

Lógicamente durante la etapa de desarrollo, puede hacerse uso de este tipo de tecnologías para mantener actualizada nuestra aplicación en el caso de que así lo requiriese.

4) Entrada y retroalimentación

El iPhone es especialmente brillante cuando hablamos de este tipo de mecanismos, así como en las capacidades de las que disponemos a la hora de trabajar con ellos. Lo que nos permite este tipo de técnicas es poder recibir datos de entrada desde la pantalla multi-táctil, captar el movimiento e inclinación por medio del acelerómetro o determinar la posición actual por medio del GPS (precisamente esta es la técnica que tendremos en cuanta a la hora de desarrollar nuestra aplicación).

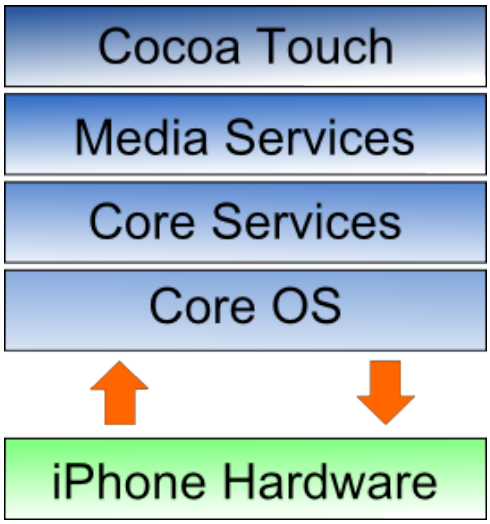
El iPhone también es compatible con la captura de imágenes y video directamente desde nuestras aplicaciones, abriendo así un amplio abanico de posibilidades para poder interactuar con la aplicación.

Por último cabe mencionar que, para cada acción que el usuario realice en la aplicación, podemos proporcionar retroalimentación, refiriéndonos con esto a que podemos responder a cualquier acción que se realice sobre nuestra aplicación con elementos disponible dentro del propio dispositivo, como bien pueden ser vibraciones, sonidos, alertas, etc.

APPLE IOS

Cuando nos referimos a este concepto, estamos haciendo referencia al sistema operativo propio de los dispositivos Apple. Se trata de un sistema operativo basado en una variante del Mach Kernel propio de Mac OS X.

Para realizar una descripción más comprensiva y que pueda resultar clarividente vamos a basarnos en el la siguiente figura, la cual constituye el diagrama correspondiente a la arquitectura del sistema.



- **Core OS Layer**: Se trata de la capa que ocupa el lugar más bajo de la pila que compone el sistema operativo y como tal, se asienta directamente sobre el hardware del dispositivo. Esta capa es la encargada de proveer una variedad de servicios incluyendo los relacionados con el nivel de red, acceso a accesorios externos y los usuales servicios de sistema operativo como la administración de memoria o el manejo del sistema de ficheros e hilos de ejecución.

- **Core Services Layer**: Esta capa contiene los servicios fundamentales del sistema, los cuales, son usados por todas las aplicaciones. Incluso si no lo usamos directamente, puede que algunas partes del sistema estén construidas en base a esta capa.

- **Media Services Layer**: Es la encargada de gestionar todo lo relacionado con las tecnologías gráfica, de audio y video orientadas a la existencia de una mejor experiencia multimedia disponible en el dispositivo.
- **Cocoa Touch Layer**: La última capa de la arquitectura contiene los framework clave para poder construir aplicaciones en el sistema operativo iOS. De este modo define la estructura básica de aplicación y da soporte a tecnologías clave como son la multitarea, la entrada basada en pulsaciones, las notificaciones push (básicamente se refiere a notificaciones que indican que existe nueva información disponible) y otras tantas tecnologías de alto nivel.

OBJECTIVE-C

Se trata de lenguaje de programación utilizado para poder realizar aplicaciones que han de estar disponibles para dispositivos Apple y que fue creado hallá por 1980.

Objective-C es un lenguaje de programación simple diseñado para habilitar el desarrollo de una sofisticada programación orientada a objetos. Este es definido como un pequeño pero potente conjunto de extensiones del lenguaje estándar ANSI C, por lo que constituye un enfoque de programación orientada a objetos completa al lenguaje C, haciéndolo de una forma sencilla.

Como ejemplo de lo sencillo que puede llegar a ser este lenguaje podemos poner un ejemplo y observar que se puede deducir de forma intuitiva y sin la necesidad de soluciones complejas, cual será el resultado de la ejecución de dicho fragmento de código:

```
NSString *string;

[string isEsqualToString:@"Hola mundo" ];
```

Como podemos observar no parece complicado deducir cuál será el resultado almacenado en la variable **string.** Se declara una variable de tipo String que en este caso recibe el otro nombre concreto y a se le asigna un valor en concreto mediante un método que a juzgar por su nombre parece bastante ilustrativo de la acción que llevará a cabo.

XCODE

Se trata del IDE, o entorno de integrado de desarrollo, que gestiona los recursos de nuestras aplicaciones y nos permite editar el código que enlaza entre si las distintas piezas.

La herramienta aúna e integra todo lo necesario para el desarrollo de nuestras aplicaciones, permitiéndonos en una única herramienta disponer de todos los elementos y accesorios necesarios para la realización de la misma (desarrollo, depuración, diseño,…).

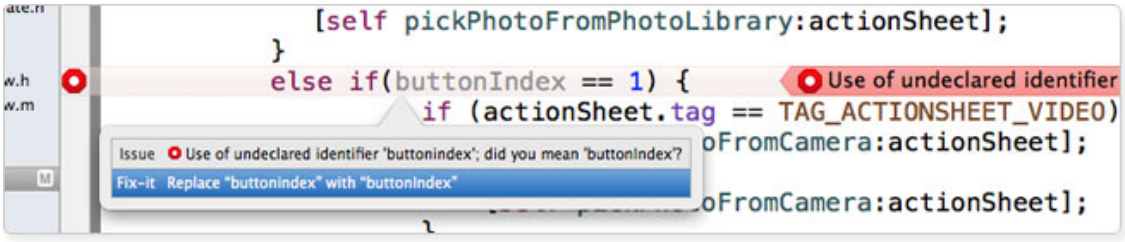
Además incorpora herramientas que nos permiten obtener a tiempo real una visión de cuan correctamente se esta llevando a cabo nuestro proyecto, debido a que utiliza una herramienta denominada "live issues" que nos va informando mediante la aparición de un globo informativo, de la posible presencia de errores o advertencias en las sentencias escritas para desarrollar nuestro código.

Una vez tenemos una visión un poco general de lo que representa Xcode, podemos pasar a detallar los componentes de los que dispone, para aunar en una sola herramienta todos los medios necesarios para el desarrollo.

COMPILADOR APPLE LLVM (APPLE LLVM COMPILER)

El compilador incorporado en la herramienta va más allá del simple hecho de compilar nuestra aplicación. Básicamente se basa en la idea de utilizar el mismo compilador que utiliza Xcode para archivos C/C++ se utiliza también para compilar archivos en lenguaje Objective-C. Se encuentra continuamente evaluando las expresiones introducidas durante el desarrollo mediante la citada herramienta Live issues y partir de las mismas propone un abanico de posibles soluciones en tiempo real, con la finalidad de poder eliminar el fallo detectado, pudiendo elegir el usuario entre la mas adecuada al comportamiento deseado en el código.

Como muestra de esta explicación, podemos observar una imagen que nos indica a las claras lo que se pretende explicar.



INSTRUMENTS FOR PERFORMANCE AND BEHAVIOR ANALYSIS

Lo que se pretende a la hora de desarrollar una aplicación es que la experiencia de usuario sea buena y no simplemente se refiere a cuestiones de diseño y a un funcionamiento intuitivo. Las aplicaciones deben responder de un modo óptimo para mejorar todavía mas si cabe lo conseguido en la fase de diseño y eso es precisamente lo mas importante.

Es por esto que Xcode incorpora una herramienta de desarrollo denominada Instruments que se encarga de localizar los cuellos de botella que pueden darse en el funcionamiento de nuestra aplicación. Para ello, recolecta datos tales como disco, memoria y CPU con nuestro dispositivo conectado, mostrándonos de manera gráfica los resultados, hecho que nos lleva a poder identificar y localizar de un modo rápido la causa de los problemas de nuestra aplicación para poder solventarlos.

SIMULADOR IOS

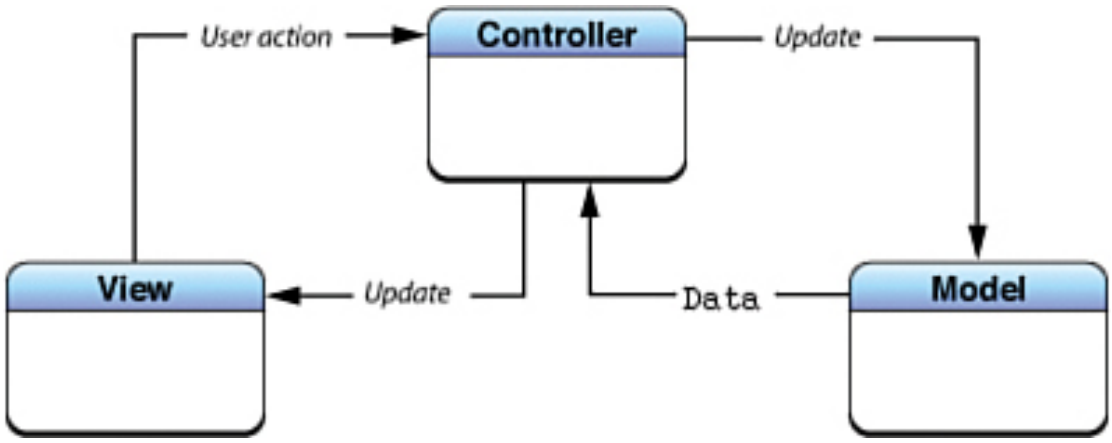
Es el encargado de ejecutar nuestra aplicación del mismo modos de que se produciría en un dispositivo real. Debido a que es rápido de ejecutar y de depurar, esta herramienta se convierte en un test perfecto para nuestra aplicación y así poder hacer una estimación sobre el correcto comportamiento que tiene nuestra aplicación frente a los diferentes casos de prueba a los que puede verse sometida.

MODEL-VIEW-CONTROLLER

Finalmente para encajar todas las piezas de puzle descrito y que componen el modo de desarrollo llevado a cabo para realizar nuestras aplicaciones, debemos mencionar en el patrón de diseño que han de seguir las mismas, ajustándose al patrón establecido por la compañía tanto en iPhone como en Macintosh, y este no es otro que MVC (Model-View-Controller). Se trata de un modo de mantener organizados nuestros proyectos de aplicaciones, para que podamos ampliarlos y actualizarlos en el futuro, estableciendo una separación clara entre los componentes críticos de nuestra aplicación, definido en tres partes:

- Un modelo proporciona los datos internos y los métodos que ofrecen información al resto de la aplicación (el modelo no define la apariencia o el comportamiento de la aplicación.
- Una o más vistas construyen la interfaz de usuario.
- Un controlador suele estar vinculado con una vista. El controlador es responsable de recibir la entrada del usuario y reaccionar en consecuencia.

Como muestra explicativa del funcionamiento del modelo, se expone a continuación una imagen ilustrativa que ayuda a extrapolar los conceptos teóricos del modelo a una compresión visual de las acciones que se llevan a cabo en el mismo.



GOOGLE APP ENGINE

Se trata de un servicio que nos permite poder ejecutar nuestras aplicaciones en la propia infraestructura de **Google.** Las aplicaciones App Engine son fáciles de crear, de mantener y de ampliar al ir aumentando el tráfico y las necesidades de almacenamiento de datos. Con App Engine no es necesario utilizar ningún servidor: simplemente basta con subir la aplicación para que los usuarios puedan empezar a utilizarla.

Se puede dotar a nuestra aplicación de su propio dominio (http://www.example.com/) o bien utilizar el que la plataforma nos ofrece por defecto appspot.com.

Se puede utilizar este servicio de forma totalmente gratuita siempre y cuando no supere los 500 MB de espacio de almacenamiento y pueden utilizar la suficiente CPU y ancho

de banda como para permitir un servicio eficaz de alrededor de cinco millones de visitas a la página al mes.

Los servicios de los que podemos disponer a la hora de poder implementar para esta tecnología son tales como:

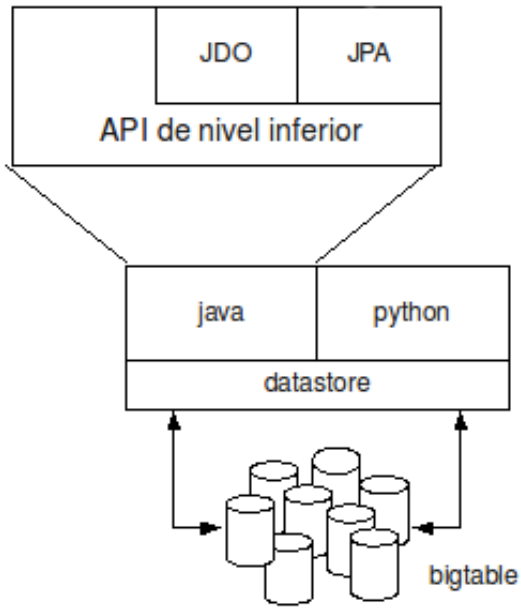
- servidor web dinámico, totalmente compatible con las tecnologías web más comunes.
- almacenamiento permanente con funciones de consulta, clasificación y transacciones.
- escalado automático y distribución de carga.
- API para autenticar usuarios y enviar correo electrónico a través de Google Accounts.
- un completo entorno de desarrollo local que simula Google App Engine en tu equipo.
- colas de tareas que realizan trabajos fuera del ámbito de una solicitud web.
- tareas programadas para activar eventos en momentos determinados y en intervalos regulares.

Sobre todos los servicios relatados podemos basar el funcionamiento de nuestra aplicación escogiendo aquel que se adapte mejor a las prestaciones que queramos ofrecer al usuario. Sin duda, tanto la amplia variedad de posibilidades y servicios ofrecidos así como la gratuidad, hacen de este servicio una opción bastante atractiva a la hora de poder desarrollar nuestras aplicaciones sin la necesidad de tener que montar un servidor propio e independiente.

JDO (JAVA DATA OBJECT)

Se trata de una especificación de almacenamiento de objetos en Java, la cual, no se limita al almacenamiento de objetos en bases de datos relacionales únicamente, pudiendo también utilizarse contras ficheros XML, documentos de OpenOffice, objetos JSON, etc. Esta versatilidad facilita la integración entre JDO y el datastore de Google App Engine, el cual, no utiliza un modelo relacional por debajo.

Lo que si utiliza JDO por debajo es el API de nivel inferior y como muestra de ello la siguiente figura pretende ilustrar esta afirmación.



El API JDO es uno de los APIs más fáciles de aprender de toda la tecnología existente actualmente para la persistencia de objetos estandarizada. Una razón de la simplicidad de JDO es que permite trabajar con objetos normales de Java (POJOs plain old Java objects) en lugar de con APIs propietarios. Además permite que los desarrolladores puedan centrarse más en qué quieren buscar, almacenar, eliminar, modificar, etc. que en el cómo hacerlo.

El poder centrarnos en el qué queremos hacer en lugar del cómo, nos hace disponer de un grado de abstracción alto, claramente mayor que el que tendríamos si tuviésemos que desarrollar nuestra aplicación con el API de nivel inferior, sin embargo, perderemos el disponer de un control absoluto sobre lo que hacemos.

JDO ha recibido ataques por muchas razones, entre las que se incluyen: el modelo de mejora de código debería reemplazarse por un modelo de persistencia transparente; los vendedores podrían perder sus bloqueos propietarios; JDO se solapa demasiado con EJB y CMP. A pesar de estos puntos de resistencia, muchos expertos están aconsejando JDO como una enorme mejora sobre las tendencias actuales de tecnologías objeto-a-datos y datos-a-objeto.

CLOUD COMPUTING

Con este término nos estamos refiriendo a un paradigma que permite ofrecer servicios a través de internet. Toma su nombre del concepto de internet como nube que alberga

todo y se basa en la capacidad de que la computación que un sistema informático puede llevar a cabo sea ofrecida como un servicio.

Esto quiere decir que se nos ofrece la posibilidad de que nuestro sistemas como tal (hardware y software) que debería ser mantenido y ubicado por nosotros mismos, pueda ser albergado en la "nube" con la única necesidad de dotar al mismo de las capacidades que nosotros queramos otorgarle.

Como se puede deducir de la explicación anterior, esto conlleva un ahorro bastante considerable pues utilizando este tipo de medios podemos disminuir costes de infraestructura, hardware, software, mantenimiento, etc.

"Cloud Computing" puede ofrecer mecanismos de almacenamiento en red (dropbox), aplicaciones genéricas en la red, servicios web o bien aplicaciones propias. El inconveniente que presentan es precisamente que depende del proveedor que es el que nos ofrece la posibilidad de utilizarlas así como de posibles cortes de red que se puedan producir.

Este servicio está compuesto de los siguientes componenetes:

- **Clientes** (teléfonos móviles, navegadores, ordenadores con aplicación cliente,…)
- **DataCenter** (colección de servidores, normalmente virtualizados, que pueden estar distribuidos en diversas ubicaciones)
    - **Virtualización** (todo el hardware es emulado)
    - **ParaVirtualización** (no todo el hardware es emulado, lo que permite un mejor escalado de las aplicaciones)
- **Internet**

El objetivo que persiguen los servicios ofrecidos por esta tecnología es por un lado, proporcionar un fácil acceso para aquellas empresas pequeñas que no dispongan de medios necesarios para poder llevar a cabo la tarea por su cuenta, ofrecer una alta escalabilidad y ofrecer independencia del dispositivo que se emplee para poder acceder a nuestro servicio.

MODELO DE DATOS

Enlazando con el apartado anterior podemos indicar que existen diferentes posibilidades en cuanto al modelo de datos a utilizar se refiere y como en el resto de aspectos, a la hora de optar por uno u otro dependerá de las características de las que queramos dotar a nuestra aplicación.

En nuestro caso se ha optado por un modelo de datos remoto basado en XML, puesto que así lo requiere la interacción con la tecnología App Engine.

Podemos almacenar la información en un servidor remoto, ya sea en una base de datos o un fichero XML, y leer esa información mediante consultas a la base de datos o mediante parsing del documento XML.

El almacenamiento de datos de forma remota conlleva que los datos puedan ser actualizados de un modo continuo y de forma totalmente transparente al usuario para que con el simple hecho de acceder a la aplicación, estos se descarguen y sean visibles al usuario mediante el citado parsing.

ECLIPSE

Finalmente cabe mencionar que la herramienta utilizada para realizar la parte servidora que interactuará con la aplicación del dispositivo se ha utilizado la herramienta de desarrollo eclipse.

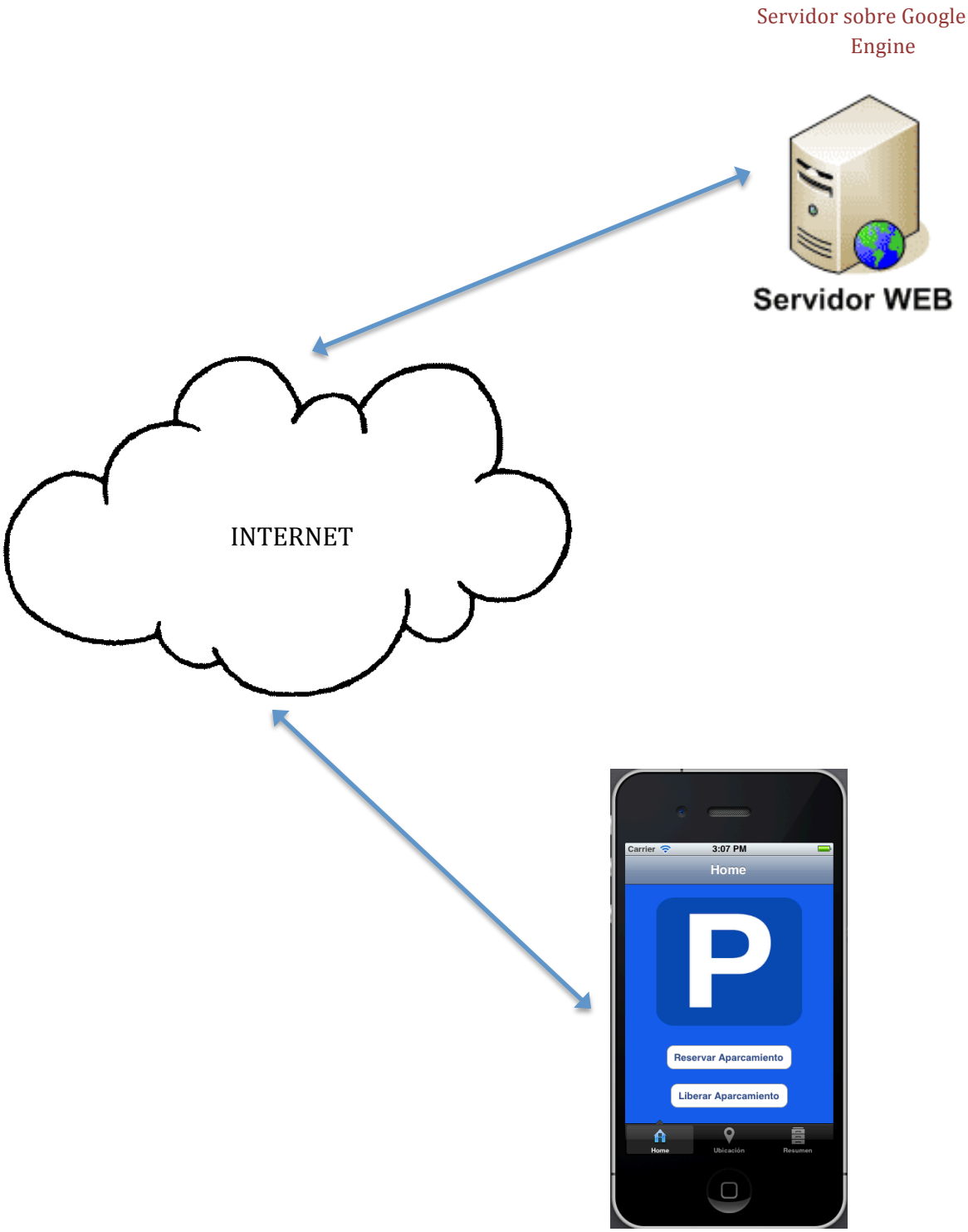
Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La arquitectura de la que se compone esta herramienta está basada en la utilización de plugins, los cuales, son los encargados de proveer toda la funcionalidad necesaria al entorno en lo más alto del sistema de ejecución (se van incluyendo los plugins necesarios en función de las características y necesidades que nuestra aplicación en desarrollo necesite en un momento determinado), en contraste con lo que ocurre con otras herramientas, que por el contrario, se basan en el concepto de hard coding, consistente en incluir toda la funcionalidad en el código fuente de la herramienta.

3. ARQUITECTURA DE LA SOLUCIÓN

El funcionamiento esta basado en una aplicación distribuida en el dispositivo móvil, el cuál, en función de la opción escogida en la misma realiza peticiones al servidor (la petición se realiza al servlet en concreto que lleva a cabo tal acción) y este cuando ha procesado la misma responde y devuelve los resultados al dispositivo telefónico.



PASOS PREVIOS

Previamente al desarrollo de nuestra aplicación es necesario realizar una serie de pasos para disponer de todas las herramientas necesarias para realización de la misma. Por tanto el cometido de este punto, es el de explicar como se realiza la instalación y configuración de las herramientas utilizadas para poder llevar a cabo el desarrollo de una forma correcta.

Para el desarrollo de la parte encargada de realizar las tareas de servidor, utilizamos la herramienta Eclipse. Para llevar a cabo la instalación de la misma, simplemente necesitamos acceder a la página web oficial de la plataforma (www.eclipse.org) y proceder a la descarga de la plataforma para poder instalarla en nuestro equipo.

Como mencionábamos anteriormente, nuestra aplicación servidora se basa en la utilización de la tecnología que Google nos ofrece por medio de Google App Engine, por lo tanto, basándonos en el concepto de que la arquitectura de Eclipse se basa en la utilización de plugins, debemos aplicar el plugin correspondiente para poder utilizar dicha tecnología. Para ello, debemos acceder al lugar de descarga de dicho plugin (https://developers.google.com/appengine/) y proceder a la descarga del mencionado plugin para la plataforma Eclipse. Una vez descargado, la instalación del mismo es bastante sencilla, pues solo tendremos que introducir en la carpeta "plugins" de la aplicación, los archivos que se han descargado del sitio oficial. Por lo tanto una vez llevados a cabo estos pasos, ya tendríamos nuestra plataforma Eclipse lista y preparada para poder llevar a cabo el desarrollo de nuestro servidor.

Para completar el desarrollo de nuestro proyecto global de aplicación, es necesario del mismo modo disponer y configurar las herramientas necesarias para poder desarrollar la parte correspondiente al desarrollo de aplicaciones iOS.

Así, lo que haremos a continuación será proceder a la descarga del SDK correspondiente para el desarrollo de aplicaciones para dispositivos iOS que se encuentra en el portal propio de Apple (App Store) de forma gratuita. Dicho SDK consta de todo lo necesario para poder llevar a cabo nuestro desarrollo por lo que únicamente nos faltaría para poder comenzar, estar dados de alta en el entorno de desarrollador de Apple (Apple Developer Program). No existe ninguna cuota de registro asociada a tal acción, sin embargo el uso simple y llano de este paso conlleva una serie de limitaciones sobre lo que podemos hacer y lo que no podemos hacer de forma gratuita. Si lo que queremos es tener acceso anticipado a las versiones beta, cargar las aplicaciones a dispositivos reales o difundir aplicaciones a través de la App Store, lo que deberemos hacer es pagar una cuota de socio.

Si se diese el caso y se toma la decisión de realizar el pago de la cuota de socio correspondiente, el programa de pago para desarrolladores ofrece dos niveles:

- Un programa estándar de 79 €, para aquellos que crearán aplicaciones que desean distribuir en la App Store.
- Un programa empresarial de unos 223 € aproximadamente, destinado a empresas grandes que quieran desarrollar y distribuir aplicaciones internamente, en lugar de hacerlo a través de la App Store.

En mi caso concreto he tenido la posibilidad de poder acceder al paquete de solución estándar para llevar a cabo mi trabajo por medio de las posibilidades que la Universidad Politécnica de Valencia ofrece como órgano universitario, a través del cual, se me ha otorgado un código de permiso para el desarrollo de la aplicación, el cual, una vez terminado el desarrollo y llevado a cabo las configuraciones necesarias (especificadas debidamente en la web de desarrollador: developer.apple.com), se encuentra listo para permitirnos la descarga de nuestra aplicación al dispositivo para poder llevar a cabo las pertinentes pruebas de funcionamiento de la aplicación.

ESPECIFICACIONES FUNCIONALES

APLICACIÓN IPHONE

Tal y como citábamos anteriormente, la aplicación ha sido concebida con la finalidad de proporcionar al usuario una herramienta que le facilite la ardua tarea de poder estacionar su vehículo. La información necesaria para poder llevar a cabo dicha labor, se ofrece de diferentes modos, ya sea por medio de una tabla o de un mapa ilustrativo.

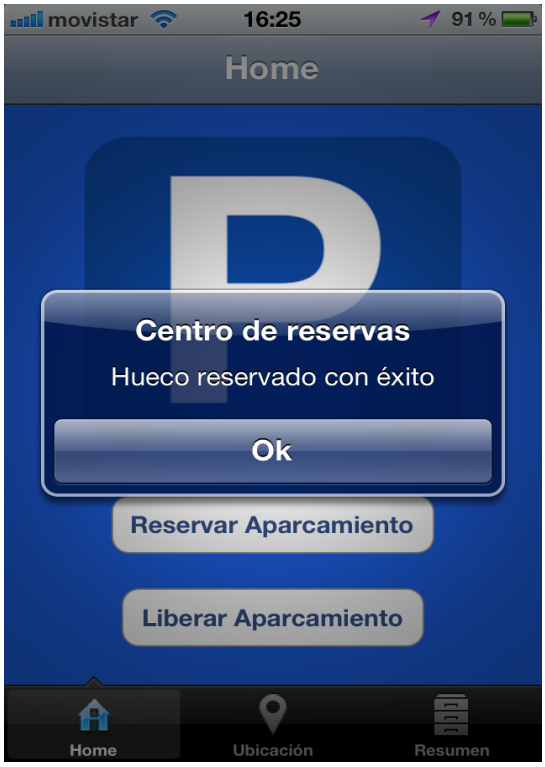
La estructura general de la aplicación está basada en una navegación por medio de una tabla de pestañas (Tab Bar) que permite identificar las tareas clave de la aplicación de un modo intuitivo, si bien, dentro de cada una de estas funciones, se ofrecen otros tipos de navegación a la hora de representar la información.

Con lo mencionado anteriormete, podemos pasar pues, a la descripción detallada del aspecto que la aplicación ofrece al usuario:

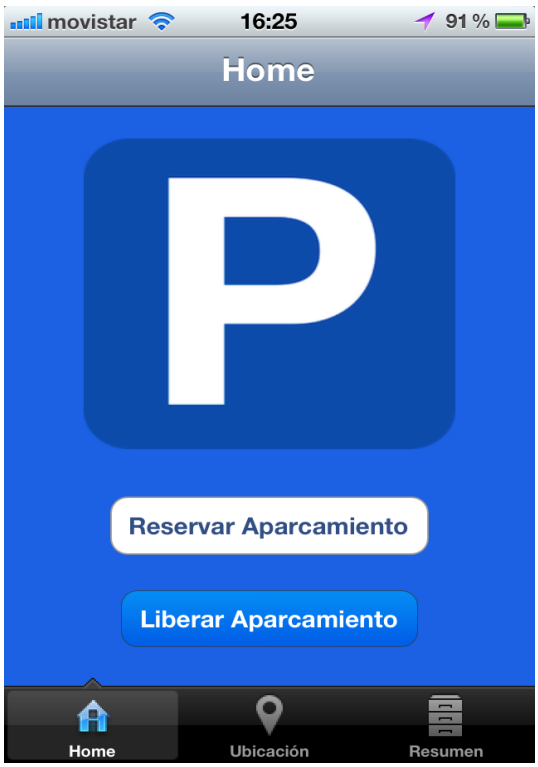
En primer lugar, la primera de las pestañas que componen la apliación, recibe el nombre de **Home**. Como su nombre indica es la primera vista que se presentará al usuario y su cometido principal es de llevar a cabo la reserva de los posibles huecos de aparcamiento, asi como a la liberación de los mismos.

Tal y como observamos en la imagen aparecen dos botones, que son los encargados de llevar a cabo las tareas de la reserva y la liberación de los huecos de estacionamiento.

Si pulsamos sobre el botón que tiene por título “Reservar Aparcamiento” estaremos indicando que hemos encontrado un hueco para estacionar nuestro vehículo que no se encuentra en los que están libres y notificados por la aplicación (alojados en la base de datos) y por lo tanto se procede a genera un nuevo “hueco” con las coordenadas actuales en las que se encuentra el usuario. Una vez llevado a cabo dicha tarea, aparecerá una alerta que nos indicará que la operación ha sido llevada a cabo satisfactoriamente.



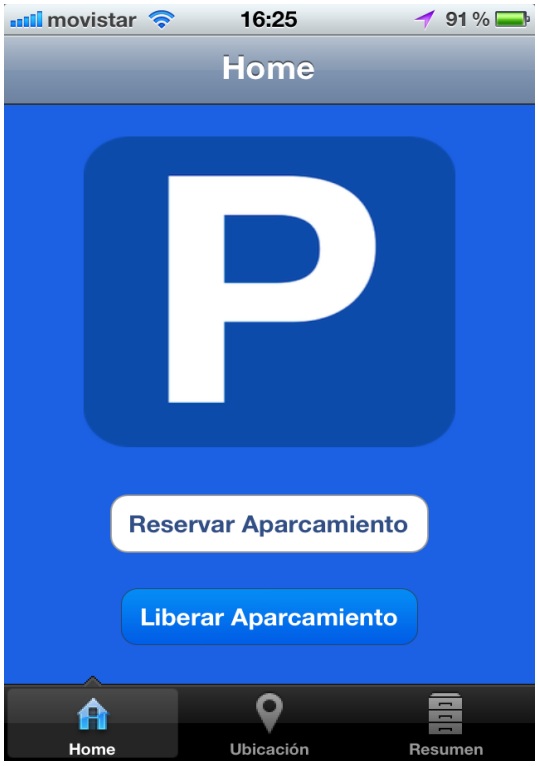
Si por el contrario, la tarea que queremos llevar a cabo es la de liberar el "hueco" de estacionamiento donde nos encontramos para notificarlo a la base de datos y que así otro usuarios puedan ocupar ese espacio libre, simplemente tendremos que pulsar el botón que se encuentra debajo del anterior y que tiene por título "Liberar aparcamiento". Una vez lo hayamos hecho, esto nos llevará a otra vista, implementada mediante la opción de navegación que nos permitirá ir hacia atrás y adelante en la jerarquía de esta pestaña, la cual nos muestra una tabla en la que aparecen todos los huecos que en ese momento determinado se encuentran ocupados.

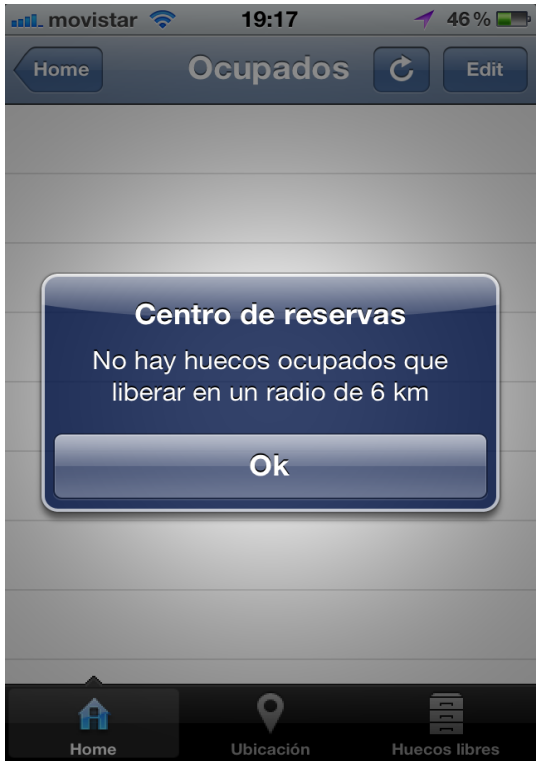


Como se puede observar en la parte superior de la tabla, aparecen diferentes opciones que nos permiten llevar a cabo otras acciones. El botón Home, nos permite volver hacia la vista principal, tal y como comentaba anteriormente. El botón de **reload** (flecha redondeada) nos permite realizar un refresco de los datos mostrados en la tabla, se conectará con el servidor y realizará un volcado de la información disponible en el mismo, con la finalidad de poder detectar cualquier modificación.

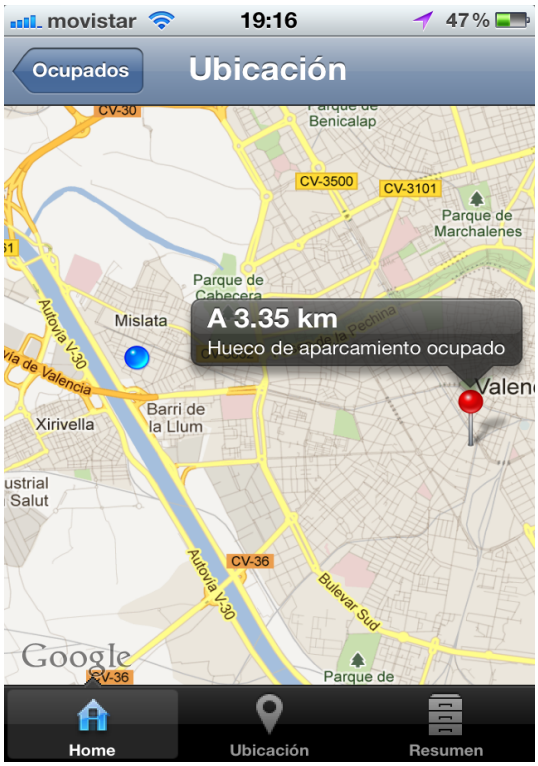
Existe también la posibilidad, debido a la configuración establecida, de que no existe ningún hueco de entre los ocupados a liberar en el perímetro establecido como límite. Esto se ha llevado a cabo del siguiente modo debido a que no resulta interesante que la aplicación muestre todo el contenido disponible en el servidor puesto que puede haber huecos que se encuentren a cientos de kilómetros de nuestra ubicación actual y por lo tanto no nos son útiles. Por esto en la configuración de la aplicación se ha llevado a cabo la realización de un filtro, una vez recibida toda la información del servidor, para que solo se muestren los huecos que se encuentran en un rango de metros que pueda resultar interesante su utilización. En nuestro caso hemos establecido 6 km, aunque se

podrá configurar. Como muestra de lo expuesto en el párrafo anterior, podemos apreciar en las siguientes imágenes:



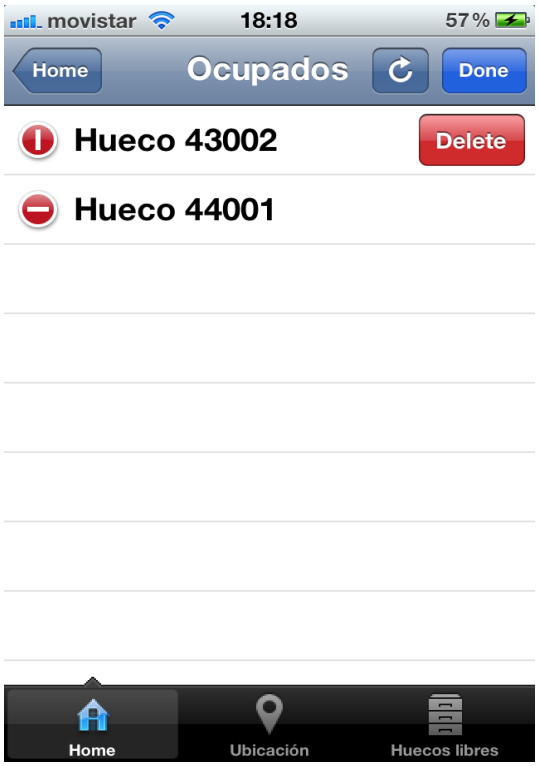


Si pulsamos sobre cualquiera de los objetos de la tabla, cada uno de los cuales hace referencia a "huecos" que en este momento están siendo utilizados, iremos a otra nueva vista en la que nos aparece un mapa indicando exactamente la ubicación de ese "hueco".



Como podemos observar en la figura anterior, el punto azul en el mapa representa nuestra ubicación actual, mientras que la anotación de color rojo representa la ubicación del hueco ocupado a liberar en cuestión. Si pulsamos sobre el icono rojo de la anotación nos aparecerá un cuadro de texto indicando a distancia a la que nos encontramos de dicho hueco.

Por último el botón **edit** nos permite poder eliminar cualquier dato de la tabla. Esta última opción no se basa simplemente en eliminar una fila de la tabla, si no que se centra en las consecuencias que ellos conlleva, es decir, si nosotros eliminamos una fila de dicha tabla, lo que estamos haciendo en cierto modo, es notificando el abandono del hueco de estacionamiento en el cuál nos encontramos y por lo tanto "liberando" nuestro hueco. Por lo tanto cuando eliminamos una fila de esta tabla lo que estamos haciendo es liberar nuestra posición de estacionamiento y generando un nuevo "hueco" libre que se notificará en la pestaña resumen que comentaremos posteriormente.



La siguiente pestaña que encontramos es la denominada como **Ubicación.** Esta pestaña es la encargada de indicarnos en el mapa la posición actual sobre la que nos encontramos.

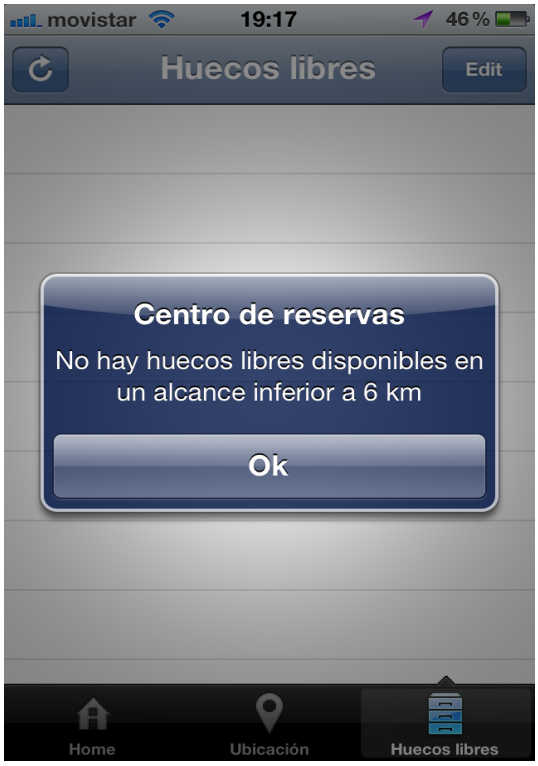
En la figura observamos un punto azul que representa la ubicación actual en la que nos encontramos. La funcionalidad del mapa incluye la opción de poder hacer zoom sobre el mismo o alejar la imagen y poder desplazarnos por el mismo para poder observar cualquier cosa que pueda resultar de nuestro interés.

Finalmente, la tercera y ultima pestaña de la que consta la aplicación, recibe el nombre de **Resumen** y su cometido es de listarnos en una tabla los "huecos" libres que se encuentran disponibles en ese momento determinado, para poder realizar la reserva de los mismos. La procedencia de estos "huecos" es, como decíamos más arriba, aquellos huecos disponibles en la tabla que se encuentra en la pestaña Home y de los cuales hemos procedido a borrarlos de la lista y por tanto liberándolos. Los datos mostrados en dicha tabla, corresponden a todos los huecos existentes en la base de datos que se encuentran a una distancia inferior a seis kilómetro de nuestra posición actual, con el propósito de evitar listar ciertos huecos que por su lejanía, nos resulten inservibles en ese momento. Automáticamente a la eliminación de ese objeto de la tabla, se realiza una llamada al servidor con los datos de ese "hueco" liberado para que se proceda a la creación de otro "hueco" de carácter libre para poder realizar la conveniente notificación de que dicho espacio queda libre para el estacionamiento.

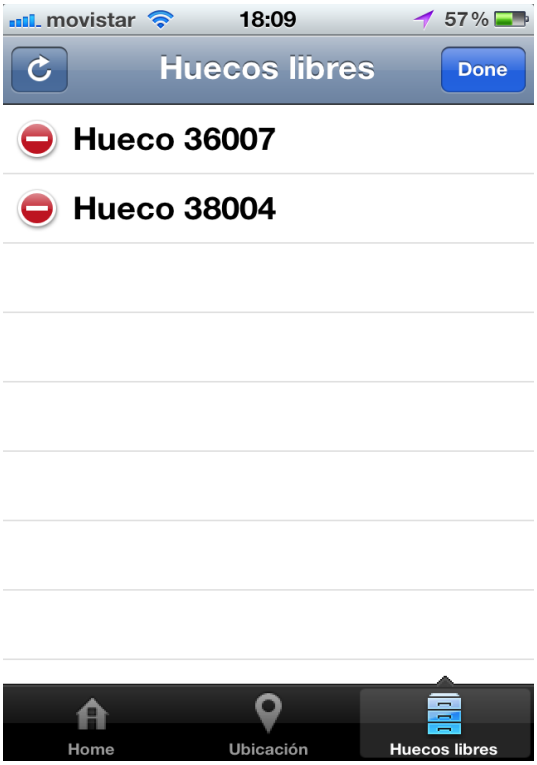
Al igual que ocurría en la pestaña **Home**, se ha incluido la posibilidad de establecer un filtro respecto a los huecos existentes en el servidor para poder dotar a la información que se muestra en la aplicación de una mayor utilidad y así poder excluir aquellos huecos que debido a la lejanía que existe de los mismos a nuestra posición actual no nos resultan de mucha utilidad.

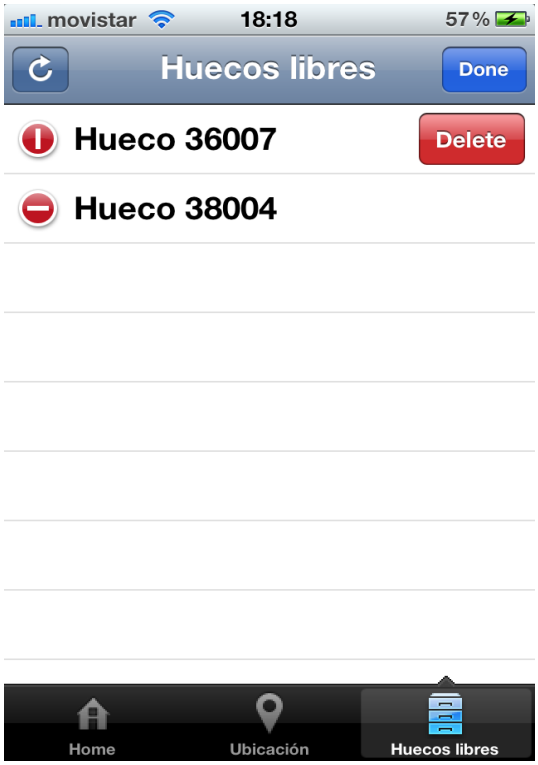
De este modo la apariencia que tendría nuestra aplicación en el caso de que esta circunstancia se diese, puede apreciarse en la figura siguiente, en la que del mismo

modo que en la primera de las pestañas, hemos establecido el límite para poder mostrar un hueco en 6 km.

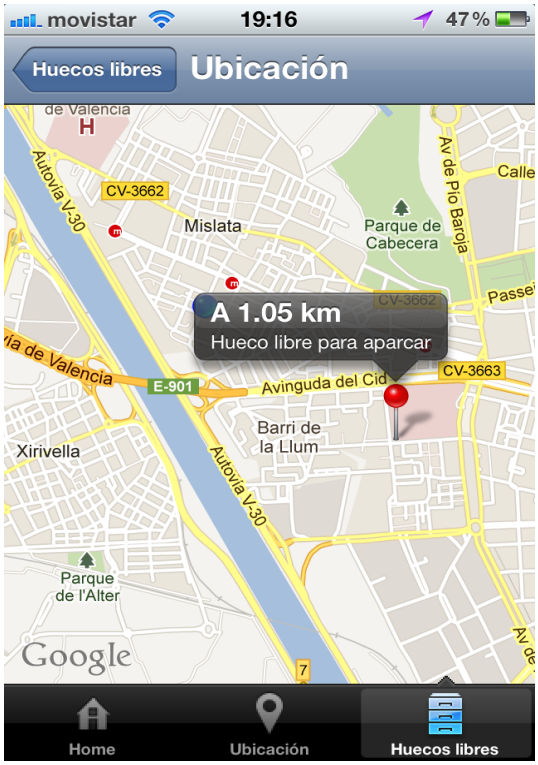


Del mismo modo que procedíamos anteriormente podemos realizar un refresco de los datos disponibles en la tabla, mediante el botón reload de la parte superior izquierda y podemos eliminar un elemento de la lista mediante el botón Edit, situado en la parte superior derecha de nuestra pantalla del dispositivo. Tal y como sucedía en la reserva de espacios de estacionamientos, la eliminación de uno de los elementos de la lista supone que el usuario ha localizado esta posición y pretende estacionar su vehículo en el mismo, por lo que procede a notificar que esa posición ya no estará libre y por lo tanto por eso procede a la eliminación de la lista. Dicha eliminación conlleva una llamada al servidor con los datos del "hueco" eliminado para crear un objeto "hueco" en el apartado de huecos ocupados.





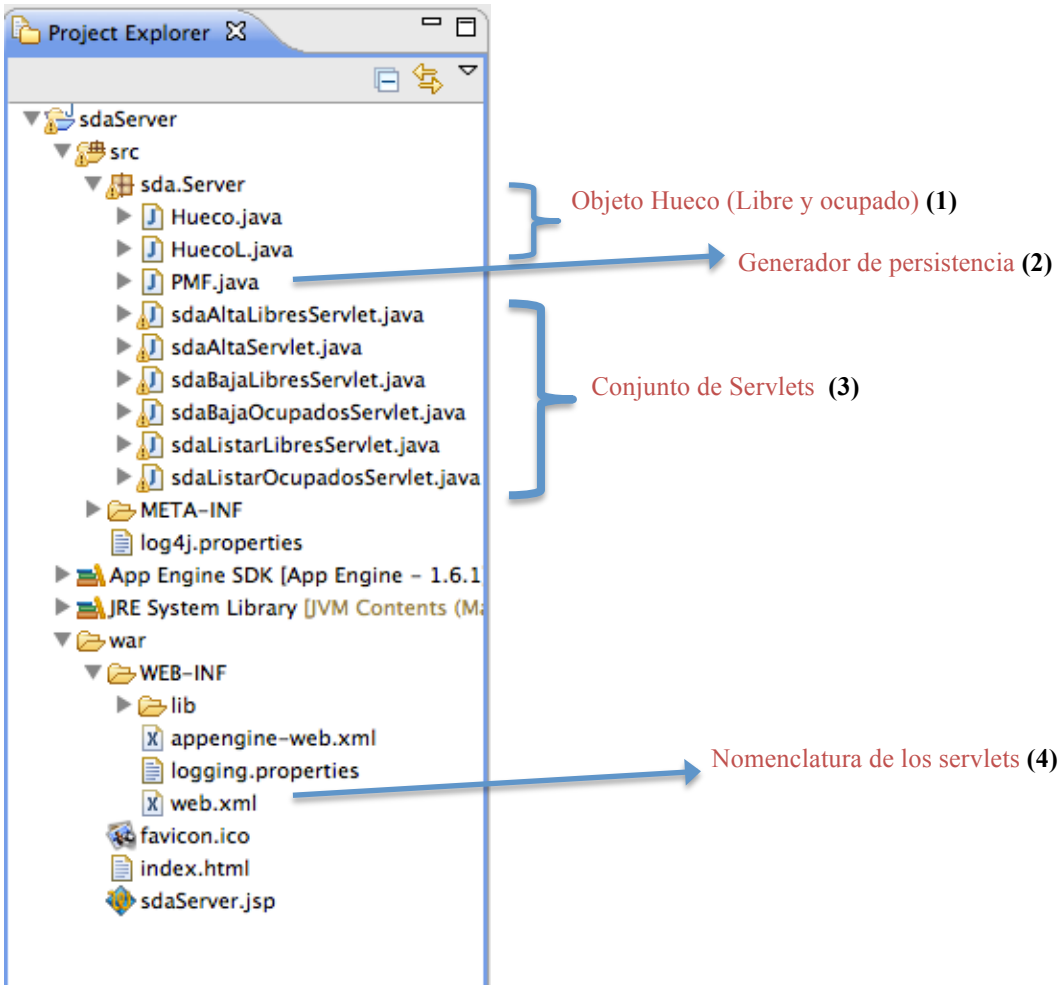
Por último, otra funcionalidad que incorpora la aplicación es poder visualizar en el mapa el lugar concreto del "hueco" listado. Para ello, tan solo debemos pulsar sobre cualquiera de los objetos de la tabla, para poder acceder a la vista en el mapa con la posición concreta del hueco. En el mapa, aparecerá una anotación de color rojo que nos indica precisamente la posición concreta.



Se puede observar el punto azul en el mapa que representa nuestra ubicación y el pin de color rojo que representa la ubicación del hueco libre que disponemos para aparcar. Así si pinchamos sobre la anotación del hueco nos aparecería un cuadro de texto indicativo, con los metros o kilómetros a los que se encuentra dicho hueco respecto a nosotros.

SERVIDOR JDO (CLOUD COMPUTING)

Al otro lado de la aplicación estaría el servidor implementado en lenguaje Java, utilizando los principios de la computación en la nube y cuyo aspecto pasamos también a detallar para que lector de la presente memoria tenga conciencia de cómo está estructurado el funcionamiento de la aplicación tanto desde la parte iPhone (descrita arriba) y la parte servidora con la que interactúa. Para ello presentamos una imagen global de la estructura del servidor de la aplicación.



Una vez vista la apariencia principal de nuestro servidor, estamos en disposición de ir detallando un poco más cada una de las partes identificadas en el mismo y que a efectos de funcionamiento para nuestra aplicación, son las más relevantes.

1) En primero lugar tenemos las dos clases que establecen los objetos Java con los que se va a trabajar en la manipulación de datos por parte del servidor. Así pues existe tanto un objeto para "huecos libres" (HuecoL.java), como para "huecos ocupados"(Hueco.java) y cuyo aspecto es similar al de cualquier objeto que hayamos creado en Java, con sus constructores, modificadores, etc.

2) El segundo bloque corresponde con la clase PMF.java. Se trata del API de Java que nos permitirá que podamos almacenar nuestros objetos de forma persistente y que podamos utilizar los mismos a modo de base de datos, pudiendo hacer consultas sobre la misma (consultas simples) y volcado de datos. Es por tanto en esta clase donde establecemos los parámetros necesarios para poder utilizar este servicio.

3) La tercera parte de las establecidas, es la más extensa y la que engloba todo el grupo de servlets que llevarán a cabo las tareas asignadas al servidor. En el interior de cada uno de los servlets, cabe mencionar, que es necesario la utilización del elemento de persistencia previamente mencionado para que las operaciones que llevemos a cabo con los objetos, puedan ser almacenadas, eliminadas o modificadas y los cambios se vean reflejados en el servidor. El resto de la lógica que contiene cada uno de ellos viene determinada por la funcionalidad del mismo. Así como vemos en la nomenclatura asignada a cada uno, resulta sencillo imaginar que es exactamente lo que hará cada uno de los servlets.

4) Por último y no menos importante cabe mencionar el archivo web.xml. Es en este archivo donde especificaremos cual será la nomenclatura necesaria para poder llamar a los diferentes servlets que componen nuestro servidor para que realicen la tarea que deseamos. Por ello, en el interior del archivo debemos especificar el nombre del servlet en cuestión y asignarle la extensión que tendrá dentro de la url del mismo. Como muestra, se ofrece una imagen que permite ver a las claras a lo que nos estamos refiriendo.

web.xml

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
<servlet>
    <servlet-name>init</servlet-name>
    <servlet-class>sda.Server.sdaAltaServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>init</servlet-name>
    <url-pattern>/init</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>initlibre</servlet-name>
    <servlet-class>sda.Server.sdaAltaLibresServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>initlibre</servlet-name>
    <url-pattern>/initlibre</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>listarocupados</servlet-name>
    <servlet-class>sda.Server.sdaListarOcupadosServlet</servlet-class>
</servlet>
<servlet-mapping>
```

Como se puede observar, escribiendo el lenguaje XML establecemos la estructura servlet mediante la expresión <servlet></servlet> en la que se indica la ruta completa de servlet en cuestión y posteriormente se establece su mapeo para poder utilizarlo a través de una URL, mediante las etiquetas <servlet-mapping></servlet-mapping>.

Este proceso habría que realizarlo para todos y cada uno de los servlets que queremos incluir en el funcionamiento de nuestra aplicación.

Aunque no lo hemos mencionado porque en el contexto de nuestra aplicación no se utiliza y no resulta útil, cabe mencionar la existencia de un archivo con extensión ".jsp" que es el encargado de realizar las funciones de mostrar el aspecto que tendrá el servidor si se le invoca desde un navegar o si queremos realizar alguna función en el mismo a través de la web. En este caso no resulta relevante porque nuestra aplicación en ningún momento ofrece una vista web del servidor y simplemente se comunica con él para poder realizar su cometido. En el caso de que si que lo hiciese, sería necesario configurar este archivo para poder ofrecer una vista depurada de nuestro servidor web.

TECNOLOGÍA EMPLEADA

Una vez hemos explicado la arquitectura de la solución propuesta, así como las funcionalidades de la misma y el modo de poder utilizarlas, es conveniente que expliquemos mas detalladamente las tecnologías utilizadas para llevarlas a cabo.

XML

Esta tecnología hace referencia al modelo de datos utilizado para llevar a cabo las comunicaciones entre el servidor y nuestra aplicación iPhone. Los datos intercambiados con el mismo en dicho lenguaje se limitan principalmente a transmitir las diferentes coordenadas (longitud, latitud) de los huecos de estacionamiento libres u ocupados residentes en el servidor así como la fecha de creación de los mismos y una breve descripción de los mismos, basada en indicar el número de identificador del "hueco" en el servidor.

Lo que se pretende es tener la información en el tiempo menor posible, debido a que la finalidad de la aplicación es que el usuario disponga en su vehículo de una forma rápida, información del lugar donde podría estacionarlo, es por ello, que hemos elegido este tipo de intercambio de datos.

Del mismo modo, la información hubiera podido ser almacenada de forma local en el dispositivo, pero con este mecanismo, tan solo dispondríamos de información propia al dispositivo y lo que se pretende es disponer de información que los usuarios de la aplicación hayan facilitado con el uso de la misma sobre los lugares de estacionamiento de los vehículos y posibles lugares donde poder estacionar el nuestro.

Por todo ello, hemos optado por escoger el intercambio de datos por medio de XML que nos permite una comunicación rápida y un manejo de la información sencillo para poder ofrecerla al usuario de una forma intuitiva.

GEOLOCALIZACIÓN

Basamos el funcionamiento de nuestra aplicación en el uso de las capacidades y características GPS de las que dispone el dispositivo, para poder identificar posiciones en el mapa y así poder ofrecer al usuario información no solamente textual, si no, que también pueda ver la misma plasmada en un mapa y facilitarle de este modo la utilización de la aplicación.

JDO (JAVA DATA OBJECT - CLOUD COMPUTING)

Como mencionábamos en la descripción de esta tecnología, es un servicio que Google nos facilita para poder aportar soluciones de hosting de una forma sencilla sin tener la necesidad de montar nuestro propio servidor. De este modo nuestra labor será construir nuestro servidor mediante la utilización de **servelets** (clase Java que puede recibir peticiones http y genera salidas normalmente en http, wml o xml y que se ejecuta en el servidor), los cuales, serán los encargados de llevar a cabo las tareas asignadas a la aplicación en la parte servidora.

Mas concretamente, en el extremo servidor de nuestra aplicación disponemos exactamente de seis **servlets** encargados respectivamente de dar de alta nuevos huecos de estacionamiento (libres y ocupados), darlos de baja y listar el contenido del servidor para poder volcarlos a la aplicación y cuya explicación se ha detallado anteriormente.

4. DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN

Este apartado se basa en establecer una vista mas técnica de la estructura de nuestra aplicación para dispositivos iPhone por medio de diagramas que indican tanto las variables de las que dispone cada una de las clases así como el modo de interactuar entre las mismas.

Pero antes, es conveniente resaltar que en la construcción de una aplicación para dispositivos iOS, el modo de generar los ficheros, tiene un punto común, pues cuando creamos un proyecto, siempre se generan unos archivos **AppDelegate.h** y **AppDelegate.m** encargados de gestionar el ciclo de vida de nuestra aplicación. Además posee los métodos que se ejecutan durante los diferentes estados de la aplicación como puede ser al inicio, al entrar en un estado de inactividad, al volver de él, etc...Por otro lado también se genera un archivo denominado **MainWindow.xib**, el cual es el encargado de gestionar la apariencia gráfica que tendrá nuestra aplicación. A partir de este diseño base, se van añadiendo las diferentes clases y archivos ".xib" para poder dar la forma y la funcionalidad deseadas a nuestra aplicación.

Es importante también el uso de los métodos delegados **didFinishLaunchWithOptions**, el cual, nos permite establecer parámetros de funcionamiento específicos de nuestra aplicación una vez se haya lanzado la misma o en casos en los que la memoria sea baja y el método **didSelectViewController**, que se encarga de gestionar la vista a mostrar en función de la opción elegida en la barra de pestañas (tabBar). En el primer caso, nuestro código se basa en establecer los parámetros necesarios para la inicialización de la vista de la barra de pestañas, mientras que en el segundo método, establecemos el código necesario para que la animación del puntero establecido para que se mueva en caso de cambio de vista se vaya desplazando de uno a otro lugar. En caso de que el comportamiento de la barra de pestañas fuese el establecido por defecto a cada transición de vista no sería necesario dotar de código este método.

De modo genérico se han definido una serie de clases que serán llamadas desde las diferentes pestañas que componen la aplicación. Estas clases reciben los nombres de: **XMLParser** (encargada de realizar la llamada al servidor para poder listar el contenido de huecos libres mediante la transcripción de código XML a objetos Objective-C y que estarán almacenados en la clase delegada para su posterior uso), **XMLParser2** (encargada del mismo cometido que la anterior pero en el caso de los huecos ocupados disponibles en el servidor), **Hueco** (que es la clase genérica que constituye un objeto de tipo huecos para poder representar la información concerniente al mismo) y la clase **MyAnnotation** (que es la clase a la que se instanciará cada vez que se desee mostrar una anotación en el mapa).

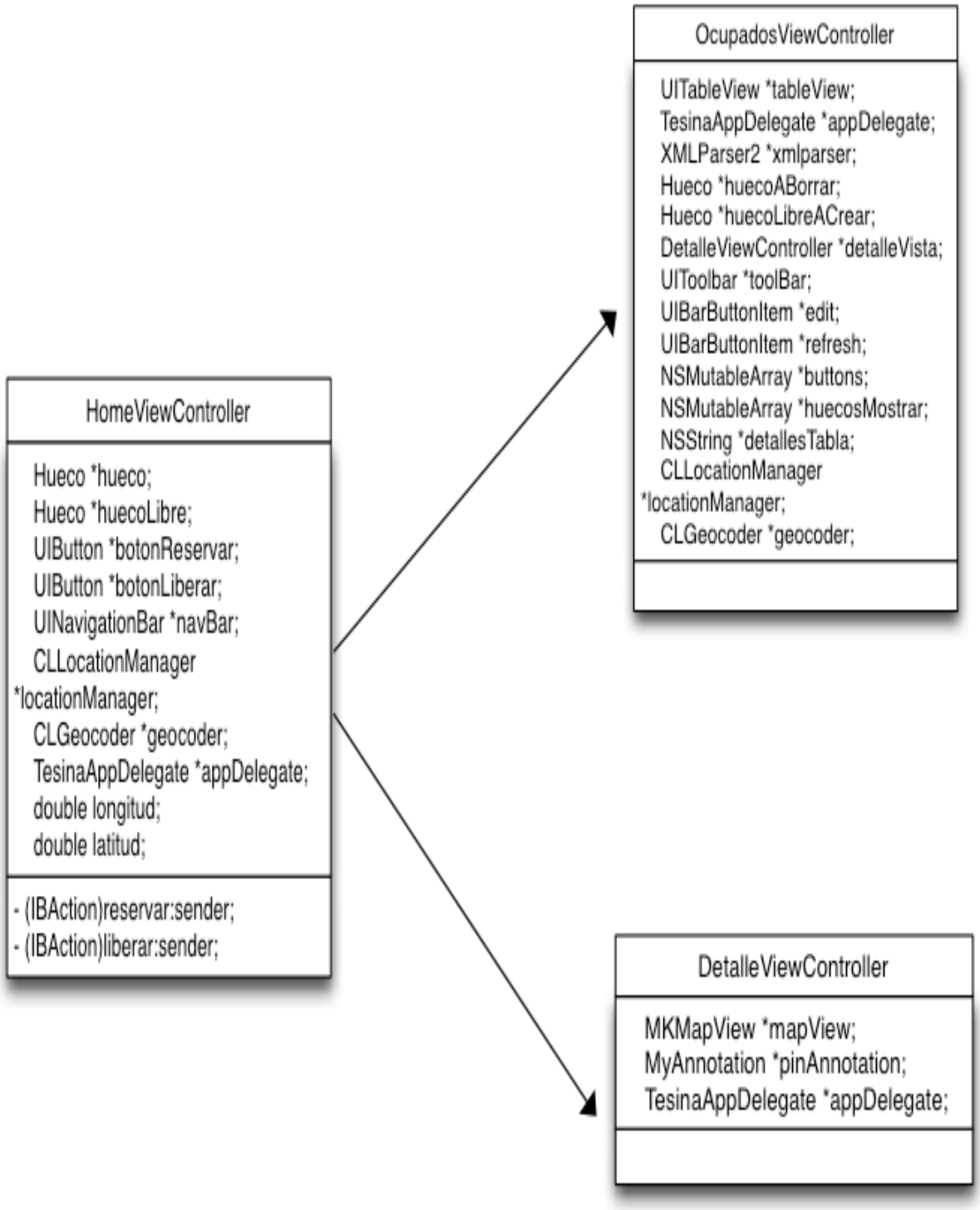
El aspecto que presenta la información que las clases "parser" realizarán es como el que se presenta en la imagen que precede, en la cuál, la información aparece representada como un archivo XML sobre el que la aplicación trabajará con tal de identificar los

diferentes huecos existentes en el servidor y sobre los que se ha realizado la consulta por medio de las diferentes etiquetas que hemos establecido previamente en el diseño del documento XML en la parte servidora.

Código fuente de http://tesinapark.appspot.com/listarocupados

```
<?xml version='1.0'?>
<huecos>
 <hueco>
  <id>27007</id>
  <latitud>39.5363</latitud>
  <longitud>-1.5949</longitud>
  <fecha>05/05/2012</fecha>
  <descripcion>Hueco con identificador: 27007</descripcion>
 </hueco>
 <hueco>
  <id>33002</id>
  <latitud>39.4728</latitud>
  <longitud>-0.419887</longitud>
  <fecha>03/05/2012</fecha>
  <descripcion>Hueco con identificador: 33002</descripcion>
 </hueco>
</huecos>
```

PESTAÑA HOME



En terminología Cocoa Touch (conjunto de frameworks utilizados por Mac OS X orientada a objetos destinados a la implementación de aplicaciones nativas), decimos que un objeto delegado es aquel al que se puede remitir desde otros objetos en cuestiones de comportamiento e informa sobre los cambios de su estado. Por lo tanto, extrapolando este funcionamiento de un modo más genérico a una clase, podemos decir que la clase delegada del proyecto es la encargada de describir el comportamiento y establecen el control del resto de clases que remiten a ella.

TesinaAppDelegate.h/TesinaAppDelegate.m, son las clases destinadas a tal fin. Como se puede observar la estructura de clases es similar a la que se utiliza en el lenguaje nativo "C", en el cual, existe un archivo que contiene las cabeceras y métodos sin implementación de la clase con extensión ".h" y por otro lado tenemos la clase con extensión ".m", la cual, se encarga de llevar a cabo la implementación propiamente dicha de la misma.

Es en esta clase donde establecemos las variables y controladores necesarios para el funcionamiento básico de nuestra aplicación, estableciendo objetos delegados que serán los encargados de describir el funcionamiento indicado y descrito más arriba.

- **UITabBarControllerDelegate** es el encargado de llevar el control de la navegación por mediación de pestañas que es la que hemos elegido como modelo de navegación principal en nuestra aplicación
- **UIApplicationDelegate** será el encargado de proporcionar información sobre los eventos claves que se puedan producir en la aplicación, como pueden ser, cuando se ha terminado de cargar una vista, cuando queda poca memoria o cuando esta a punto de terminarse.

También es necesario como paso previo, añadir a nuestra clase delegada todos aquellos "frameworks" necesarios para poder llevar a cabo las tareas que nuestra aplicación ha de llevar a cabo. En nuestro caso hablamos de los "frameworks": **UIKit**(encargado de la interfaz gráfica) o **Fundation** (encargado de proporcionar una capa base de código) que ya vienen establecidos por defecto a la hora de llevar a cabo una aplicación y de otros añadidos como **MapKit** (necesario para poder llevar a cabo las tareas relacionadas con mostrar información en el mapa) y **CoreLocation** (encargado de proporcionar los útiles necesarios para poder usar las propiedades de GPS propias del dispositivo).

A parte de las clases establecidas por defecto para el desarrollo de la aplicación son necesarias las clases que nosotros vayamos desarrollando con el fin de dotar de funcionalidad concreta a nuestra aplicación:

- **HomeViewController.h/HomeViewController.m**: Se trata de una clase que hereda de CLLocationManager. Es la clase cuyo archivo XIB, nos dará la apariencia de la aplicación en el momento de su apertura. Como detallábamos en su funcionamiento consta de dos botones específicos de acción y es en esta clase en la cual, se da código a los métodos pertinentes para llevarlas a cabo. Por un lado el método "reservar" correspondiente con el botón "realizar Reserva", se

encarga de obtener las coordenadas actuales del usuario haciendo uso del método delegado del que hereda para inicializar las variables y atributos pertinentes y realizar una llamada al módulo correspondiente del servidor para poder dar de alta y crear el nuevo hueco.
Por el contrario, el botón liberar nos realizará una llamada para poder pasar a la siguiente vista.

- **OcupadosViewController.h/OcupadosViewController.m**: Se trata de una clase de tipo UITableViewController, que tal como su nombre indica, es la encargada de establecer una vista de una tabla de datos.
  En dicha clase lo que se realiza es la llamada al "parser" encargado de hacer la lectura de los huecos de aparcamiento ocupados para poder establecer los mismos, en la tabla. Se implementan pues, los métodos delegados y necesarios para poder llevar a cabo la presencia de la tabla (número de secciones por fila, numero de filas, acción a realizar ante una edición de tabla o cuando se pulsa sobre una de las filas de la misma).
  Estos método a los que hago referencia se denominan **numberOfSectionsInTableView, numberOfRowsInSection**, **cellForRowAtIndexPath**, **commitEditingStyle** y **didSelectRowAtIndexPath** respectivamente.

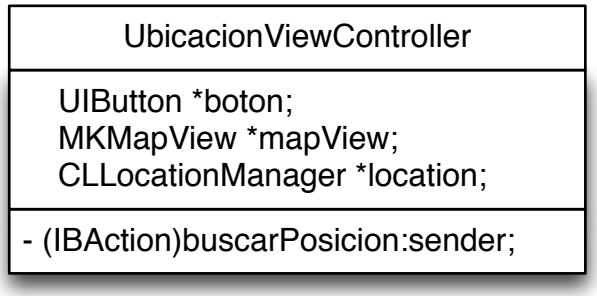
  - **numberOfSectionsInTableView** es un método en el cual se especifican cuantas secciones habrá en cada una de las celdas que componen nuestra tabla, que en nuestro caso hemos optado por establecer solo una, la correspondiente a la descripción del hueco en cuestión.
  - **numberOfRowsInSection** es el encargado de establecer el número de celdas que tendrá nuestra tabla, que en nuestro caso concreto viene establecido por el array encargado de almacenar los huecos ocupados disponibles en el servidor que se ajustan a los requisitos de limitación de distancia establecidos.
  - **cellForRowAtIndexPath** es el método delegado, encargado de establecer la información que se representa en cada una de las celdas de la tabla. Para el caso que nos ocupa, es tan sencillo como asignarle la descripción de cada uno de los huecos para cada una de las celdas.
  - **commitEditingStyle** se encarga de realizar las tareas necesarias cuando se lleva a cabo una modificación en la tabla, que en nuestro caso se limita únicamente al borrado. Cuando se borra una de las celdas de nuestra tabla estamos haciendo referencia al hecho de que estamos liberando el hueco de aparcamiento que estábamos ocupando, con lo cual, cuando hacemos dicha modificación el método se encarga de dar de baja el hueco

ocupado en el módulo del servidor, eliminar dicho hueco del array que contiene la información de la tabla para que no aparezca en la visualización de la misma y por último crear el pertinente hueco libre en el módulo correspondiente del servidor.

- **didSelectedRowAtIndexPath** es el último método de los implementados y se encarga de realizar el paso a la vista siguiente (visualización en el mapa del hueco en cuestión) cuando se pulsa sobre alguna de las celdas de la tabla, correspondiente a un hueco.

- **DetalleViewController.h/DetalleViewController.m**: Se accede a ellas cuando presionamos sobre cualquiera de las filas disponibles en la tabla. Se trata de una clase de tipo UIViewController que tiene como delegada a MapViewDelegate, encargada de realizar las tareas de visualización en el mapa de los puntos de aparcamiento, en este caso ocupados, a tenor de las coordenadas de cada uno de los huecos almacenados en el servidor y que se encuentran a una distancia inferior a los kilómetros establecidos como umbral de visualización. Para la visualización del hueco en concreto es necesaria la creación de un objeto de tipo MKAnnotation, el cual, se encargara de aparecer en el mapa a modo de una chincheta en el mismo para mostrarnos la posición del hueco al que estamos haciendo referencia. Si pulsamos sobre el, nos aparecerá información de los metros a los que nos encontramos de la posición del hueco en concreto.

PESTAÑA UBICACIÓN

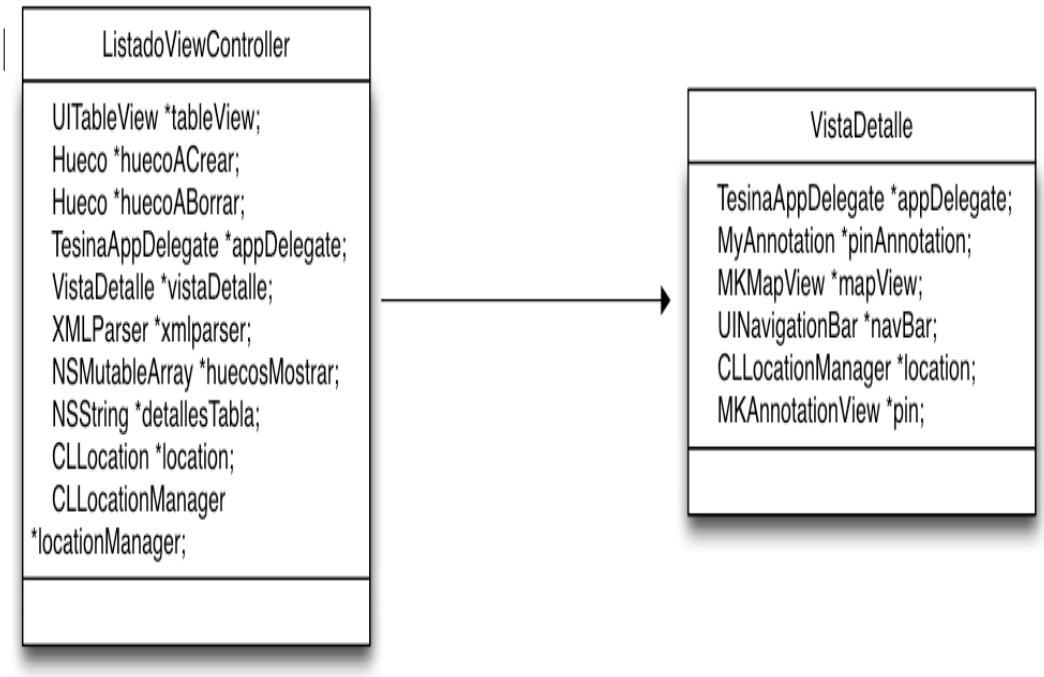


En esta pestaña tan solo hacemos uso de una clase para llevar a cabo nuestro cometido. En ella se muestra la visualización de nuestra posición acutal en el mapa.

UbicacionViewController.h/UbicacionViewController.m es la clase encargada de tal cometido. Se trata de una clase de tipo UIViewController, por lo tanto, una vista normal, pero con la salvedad que al igual que la descrita en último lugar en la pestaña "Home", hereda de MapViewDelegate, con lo cual, se encarga de visualizar nuestra posición actual en el mapa. En ella no se ha llevado a cabo ninguna implementación adicional salvo la inicializar el objeto de localización en el método encargado de procesar la carga de la vista de la clase (viewDidLoad).

En este caso se da código al método delegado **didUpdateToLocation**, el cual, se encarga de realizar las acciones pertinentes cuando se inicializa al objeto "locationManager". Concretamente se establece el rango de visualización en el mapa y los parámetros necesarios para la visualización.

PESTAÑA HUECOS LIBRES



La tercera y última pestaña era la encargada de mostrarnos los huecos libres para poder estacionar el vehículo, que se encuentran almacenados en el servidor y que cumplen los requisitos, en cuanto a distancia a nuestra posición actual se refiere, de posición.

En primera instancia nos aparece una vista con una tabla en la cual se listan los huecos libres almacenados en el servidor y que de nuevo cumplen los requisitos de restricción de distancia respecto a nuestra posición actual establecidas. Si pulsamos sobre cualquiera de las celdas, accederemos a una vista de mapa en la que observaremos exactamente la posición en el mapa de hueco seleccionado. Se trata por lo tanto, de un funcionamiento similar al que podemos encontrar en la primera pestaña.

ListadoViewController.h/ListadoViewController.m son clases de tipo **UITableViewController**, por lo tanto son las encargadas de realizar la visualización de la tabla correspondiente a los huecos libres. Esta clase hereda de **CLLocationManagerDelegate**, con motivo de inicializar el objeto de este tipo para establecer la distancia entre nuestra posición actual y la del hueco disponible en el servidor. Del mismo modo que hacíamos en la pestaña "Home", se implementan una serie de método delegados correspondientes a **UITableView**.

- **numberOfSectionsInTableView** es un método en el cual se especifican cuantas secciones habrá en cada una de las celdas que componen nuestra tabla, que en nuestro caso hemos optado por establecer solo una, la correspondiente a la descripción del hueco en cuestión.
- **numberOfRowsInSection** es el encargado de establecer el número de celdas que tendrá nuestra tabla, que en nuestro caso concreto viene establecido por el array encargado de almacenar los huecos libres disponibles en el servidor que se ajustan a los requisitos de limitación de distancia establecidos.
- **cellForRowAtIndexPath** es el método delegado, encargado de establecer la información que se representa en cada una de las celdas de la tabla. Para el caso que nos ocupa, es tan sencillo como asignarle las descripción de cada uno de los huecos para cada una de las celdas.
- **commitEditingStyle** se encarga de realizar las tareas necesarias cuando se lleva a cabo una modificación en la tabla, que en nuestro caso se limita únicamente al borrado. Cuando se borra una de las celdas de nuestra tabla estamos haciendo referencia al hecho de que estamos ocupando el hueco de aparcamiento que estaba libre, con lo cual, cuando hacemos dicha modificación el método se encarga de dar de baja el hueco libre en el módulo del servidor, eliminar dicho hueco del array que contiene la información de la tabla para que no aparezca en la visualización

de la misma y por último crear el pertinente hueco ocupado en el módulo correspondiente del servidor.

- **didSelectedRowAtIndexPath** es el último método de los implementados y se encarga de realizar el paso a la vista siguiente (visualización en el mapa del hueco en cuestión) cuando se pulsa sobre alguna de las celdas de la tabla, correspondiente a un hueco.

VistaDetalle.h/VistaDetalle.m finalmente son las clases de tipo UIViewController que heredan de MapViewDelegate, encargadas de realizar la visualización del punto exacto donde se encuentra el hueco seleccionado, mediante la creación de un objeto **MKAnnotation**, el cual, aparecerá en la vista del mapa como una chincheta de color rojo, de tal modo, que si pulsamos sobre ella, nos aparecerá información exacta sobre a que distancia nos encontramos de la posición del hueco en cuestión.

5. DISEÑO E IMPLEMENTACIÓN DEL SERVIDOR

Una vez repasado el apartado técnico en lo que al aspecto de la aplicación en sí se refiere, lo que nos ocupa a continuación es establecer del mismo modo las bases y explicaciones técnicas referentes al desarrollo del servidor.

Como hemos mencionado a lo largo de esta memoria, el desarrollo del servidor se basa en utilizar la tecnología que Google nos brinda por medio de Google App Engine para establecer un servidor que haga las veces de base de datos para nosotros poder almacenar y disponer de la información en el momento que nos sea preciso y poder establecer una comunicación entre el mismo y la aplicación en sí.

El desarrollo está basado en lenguaje Java Data Object y simplemente es necesario para este caso observar los métodos necesarios y librerías a importar para poder utilizar esta metodología puesto que los fundamentos y el modo de proceder es exactamente igual al experimentado durante el desarrollo de la carrera universitaria.

Como último apunte, cabe mencionar, que cada uno de los funcionamientos específicos de los que dispondrá nuestro servidor, se desarrolla en un "servlet" diferente, entendiendo por "servlet" a la función Java que desarrolla un cometido en concreto.

CLASE HUECO Y HUECOL

El desarrollo de estas clases corresponde con la creación de los objetos Java que son con los que interactuaremos a la hora de almacenar o borrar los mismos. La creación de estos objetos no conlleva mayor modificación que la que haríamos con cualquier objeto que queramos crear en lenguaje Java, es decir, un método o métodos constructores para el objeto, una serie de modificadores para cada uno de los atributos y una serie de métodos para obtener el valor de dichos parámetros. La única modificación que hay que llevar a cabo es la de establecer una serie de características a la hora de declarar los atributos que ha de tener el objeto.

El objetivo a la hora de crear nuestros objetos es que estos sean persistentes para poder ser almacenados. El "dataStore" que nos proporciona Google App Engine, trabaja con entidades, mientras que JDO lo hace con objetos persistentes por lo que tendremos que hacer la conversión de los mismos mediante anotaciones y precisamente estas anotaciones constituyen las diferencias de los objetos declarados con un objeto normal de Java.

```
import javax.jdo.annotations.IdGeneratorStrategy;

@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Hueco {

    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Long id;

    @Persistent
    private double longitud;

    @Persistent
    private double latitud;

    @Persistent
    private String fecha;

    @Persistent
    private String descriptor;
```

La directiva **@PersistentCapable** indica que una clase puede ser almacenada y recuperada mediante el soporte JDO. Es por este motivo por el que hacemos uso de esta clausula antes de la declaración de la clase para que podamos utilizar la misma como un objeto JDO y poder tratar con él en el "dataStore".

El siguiente elemento novedoso que encontramos es **@PrimaryKey** que constituye la clave primaria de la clase. Se refiere al hecho que para cada objeto que nosotros creemos de este tipo, tendrá un identificador que lo hará único en el servidor para poder tratar con él. En nuestro caso se ha optado por un identificador numérico de tipo "Long" que facilitará el trato y la representación de cada uno de los objetos hueco. Esta clausula irá obligatoriamente acompañada de la directiva **@Persistent**, puesto que también se almacenará.

Precisamente la directiva **@Persistent** constituye el elemento necesario para poder almacenar dicho atributo perteneciente a la entidad en el "dataStore" y supone la última modificación reseñable de esta clase.

CLASE PMF

Se trata de la clase que nos proporcionará la opción de poder operar con los objetos persistentes almacenados en el servidor. En nuestro caso simplemente necesitamos realizar la instancia del mismo para poder utilizar dichas operaciones. Con todo, este es el aspecto que presenta la clase.

```java
package sda.Server;

import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;

public final class PMF {

    private static final PersistenceManagerFactory pmfInstance = JDOHelper.getPersistenceManagerFactory("transactions-optional");

    private PMF() {}

    public static PersistenceManagerFactory get()
    {
        return pmfInstance;
    }
}
```

Como se puede observar, se crea para la clase PMF, un atributo “pmfinstance”, en el cual, se realiza la pertinente llamada al gestor de persistencia y creamos el método que nos devolverá dicho atributo.

SERVLETS PARA INICIAR UN HUECO

Los “servlets” que tiene por objetivo crear un hueco, ya sea libre o ocupado, para que sea persistente en el servidor se basan en aceptar peticiones al método **post** del “servlet”, recogiendo los datos de la petición e instanciando al constructor pertinente del objeto hueco y así, una vez creado el mismo, poder dotarlo de la persistencia correspondiente. El aspecto que presenta es el que se puede apreciar en la figura que se presenta a continuación.

```java
public class sdaAltaLibresServlet extends HttpServlet {

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException
    {
        PersistenceManager pmf = PMF.get().getPersistenceManager();

        //Extraigo en cada petición la longitud, latitud y la fecha de la misma para inicializar
        //un objeto hueco. En el caso de la latitud y la longitud obtengo su contenido String
        //y lo paso a double que es el tipo primitivo de las variables que componen el objeto hueco
                                                                    necesaria la transformación para
        //poder instanciar al constructor de la clase hueco
        String lon = req.getParameter("longitud");
        double longitud = Double.valueOf(lon).doubleValue();
        String lat = req.getParameter("latitud");
        double latitud = Double.valueOf(lat).doubleValue();
        String fecha = req.getParameter("fecha");

                                                                    la petición y creo un vector que almacena
        //todos los huecos que se van creando
        HuecoL h = new HuecoL(longitud,latitud,fecha);

        {
            pmf.makePersistent(h);
        }
        {
            pmf.close();
        }

        //resp.sendRedirect("/sdaServer.jsp");
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException
    {
        doPost(req,resp);
    }
```

Como vemos en la zona marcada mediante el rectángulo lo que hacemos es una vez se ha llamada al método post, directamente o redirigido por el método get, se capturan los parámetros de la petición por medio de la instrucción **getParameter**, correspondiente a la variable "req" que es de tipo HttpServletRequest (variable que almacena los parámetros de la petición recibida por el servidor) y se van almacenado los valores correspondientes en consonancia a los parámetros que definen cada uno de los huecos que vamos a ir almacenando en nuestro servidor.

Finalmente como se observa en la zona marcada mediante un círculo, apreciamos como haciendo uso del objeto de persistencia creado previamente (pmf), utilizamos su método **makePersistent** para hacer que nuestro servidor haga el objeto creado persistente en el datastore.

El ejemplo representado hace referencia a la creación de un hueco libre. El aspecto que presenta el servlet encargado de inicializar huecos ocupados tiene el mismo aspecto, cambiando únicamente el constructor al que se invoca, una vez extraídos los parámetros de la petición.

SERVLETS PARA ELIMINAR UN HUECO

En estos servlets lo que se pretende es que a través del identificador de cada uno de los huecos, tanto libres como ocupados, se pueda eliminar el hueco en concreto. Esto quiere decir que en cada petición al servidor, debemos pasar el identificador que habíamos definido mediante la claúsula @primaryKey, y a través del mismo proceder a la eliminación de forma persistente del datastore. La apariencia que tendrán nuestros servlets encargados de la eliminación es la que se presenta.

```
public class sdaBajaOcupadosServlet extends HttpServlet{

    Long id;

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException
    {

        PersistenceManager pm = PMF.get().getPersistenceManager();

        String id_cadena = req.getParameter("id");
        id = Long.valueOf(id_cadena).longValue();

        Hueco h = pm.getObjectById(Hueco.class, id);

        pm.deletePersistent(h);

        pm.close();
        //resp.sendRedirect("/sdaServer.jsp");

/*

        try
        {

            //updateXMLContent(Long.parseLong(id_cadena));

        }
        catch (Exception e)
        {

        }*/
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException
    {
        doPost(req,resp);
    }
```

Como observamos en la figura, en primer lugar se crea una instancia del objeto de persistencia, para posteriormente capturar el id del mismo modo que capturábamos los parámetros de las peticiones de alta de huecos, mediante el método **getParameter()** y una vez tenemos el id, único para cada hueco almacenado en el servidor, debemos hacer una consulta al mismo para obtener el hueco en cuestión (mediante el método **getObjectById**) y su posterior eliminación (con el método **deletePersistent**, que eliminará de forma persistente el hueco del datastore).

Del mismo modo que en el caso anterior, el ejemplo expuesto en la figura se corresponde con la llamada al servlet encargado de la eliminación de huecos ocupados, por lo que el código correspondiente a los huecos ocupados presenta el mismo aspecto, pero haciendo referencia lógicamente a ese tipo de hueco.

SERVLETS DE LISTADO DE CONTENIDO

Por último, estos son los servlets encargados de mostrar el contenido de la aplicación, ya sea en el navegador o bien invocado por la propia aplicación que hemos diseñado.

Básicamente lo que se realiza en ellos es una petición de consulta sobre el servidor para obtener una serie de objetos que se ajusten a un patrón de búsqueda simple (no se puede hacer una petición compleja en la que seleccionar varios parámetros para poder obtener los datos) para posteriormente establecer la estructura XML que tendrán los datos obtenidos en dicha consulta y redirigir los mismos a la salida estándar para que en el momento en que sean invocados por el navegador, la información se vea plasmada en el mismo y en el caso de que sean invocados desde la aplicación obtener la información en formato XML. Por todo lo mencionado el aspecto que presentarán estos servlets es el que podemos observar en la figura que se muestra a continuación.

```
public class sdaListarLibresServlet extends HttpServlet {

    private Query query;
    //private String xml;

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException
    {
        PersistenceManager pm = PMF.get().getPersistenceManager();
        resp.setContentType("text/xml; charset = UTF-16");
        PrintWriter out = resp.getWriter();

        query = pm.newQuery(HuecoL.class);
        List<HuecoL> huecosAparc = new ArrayList<HuecoL>();
        huecosAparc = (List<HuecoL>) (query.execute());//Descargo todas las ocurrencias relacionadas huecos libres

        //Construyo el xml que pasaré a la aplicación
        out.println("<?xml version='1.0'?>");
        out.println("<huecos>");

        for (HuecoL h : huecosAparc)
        {

            out.println(" <hueco>");
            out.println("  <id>"+h.getId()+"</id>");
            out.println("  <latitud>"+h.getLati()+"</latitud>");
            out.println("  <longitud>"+h.getLongi()+"</longitud>");
            out.println("  <fecha>"+h.getFecha()+"</fecha>");
            out.println("  <descripcion>"+h.getDescriptor(h)+"</descripcion>");
            out.println(" </hueco>");

        }
        out.println("</huecos>");

    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException
    {
        doPost(req,resp);
    }
```

Como observamos, el procedimiento seguido se basa en realizar una instancia del gestor de persistencia, para posteriormente realizar una llamada al servidor en modo consulta (query) indicando sobre que objeto queremos realizar la consulta (nos devolverá todos aquellos objetos que coincidan con ese patrón simple) y posteriormente nosotros iremos conformando la estructura de nuestro documento XML a partir de todos los objetos recibidos en la consulta.

6. PRUEBAS

El procedimiento llevado a cabo para poder hacer las comprobaciones correspondientes al uso de la aplicación, han sido las siguientes:

- Por un lado una vez desarrollado el servidor, se ha optado por realizar las pruebas del funcionamiento de cada uno de los servlets mediante el uso de un archivo ".jsp" que contenía ciertas instrucciones que realizaba llamadas al servlet que se estuviese comprobando en ese momento. Una vez que la llamada al servlet se producía mediante el navegador web, el servidor redirigía su acción al archivo "jsp" y realizaba las acciones pertinentes, comprobando que éstas se llevaban a cabo de un modo correcto.

- Con la certeza del correcto funcionamiento del servidor, se procedió a realizar las pruebas correspondientes a la aplicación. En primer lugar se utilizó el simulador disponible en la herramienta de desarrollo (Xcode), comprobando que las acciones que tenía que llevar a cabo de realizaban de forma correcta. Una vez comprobado esto, se pasó a descargar la aplicación a mi dispositivo móvil, para poder probar la aplicación de un modo real, comprobando que las se verificaba el comportamiento que le aplicación había tenido en su ejecución en el simulador. Como prueba del correcto funcionamiento de la aplicación mencionar que las capturas de pantalla presentadas en el apartado correspondiente a la descripción del funcionamiento de la aplicación se han realizado desde el dispositivo móvil en diferentes ubicaciones, constatando que efectivamente lo plasmado en el código se ejecuta de un modo correcto.

7. CONCLUSIONES

En el desarrollo de esta tarea de tesina de máster se ha llevado a cabo la realización de una aplicación distribuida para dispositivos iOS utilizando las técnicas de posicionamiento disponibles en los dispositivos, combinado con el uso de los servicios de Cloud Computing disponibles en la actualidad, concretamente proporcionados por Google App Engine. Permite obtener en tiempo real datos correspondientes a ubicaciones de aparcamiento.

El propósito de la aplicación como se ya se ha mencionado con anterioridad es utilizar esta aplicación de un modo colaborativo entre los usuarios que dispongan de la misma, con el fin de ofrecer información sobre posibles huecos de estacionamiento para nuestros vehículos, puesto que esta tarea es un poco complicada en las grandes ciudades.

Por otra parte, el desarrollo de dicha aplicación me ha proporcionado conocimientos en diferentes e interesantes materias que pueden ser útiles en un posible futuro profesional.

Las ventajas que aporta la utilización de la aplicación desarrollada van destinadas principalmente a facilitar las tareas de estacionamiento de vehículos. Si usamos esta aplicación podemos ir directamente al lugar donde se encuentra el hueco disponible para estacionar y ahorrar mucho tiempo a la hora de poder encontrarlo.

Del mismo modo que ahorramos tiempo, también podemos ahorrar combustible acudiendo directamente y evitando rodeos innecesarios.

DESARROLLO

En cuanto a la labor de la aplicación propiamente dicha, lo más complicado y lo que más tiempo me ha requerido invertir es el aprendizaje del lenguaje. Dicha labor se ha visto facilitada mediante la utilización de libros destinados a tal fin, que desde un principio especifican que en un principio puede resultar una tarea complicada pero poco a poco y con la práctica se va adquiriendo experiencia y comprensión en el desarrollo de aplicaciones.

También han resultado de gran ayuda los numerosos ejemplos y tutoriales disponibles en la web de las diferentes futuros desarrolladores, que en momentos de dudas concretas han servido de gran ayuda.

Una vez finalizado el desarrollo de la aplicación considero que he obtenido la experiencia base necesaria para poder desarrollar aplicaciones para dispositivos móviles con sistema operativo iOS, a parte, de que ha establecido una gran satisfacción al ver plasmados sobre la pantalla del dispositivo las ideas que estaban presentes para poder llevar a cabo la implementación.

A su vez se ha adquirido experiencia en el desarrollo de una tecnología que está en evidente auge como es el desarrollo en "Cloud Computing". Esto permite y dota a la aplicación de gran funcionalidad, además de hacer que personalmente tenga la experiencia de desarrollo para este tipo de tecnología que tanta importancia está teniendo en la actualidad.

Como conclusión, podemos decir que una vez finalizado el desarrollo, el aprendizaje no requiere de un esfuerzo excesivo, pues los fundamentos básicos de programación son de gran utilidad en el desarrollo. Tan solo, es necesario comprender la sintaxis y el funcionamiento del lenguaje para poder plasmar prácticamente cualquier idea que se pretenda desarrollar.

DIFERENTES SISTEMAS

A la hora de decidir el sistema operativo móvil en el cuál llevar a cabo el desarrollo de la aplicación, se ha realizado una análisis de las diferentes tecnologías existentes en el mercado en la actualidad, a parte de haber considerado los diferentes medios de los que dispongo para poder llevar a cabo la tarea.

Por la pertenencia de dispositivos correspondientes a la marca Apple y debido al análisis llevado a cabo que determina que Apple goza de un amplio sector de mercado y de una amplia gama de aplicaciones, he determinado elegir el desarrollo para dispositivos iOS para llevar a cabo la aplicación que se presenta como proyecto de tesina de máster.

En cuanto a la decisión de llevar a cabo el desarrollo de la parte servidora por medio de la tecnología ofrecida por Google App Engine, hay que decir que el factor principal que me hizo decantarme por esta tecnología es, a parte de que nos da soporte tanto software como hardware sin necesidad de que tengamos que disponer de la infraestructura para albergar el mismo y llevar a cabo su mantenimiento, es la importancia que en la actualidad está teniendo el uso de "Cloud Computing" entre la comunidad informática, ya que cada vez más podemos observar ejemplos de aplicaciones que principalmente tienen espacios de almacenamiento en la nube, permitiendo a los usuarios disponer de un lugar en la nube para poder llevar a cabo sus tareas disponiendo del contenido en cualquier lugar, sin más que disponer de una conexión a la red. De este modo nuestra aplicación estaría disponible el cualquier lugar del mundo sin más que una conexión a la red.

De todos modos, es preciso indicar, que desde un punto de vista comercial es conveniente no cerrarse tan solo en un sistema operativo en concreto para poder tener más posibilidades en el mercado laboral.

MERCADO LABORAL

Al hilo del final del punto anterior, opino que el mercado laboral en cuanto a dispositivos móviles se refiere, goza en este momento de un auge que confiere a la aplicación de un punto de vista bastante práctico y acertado en estos momentos.

La amplia gama de "smartpohones" y la amplia demanda de dichos dispositivos y más concretamente en lo que se refiere a dispositivos iOS, hace que las posibilidades de los conocimientos adquiridos se vean incrementadas en un alto grado.

CONOCIMIENTOS

El desarrollo de la aplicación me ha llevado a tener que aprender sobre un nuevo lenguaje de programación, Objective-C, y más concretamente la sintaxis del mismo, puesto que los fundamentos de programación se asemejan al del resto de lenguajes.

El entorno ofrecido por la plataforma hace que el desarrollador se sienta bastante integrado en la herramienta, a la vez que el manejo de la misma resulta sencillo para poder llevar a cabo las diferentes tareas de las que se quiere dotar a la aplicación.

8. TRABAJO FUTURO¿?

En el desarrollo de la aplicación nos hemos encontrado con ciertos problemas que no han facilitado la labor de llevar a cabo el desarrollo de la misma.

El principal problema que he tenido en el desarrollo ha sido de parte del servidor. Hemos observado que las consultas que se realizan a la base de datos que Google nos proporciona mediante el uso de la tecnología Google App Engine, tan solo se pueden realizar de modo simple, es decir, no se pueden llevar a cabo consultas complejas en las que poder seleccionar diferentes parámetros para establecer un filtrado de los parámetros a mostrar, por lo que ha sido necesario realizar una selección de los datos de un modo interno en el código de la aplicación una vez éstos han sido descargados del servidor.

Por otro lado es conveniente mencionar las diferentes modificaciones que pueden llevarse a cabo de la misma para poder dotar a la aplicación de un carácter más práctico, todavía si cabe. Podemos utilizar el sistema de integración con cuentas de Google que incorpora Google App Engine para poder utilizar datos del usuario que en un momento determinado esta utilizando la aplicación, para poder asociar los datos de un hueco determinado con los de un individuo determinado.

Otro punto a tener en consideración para una posible mejora en el funcionamiento de la aplicación es la incorporación del uso de “notificaciones push” en la misma para que el usuario disponga de mensajes e información que se está produciendo en tiempo real. Por medio de esta herramienta, podemos suscribirnos a un canal de información, que en nuestro caso sería la cantidad de huecos libres disponible en el servidor, que nos irá ofreciendo puntualmente notificaciones ante posibles cambios en el servidor, en cuanto a huecos libres almacenados en el mismo se refiere.

Del mismo modo, puesto que hemos establecido un filtro para mostrar únicamente aquellos huecos que por la distancia a la que se encuentran de nuestra posición pueden o no pueden ser mostrados en la aplicación, podemos dotar a la aplicación de funcionalidad para aparecer en el menú de ajustes de nuestro terminal para que el usuario ajuste, dependiendo de sus deseos el grado de precisión del filtro a aplicar a la información recibida del servidor, dotando a nuestra aplicación de una mayor interacción con el usuario puesto que él es en todo momento el que toma las decisiones sobre lo que desea o no desea mostrar.

9. BIBLIOGRAFÍA

1. **Desarrollo de aplicaciones para iPhone & iPad**, Joe Conway, Aaron Hillegass. ISBN: 978-84-415-2932-8 (2011)

2. **Desarrollo de aplicaciones para iPhone,** John Ray, Sean Jhonson. ISBN: 978-84-415-2795-9 (2011)

3. **Stackoverflow,** www.stackoverflow.com

4. **iPhone Dev SDK,** www.iphonedevsdk.com/forum

5. **Apple Documentation,** developer.apple.com

6. **Imaginaformacion,** www.imaginaformacion.com/tutoriales

7. **AdictosAlTrabajo,** www.adictosaltrabajo.com/tutoriales

8. **Google App Engine**, developers.google.com/appengine

9. **Java Data Objects**, db.apache.org/jdo/pm.html

10. **Rose India**, www.roseindia.net/servlets