Universidad Rey Juan Carlos

Escuela Superior de

Ingeniería de Telecomunicación

TRABAJO DE FIN DE MASTER

Diferencias técnicas en el desarrollo de aplicaciones móviles en Android e iOS

Autor: Francisco Buitrago Pavón

Tutor: Gregorio Robles

Julio, 2013

Agradecimientos

En primer lugar a mis padres, Julio y Agustina; porque sin ellos no sería lo que soy ahora, me han dado una buena educación, me han apoyado y siempre han estado allí cuando los he necesitado, han sabido cuando presionarme y en los momentos más difíciles darme ánimos. A mi hermano Julio, sin su compresión y su ayuda esto no habría posible, siempre cuidándome, siempre dándome aliento y sin sus consejos no habría llegado a ser la persona que soy ahora.

A mi familia, por estar siempre pendiente de mi, por preocuparse, por quererme, por día a día hacerme ser mejor persona.

A mis amigos, los más especiales, los que siempre han estado en las sombras, ocultos, apoyándome en cada momento, levantándome en cada caída, animándome en cada fracaso, sois todos muy especiales para mi.

A mi tutor del proyecto Gregorio Robles sin su comprensión y su ayuda esto no habría sido posible, ha sabido ayudarme cuando lo he necesitado y siempre ha estado disponible para cualquier duda, gracias.

A los profesores que he tenido a lo largo de mi vida, porque de cada uno de vosotros he aprendido una cosa nueva, gracias por vuestro esfuerzo, dedicación, por vuestra labor educativa y por sacar a los jóvenes de cada generación adelante.

A los que no están, que no significa que se hayan ido, gracias por todos los momentos que pase con vosotros, gracias por hacerme ver la vida de otra manera, nunca os olvidaré.

A todos, muchísimas gracias por haber ido de la mano conmigo este largo camino que ha sido mi etapa universitaria, todo esto no habría sido posible sin vosotros.

Resumen

Este trabajo tiene como objetivo realizar una comparación de desarrollo en dos plataformas iOS y Android. Para ayudarnos a realizar esta comparativa se utilizará una aplicación de prueba que nos servirá de punto de partida. A parte de esto, se verán las ventajas e inconvenientes que tiene el desarrollo, desde un punto de visto económico y también desde el punto de vista del aprendizaje. La aplicación de prueba consiste en un localizador de vehículos que también encuentra los últimos parkings en un determinado radio.

Índice general

1. Introducción **1**

1.1. Plataformas a analizar 2

1.1.1. Android 2

1.1.2. iOS 3

1.2. Motivación 4

1.3. Estudios anteriores 6

2. Objetivos **9**

3. Descripción Informática **11**

3.1. Metología 11

3.1.1. Ventajas 12

3.1.2. Inconvenientes 13

3.1.3. Conclusión 13

3.2. Requisitos 14

3.2.1. Requisitos funcionales 14

3.2.2. Requisitos no funcionales 15

3.3. Análisis 17

3.3.1. Diagrama de casos de uso 17

3.3.2. Diagrama de estados 18

3.3.3. Prototipo iOS 19

3.3.4. Prototipo Android 22

3.4. Diseño 25

3.4.1. Diseño iOS 25

3.4.2. Diseño Android 27
3.4.3. Diseño de la base de datos 29
3.5. Implementación 30
3.5.1. Implementación iOS 30
3.5.2. Implementación Android 42
3.6. Tecnologías utilizadas 52
3.7. Evaluación 53
3.7.1. Ejemplos gráficos 53
3.7.2. Ejemplo analítico 56

4. Conclusiones **59**
4.1. Objetivos cumplidos 59
4.2. Evaluación personal 61
4.2.1. Diseño y prototipo 61
4.2.2. Implementación 61
4.2.3. Despliegue 63
4.2.4. Conclusión 64
4.3. Líneas futuras 67

Índice de figuras

1.1. Gráfica sistemas operativos 2
1.2. Gráfica de descarga de aplicaciones 3
1.3. Demanda aplicaciones móviles 5
1.4. Versiones de Android 7

3.1. Diagrama de casos de uso 17
3.2. Diagrama de estados global 18
3.3. Diagrama de casos obtener posición 18
3.4. Diagrama de casos obtener posición 19
3.5. Diagrama de casos obtener posición 19
3.6. Diagrama de casos obtener últimos parkings 19
3.7. Pantalla principal iOS 21
3.8. Pantalla secundaria iOS 22
3.9. Pantalla principal Android 23
3.10. Pantalla secundaria Android 24
3.11. Ciclo de vida iOS 26
3.12. Ciclo de vida Android 28
3.13. Diagrama de base de datos 29
3.14. Declaración de variable iOS 32
3.15. Declaración de botón iOS 32
3.16. Declaración de botón iOS 32
3.17. Declaración de método privado en la interfaz 33
3.18. Declaración de método privado en la clase 33
3.19. Llamada función iOS 33

3.20. Metodo de lectura de posición 34
3.21. Obtener datos de las preferencias 34
3.22. Interfaz CoreData 35
3.23. Interfaz CoreData II 35
3.24. Creación de objetos bbdd iOS 36
3.25. Creación de objetos bbdd iOS II 36
3.26. Inserción en bbdd iOS 37
3.27. Select bbdd iOS 37
3.28. Declaración mapa iOS 38
3.29. Anadir un marcador iOS 38
3.30. Llamada asincrona iOS 39
3.31. Transición de pantallas iOS 40
3.32. Frameworks iOS 40
3.33. Localizar imágenes iOS 41
3.34. Declaración de manifest Android 42
3.35. Declaración de manifest Android II 43
3.36. Declaración botón Android 44
3.37. Declaración onClickListener 44
3.38. Declaración de método privado en la interfaz 45
3.39. Declaración de método público en la clase 45
3.40. Llamada función Android 45
3.41. Método de lectura de posición Android 46
3.42. Datos preferencias Android 47
3.43. Creación base de datos Android 47
3.44. Insertar posición Android 48
3.45. Obtener posiciones base de datos Android 48
3.46. Código mapa manifest Android 48
3.47. Declaración mapa en la interfaz Android 49
3.48. Declaración mapa en el código Android 49
3.49. Añadir un marcador Android 49
3.50. Llamada asincrona Android 50

3.51. Llamada asincrona Android II 50
3.52. Transición pantallas Android 51
3.53. Localizacion Android 51
3.54. Primera pantalla 53
3.55. Segunda pantalla 54
3.56. Segunda pantalla parkings 55
3.57. Tabla de todas las posiciones almacenadas 56
3.58. Tabla de todas las posiciones cercanas 57
3.59. Parkings 58

4.1. Posicionamiento de botones 62
4.2. Ventas de Smartphones y ganancias 65
4.3. Comparativa desarrollo tiempos 66
4.4. Tabla/Gráfica de coste inicial en el desarrollo 66

Capítulo 1

Introducción

Este proyecto tiene como objetivo realizar una comparación en el desarrollo de una aplicación en dos plataformas móviles. Para conseguir este propósito se ha decidido diseñar una pequeña aplicación que consiste en un localizador de vehículos que permite, desde el propio terminal, almacenar la posición de un vehículo para después consultarla y recibir las instrucciones pertinentes para que le lleven hasta su vehículo. A parte de esta opción, la aplicación nos permite consultar los lugares, de una zona concreta, donde hemos aparcado anteriormente. Antes de entrar en materia hay que tener claro una serie de conceptos, los cuales van a salir a lo largo de esta memoria y, de los cuales, es conveniente saber su significado.

- GPS: (Global Positioning System) o sistema de posicionamiento global, es una tecnología que, mediante la triangulación de al menos 3 satélites, consigue posicionar un objeto con un error de más o menos 100 metros. Se utilizará para obtener la ubicación del teléfono móvil.

- Base de datos: es un conjunto de datos pertenecientes a un contexto y almacenados sistemáticamente para su posterior uso. En este proyecto se utilizará para almacenar las posiciones del vehículo.

- Latitud: Es la distancia angular entre la línea ecuatorial y un punto determinado de la Tierra.

- Longitud: Es la distancia angular entre un punto dado de la superficie terrestre y el

meridiano que se tome como referencia, generalmente el meridiano de *Greenwich*.

1.1. Plataformas a analizar

Dentro de este trabajo hay que tener en cuenta que estamos trabajando en el entorno de la movilidad, es decir, que las tecnologías que vamos a analizar son entornos de programación móvil. Cada uno de ellos tiene su propio lenguaje de programación y sus plataformas de desarrollo. Las dos plataformas elegidas son *iOS* y *Android* ya que son las más populares dentro del mundo del desarrollo de aplicaciones móviles, entre las dos suman el 90 % de los sistemas operativos en teléfonos móviles y de descargas de aplicaciones a nivel global. Por este motivo se han descartado otros sistemas operativos como *Windows Phone* y *Blackberry Os*

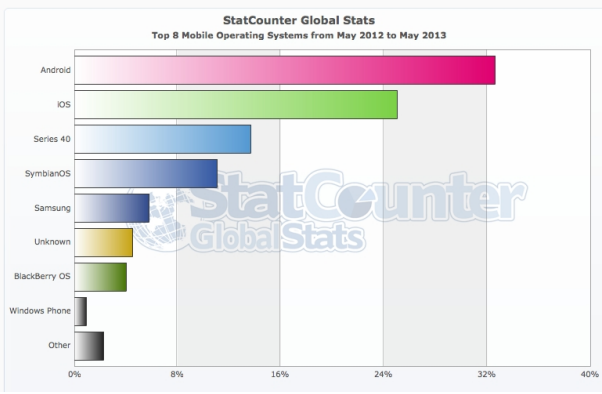


Figura 1.1: Gráfica sistemas operativos

1.1.1. Android

Es el sistema operativo que pertenece a la multinacional *Google*. Android nació en 2003. Su fundador fue *Andy Rubin* y tan solo 22 meses después fue adquirida por Google. Pero no fue hasta noviembre de 2007 cuando se hizo oficial la salida, de Android, al mercado. Desde su nacimiento ha ido de la mano de HTC. El fabricante coreano fabricó el primer Nexus, móvil con sistema operativo nativo Android. Posteriormente, se encargó de fabricarlo Samsung y, el último Nexus, LG. Actualmente, Android es uno de los sistemas operativos para móviles más importantes. Hay que destacar que este sistema operativo se programa

en Java aunque también hay plugins de .Net que permiten desarrollar aplicaciones.[17, 18]

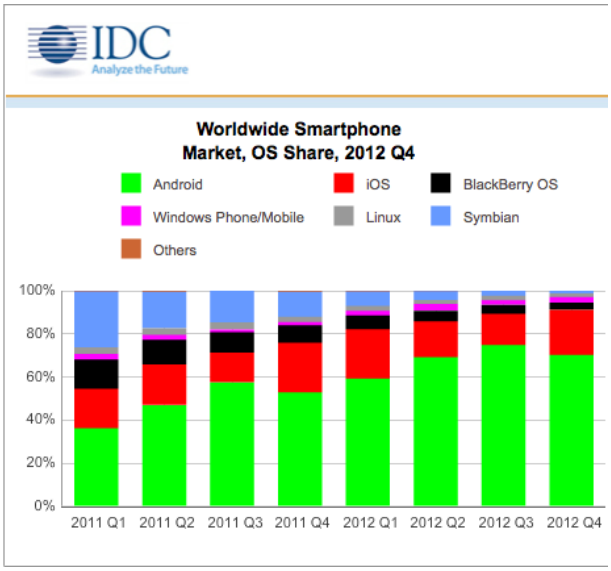


Figura 1.2: Gráfica de descarga de aplicaciones

1.1.2. iOS

También es conocido por *Iphone OS* es el sistema operativo desarrollado por *Apple*. iOS nace con la creación del primer móvil de la compañía californiana en agosto de 2007. Su SDK no fue liberado hasta Agosto de 2008. Gracias a esto, a la apertura de la tienda de aplicaciones Apple Store y, a que fue el primer dispositivo que unía en un solo *gadget*: teléfono, mp3, cámara de fotos, todo mediante un control táctil, fue cuando este sistema operativo adquirió fuerza y se alzó en el top de ventas y descargas de aplicaciones desarrolladas en los últimos años. El desarrollo de aplicaciones iOS sólo se puede realizar desde un ordenador con sistema operativo Mac Os ya que, para la generación del código, sólo se puede utilizar xCode que es propiedad de Apple y nativo de dicho sistema operativo. El lenguaje de programación que usa es Objetive C. Éste es un lenguaje de programación orientado a objetos basado en C. [14, 15]

1.2. Motivación

Con este trabajo lo que se quiere conseguir es tener una referencia previa que permita elegir una plataforma. Por este motivo, evaluaremos los inicios del desarrollo, entraremos dentro del código para ver aspectos más técnicos y veremos cómo se suben las aplicaciones a sus respectivas tiendas. Está es una cuestión importante ya que cuando se lanza al mundo de la tecnología móvil no se sabe muy bien qué plataforma elegir para desarrollar sus aplicaciones. Después de leer la memoria se deben tener conocimientos de implementación en ambas tecnologías y saber las restricciones de cada una de ellas. Esto hace que, en vez de pararse a analizar en detalle cada una de ellas y hacer un curso de introducción, consigamos que el lector opte por una sin perder demasiado tiempo. Saber lo que se va a encontrar a la hora de aprender el desarrollo de la misma.

La importancia de todo esto radica en que, al principio, cuando se va a realizar una elección de este tipo, no se sabe a lo que nos enfrentaremos a la hora de desarrollar una aplicación. En muchos casos, la decisión de elegir una plataforma u otra puede suponer el éxito o el fracaso de una aplicación o de una empresa. Si disponemos de recursos limitados y no podemos permitimos el lujo de desarrollar en ambas tecnologías, ya sea por tiempo o por dinero, esta decisión puede repercutir en el negocio. Además una cosa tan sencilla como la compra de un ordenador puede estar condicionada a la hora de elegir entre una plataforma u otra ya que, como se ha comentado anteriormente, para iOS es necesario un ordenador con sistema operativo Mac OS. De esta idea sale el objetivo de este trabajo de fin de máster: ayudar a los que se quieran iniciar en el mundo de las aplicaciones móviles, ya sea para uso personal o para crear una empresa.

Hay que tener en cuenta que, actualmente, hay unos 6 billones de usuarios de smartphones esto es en torno al 87 % de la población mundial. En China y en la India el crecimiento del mercado de este tipo de dispositivos creció el 30 % el pasado año. En los últimos tres años se han desarrollado más de 300.000 aplicaciones y Google ha ganado más de 2,5 mil millones de dólares en publicidad. Aparte de los ingresos por publicidad hay que tener en cuenta que las ganancias por venta de aplicaciones móviles al mes es de $1,200-$3,900 dependiendo de la aplicación y de la plataforma. Con el paso del tiempo, todas las aplicaciones de escritorio están realizando una migración a nuestro pequeño or-

denador de bolsillo, también llamado smartphone. Con este crecimiento, el sector de la tecnología móvil está sufriendo una fuerte renovación, llegando al punto de que *Microsoft* ha sacado su nuevo sistema operativo *Windows 8* basándose en Windows Phone. Por todo este crecimiento merece la pena evaluar las ventajas e inconvenientes del lanzamiento al mercado de una aplicación determinada.[1, 2, 3, 4, 5, 6, 7]

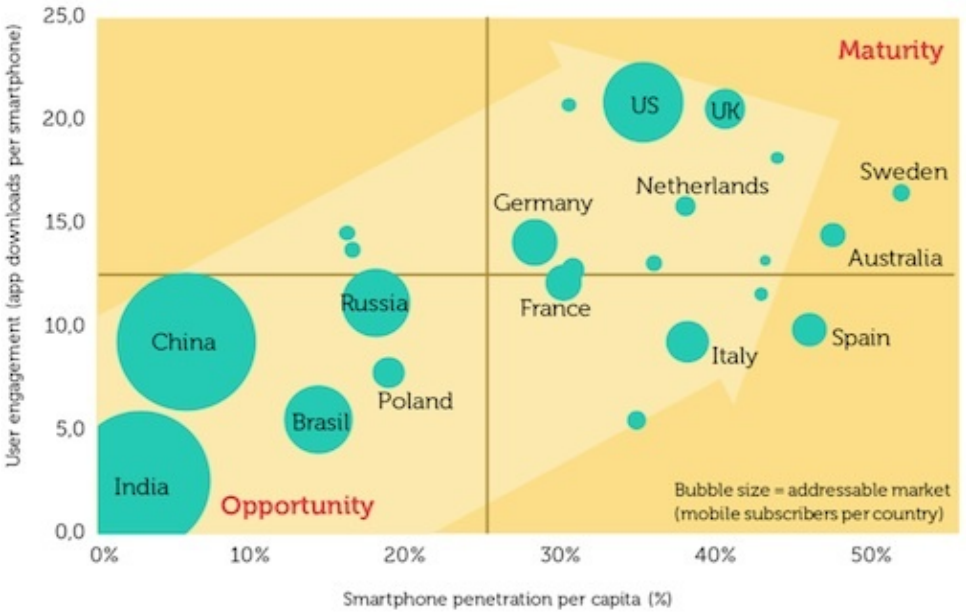


Figura 1.3: Demanda aplicaciones móviles

La memoria se estructura en estos apartados:

- **Objetivos**: dónde se definen los objetivos de este trabajo.
- **Descripción informática**: que esta compuesta por la metodología utilizada, requisitos, análisis, diseño, implementación, tecnologías utilizadas y evaluación.
- **Conclusiones**: dónde se verá si se han cumplido los objetivos, una opinión personal y las lineas futuras del proyecto.
- **Bibliografía**: se encuentran las referencias utilizadas.

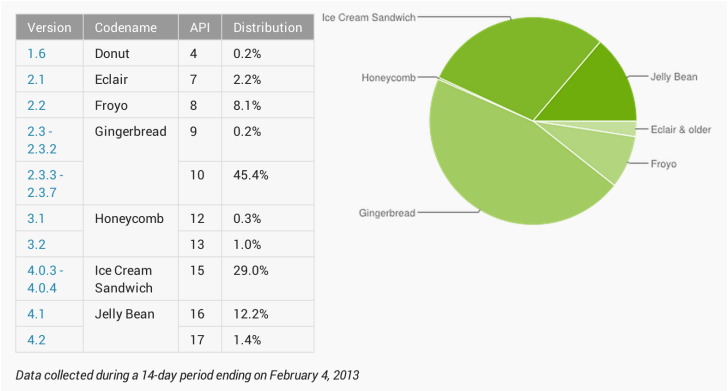
1.3. Estudios anteriores

En los últimos años han aparecido bastantes comparativas respecto al desarrollo en plataformas móviles. La mayoría de ellas se centran en Android e iOS por ser las más conocidas y las que son utilizadas por más usuarios. La mayoría de estudios están de acuerdo en que el principal problema de Android es el diseño para diferentes tamaños de pantalla y la fragmentación del sistema operativo. Está fragmentación supone que tan sólo el 70 % de los dispositivos están actualizados a la última versión del sistema operativo mientras tanto casi un 30 % de los dispositivos aún siguen bajo la versión 2.X.X de Android. Esto se produce porque los terminales no tienen suficiente capacidad para soportar las nuevas versiones y otras veces porque los terminales depende de actualizaciones de software de la marca y está decide dejar de actualizarlos. Sin embargo en iOS esta fragmentación es casi inexistente ya que casi todos sus dispositivos están actualizados a la última versión de su sistema operativo. También hacen referencias a nivel de hardware. Hay que tener en cuenta que se conocen las características técnicas de los dispositivos iOS mientras que en Android esto se desconoce por el gran número de dispositivos de características diferentes que existen en el mercado.[9, 10].

Por otro lado hay cierta indeterminación cuando se habla de los simuladores que ofrecen ambas plataformas ya que el estudio [8] dice que el simulador que proporciona xCode es mejor que el Android pero en los estudios [9, 10] dicen que es mejor el de Android. Hablando de aspectos más técnicos estos estudios sólo se refieren a la capa de interfaz gráfica. En este aspecto dicen que iOS al llevar todos los elementos en un mismo fichero hace que el código esté más agrupado y que los estilos creados por el propio desarrollador son más difícil implementarlos por código que en Android. También destacan el aspecto de la navegación entre pantallas de los sistemas. Dicen que el sistema de navegación de iOS es más intuitivo que en Android. Porque al tener el botón de atrás y la barra de navegación superior hace que el usuario siempre sepa dónde está. De Android critican que al tener la vuelta atrás en un botón físico hace confundirse a la gente ya que muchas veces este botón no los lleva a la pantalla anterior de la aplicación. Nos lleva a la pantalla anterior del dispositivo, es decir, si entro en un determinado correo al pulsar encima de una notificación mucha gente espera que al pulsar atrás vayas a todos los correos electrónicos

sin embargo, nos encontramos con que volvemos a la pantalla de inicio del móvil.

Todos ellos destacan que el proceso de revisión de iOS frena a muchos desarrolladores a la hora de elegir este lenguaje y todos coinciden que en este aspecto Android es mejor. Para concluir todos los estudios coinciden en que las ganancias para el desarrollador son del 70 % en ambos sistemas pero que la cuota anual de 99$ en iOS hace que en este aspecto Android sea una mejor opción. Según los datos [8, 9, 10] los usuarios de iOS están acostumbrados a pagar por las aplicaciones ya que al ser terminales tan caros hacen que el nivel económico de los usuarios sea más alto. Esto unido a que para el registro de una cuenta en el App Store es necesario dar los datos bancarios hacen que los usuarios pierdan al miedo a comprar en esta plataforma. Sin embargo en Android el uso de aplicaciones de pago es muy bajo. Esto es debido a que la mayoría de los usuarios usan markets alternativos ("piratas") dónde descargarse esas aplicaciones sin pagar nada. Este es el principal problema de sacar un producto de pago en esta plataforma.



Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	2.2%
2.2	Froyo	8	8.1%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	45.4%
3.1	Honeycomb	12	0.3%
3.2		13	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.0%
4.1	Jelly Bean	16	12.2%
4.2		17	1.4%

Data collected during a 14-day period ending on February 4, 2013

Figura 1.4: Versiones de Android

Tras ver estos estudios previos, en esta memoria se verán aspectos más técnicos y se hablará de las diferencias en el desarrollo de una aplicación. Basado en la imparcialidad respecto a la preferencia entre un sistema operativo u otro ya que estos estudios están muy influenciados respecto a la experiencia del desarrollador que los escribe y al éxito de sus aplicaciones en las respectivas tiendas.

Capítulo 2

Objetivos

La finalidad de este trabajo de fin de máster es conseguir utilizar los conocimientos aprendidos en este máster y plasmarlos en un trabajo que pueda ser de utilidad a la hora de elegir entre uno de los dos sistemas operativos para desarrollar una aplicación. La finalidad es realizar una comparativa entre dos plataformas de desarrollo y hacer una comparativa para, posteriormente, elaborar una conclusión de las ventajas e inconvenientes de cada una de ellas.

1. **Modelado y prototipo de la aplicación ejemplo**: Se deberá realizar un prototipo para el posterior desarrollo de la aplicación móvil.

2. **Aplicación móvil en las dos plataformas**: Realizar una aplicación móvil en las dos plataformas. Constará de una base datos sqlite, mapas de Google e interacción con un servicio web.

3. **Análisis de desarrollo de cada plataforma**: Se realizará un análisis del desarrollo en cada plataforma, donde se explicará el flujo de datos y las características de cada una de las mismas.

4. **Comparación de desarrollo**: Realizar una comparación entre el desarrollo de la aplicación en ámbar plataformas.

5. **Integración continua**: Para el desarrollo y despliegue de la aplicación se debe utilizar un sistema de control de versiones.

Una vez cumplido estos objetivos se debería tener una idea de qué plataforma elegir a la hora de realizar una aplicación móvil, ya que se tendrán conocimientos mínimos del funcionamiento de cada una de las dos plataformas, su forma de despliegue y sus principales características.

Capítulo 3

Descripción Informática

En esta sección analizaremos toda la descripción informática del proyecto, desmenuzaremos el proceso de análisis, del diseño de la arquitectura, los detalles de la implementación y la evaluación del proyecto.

3.1. Metología

La metodología que se ha usado para el desarrollo de ambos proyectos ha sido el modelo basado en prototipos. Esta metodología consiste en realizar un prototipo sin gastar mucho tiempo en el desarrollo de éste y que se centra en la representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final. Este diseño conduce a la construcción de un prototipo, el cual será evaluado por el cliente para una recibir una retroalimentación de éste; gracias a esta podremos redefinir los requisitos del software que se desarrollará y también permite al desarrollador tener una primera toma de contacto y conocer de primera mano lo que el cliente desea que realice la aplicación. Todo este aprendizaje se traduce en una mejora del resultado final. Para realizar este modelo es necesario seguir estas etapas:

1. **Plan rápido**: El modelo que inicia con una primera toma de requisitos, donde se detalla la funcionalidad debe tener: el número de pantallas, la funcionalidad de cada una de ellas, la versión para la cual va a ser desarrollada entre otros muchos requisitos.

2. **Modelado, diseño rápido**: Después de la toma de requisitos se comienza el diseño de la aplicación, este diseño no debe ser un diseño completo, tan sólo debe plasmar la funcionalidad de la aplicación, detallar el número de botones, la posible posición de estos y dar una aproximación de lo que visualmente se puede parecer la aplicación. Para este paso se suelen utilizar varias herramientas de prototipado. En este trabajo se ha usado *Wireframesketcher*. Esta herramienta nos permite realizar un prototipo funcional de las pantallas en el que incluso se le puede meter una posible navegación mediante *HTML*. Esta primera aproximación se le entregará al cliente para que verifique la funcionalidad de la aplicación y se redefinan los requisitos funcionales.

3. **Desarrollo, entrega y retroalimentación**: Después del desarrollo del primer prototipo, se realiza un prototipo nativo en la aplicación una primera aproximación de la misma. En esta primera entrega se tendrá la navegación completa por la aplicación y se empezará a atisbar parte de la funcionalidad. Se empezarán a implementar las primeras capas de diseño de la aplicación. Se entregará al cliente y se recibirán los cambios propuestos y se continuará con el desarrollo. Esta fase se puede repetir varias veces, dependiendo del tipo de proyecto y de la magnitud de éste.

4. **Entrega del desarrollo final**: Al termino del desarrollo y de las pruebas, el software es entregado al cliente.

3.1.1. Ventajas

La principal ventaja de este modelo de desarrollo es que es muy útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida. También permite un primera idea del software en un corto espacio de tiempo y gracias a la retroalimentación con el cliente permite que los cambios que haya que hacer se hagan sobre la marcha y no demoren demasiado el desarrollo. Permite un contacto constante con el cliente y que éste no viva con la incertidumbre del resultado final y hace partícipe del desarrollo del producto.

3.1.2. Inconvenientes

El usuario tiende a crearse una gran expectativa cuando ve el prototipo de cara al resultado final. Muchas veces éstas pueden ser demasiado altas y a veces obligan a modificar el producto final para satisfacer lo que se le enseñó al cliente con el primer prototipo. Hay que tener en cuenta que el desarrollo del primer prototipo es muy rápido y se suelen desatender aspectos importantes como la calidad y el mantenimiento.

3.1.3. Conclusión

Al ser un proyecto donde prima es la velocidad de resultados se elige este modelo ya que permite avanzar muy rápido en el desarrollo y al ser un proyecto de pequeña escala la modificación o remodelación del mismo no supone un gran coste.

3.2. Requisitos

Antes de empezar un proyecto de software es necesario saber la funcionalidad de la aplicación desarrollada, cuál es su alcance y las limitaciones con las que cuenta tanto a nivel de funcionalidad como de hardware. Por ello a la hora de iniciar un proyecto una de las primeras tareas es reunirse con el cliente con el fin de conocer sus necesidades y realizar una toma de requisitos. En este caso, como el proyecto no es para ningún cliente concreto se han de elaborar igualmente pero solicitando éstos al tutor del proyecto. Para ello se dividen los requisitos en tres tipos, requisitos funcionales, requisitos no funcionales y requisitos de interfaz.

3.2.1. Requisitos funcionales

En este apartado se estudian los requisitos funcionales que están relacionados con el proyecto. Para detallar cada uno de ellos se usará la siguiente nomenclatura RFXX donde RF son las siglas abreviadas de Requisito Funcional y XX el número de requisito. Por tanto, los requisitos funcionales son los siguientes:

RF01 La aplicación debe ser una aplicación móvil.

RF02 La aplicación debe obtener la posición mediante el uso de la señal GPS del dispositivo.

RF03 En caso de que no exista una señal GPS válida deberá obtener la posición a través de la red *Wi-Fi* o *3G* en su defecto.

RF04 La aplicación debe almacenar la última posición tanto en memoria interna como en una base de datos sqlite.

RF05 La aplicación debe tener un mapa donde indique la posición del usuario.

RF06 La aplicación debe mostrar en dicho mapa la última posición almacenada.

RF07 La aplicación debe poder mostrar los últimos lugares donde se ha aparcado en un radio de 5 kilómetros.

RF08 La aplicación debe ser capaz de calcular la ruta hasta el último parking siempre y cuando sea posible.

RF09 La aplicación debe mostrar un mensaje de error en caso de que no haya sido capaz de obtener la posición

RF10 La aplicación debe mostrar un mensaje de éxito en caso de que se almacene la posición correctamente.

RF11 La aplicación debe mostrar un mensaje de error si no se tiene conectividad a Internet para el cálculo de la ruta hacia el vehículo.

RF12 La aplicación debe mostrar un mensaje de error si no encontró una ruta válida.

RF13 La aplicación debe permitir situar al usuario en el mapa.

Además en este apartado, cabe destacar los siguientes requisitos de interfaz:

RF29 La aplicación debe proporcionar al usuario una interfaz de fácil manejo.

RF30 La aplicación debe proporcionar una pantalla principal donde se permita obtener la posición.

RF31 La aplicación debe proporcionar una pantalla principal donde se permita acceder a una segunda pantalla.

RF32 La aplicación debe proporcionar una pantalla segunda pantalla que mostrará un mapa con un marcador.

RF33 La aplicación debe proporcionar un marcador donde detalle un texto y las coordenadas del vehículo.

RF34 La aplicación debe proporcionar una navegación fluida.

3.2.2. Requisitos no funcionales

En este apartado se agrupan los requisitos de recursos, rendimiento, mantenimiento y portabilidad entre otros.

Para detallar cada uno de ellos se utiliza la siguiente nomenclatura: RNFXX donde RNF son las siglas abreviadas de requisito no funcional y XX el número de requisito. Previo al detalle de cada requisito se explica el tipo del que forma parte en función de las agrupaciones citadas anteriormente.

Por tanto los requisitos no funcionales son los siguientes:

1: Los requisitos de recursos especifican las características de los equipos y dispositivos participantes en la infraestructura del sistema:

RNF01 Se precisa un teléfono móvil con sistema operativo Android o iOS.

RNF02 La versión de Android será como mínimo una 3.0.0

RNF03 La versión de iOS será la version 5 o superior.

RNF04 En el caso de ser un dispositivo Android deberá tener instalado Google Play Services.

2: Los requisitos de rendimiento especifican cálculos temporales como, entre otros, el tiempo de ejecución.

RNF05 El tiempo de arranque de la aplicación debe ser eficiente, es decir, inferior a 5 segundos.

RNF06 El tiempo de transición entre pantallas debe ser eficiente, es decir, inferior a 5 segundos.

RNF07 El tiempo de obtención de la posición debe ser breve, menor a un minuto.

RNF08 El tiempo de carga del mapa debe ser breve, menor a un minuto.

RNF09 El tiempo de la obtención de los puntos debe ser muy breve, menor a 10 segundos.

RNF10 El tiempo de carga de la posición actual debe ser muy breve, menor a 10 segundos.

3.3. Análisis

La aplicación está compuesta de dos pantallas. La primera almacena la posición actual y tiene un botón que lanza la segunda pantalla. En ella se verá la ruta hasta el vehículo y se podrán consultar los últimos parkings. En esta sección veremos los diagramas de casos de uso, el diagrama de estados y el prototipo de la aplicación.

3.3.1. Diagrama de casos de uso

El proyecto se ha dividido en cinco casos de uso, estos casos son :

Menú principal: Es la parte central de la aplicación desde aquí se puede acceder al mapa y a almacenar la posición.

Obtener posición: Se encarga de obtener la posición a través del GPS y de almacenarla tanto en preferencias como en una base de datos.

Mapa: Acciona la transición a la segunda pantalla y calcula la ruta desde la posición actual hasta la última posición almacenada.

Últimos parkings: Obtiene de la base de datos los últimos parkings y los muestra en el mapa.

Cálculo de rutas: Calcula la ruta cargando la posición del vehículo y la posición actual obtenida por el GPS.

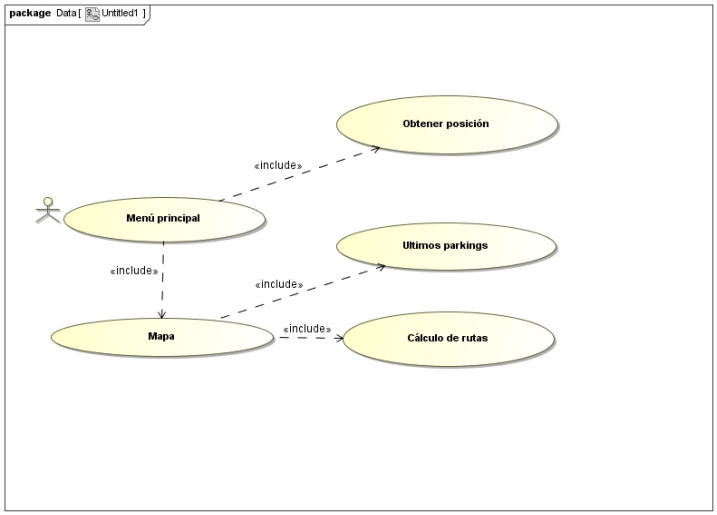


Figura 3.1: Diagrama de casos de uso

3.3.2. Diagrama de estados

- **Diagrama de estado global** Este diagrama muestra el funcionamiento de la aplicación. Primero se inicia la ventana principal en la que se tiene la opción de almacenar la posición actual o ir al mapa. En la pantalla del mapa se calculará la ruta hasta la última posición almacenada y se verán los lugares donde se ha aparcado en un radio de 5 kilómetros.

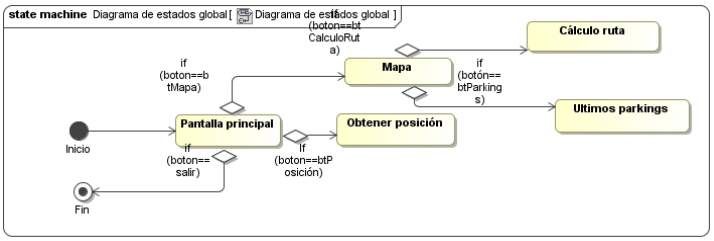


Figura 3.2: Diagrama de estados global

- **Diagrama de estado obtener posición** Se encarga de obtener la posición del sensor GPS. Una vez obtenida la almacena en las preferencias del teléfono y por último la almacena en la base de datos.

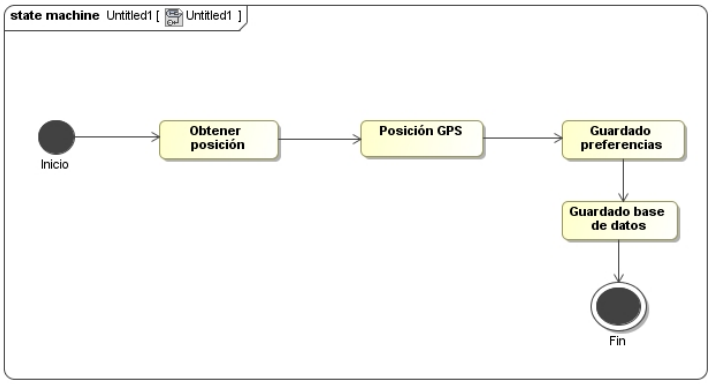


Figura 3.3: Diagrama de casos obtener posición

- **Diagrama de estado mapa** Se encarga de mostrar la segunda pantalla y el cálculo de la ruta.

- **Diagrama de estado cálculo de ruta** Obtiene la posición actual del sensor GPS y la posición del vehículo de las preferencias. Se llama a un servicio web que devuelve la ruta para posteriormente pintarla en el mapa.

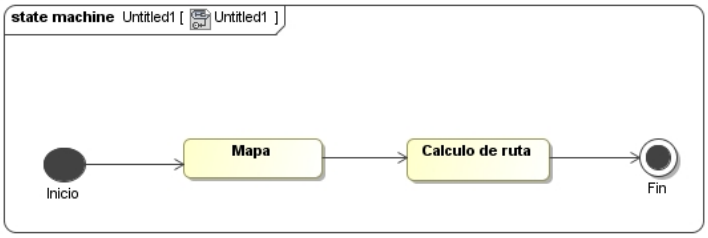


Figura 3.4: Diagrama de casos obtener posición

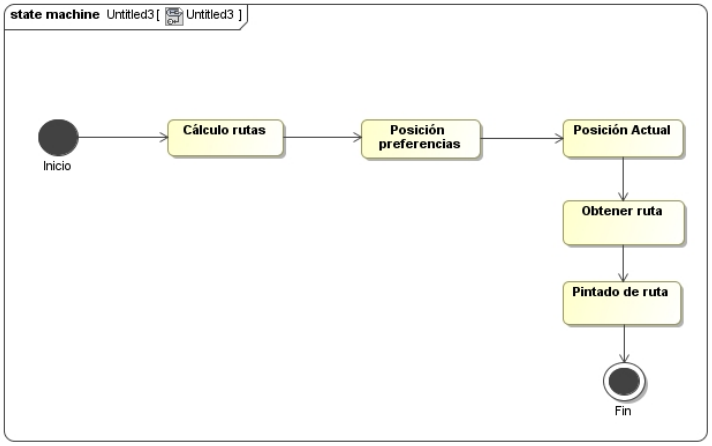


Figura 3.5: Diagrama de casos obtener posición

- **Diagrama de estado cálculo de ruta**

  Con la posición actual se obtienen las posiciones en un radio de 5 kilómetros de la base de datos y se muestran en el mapa.

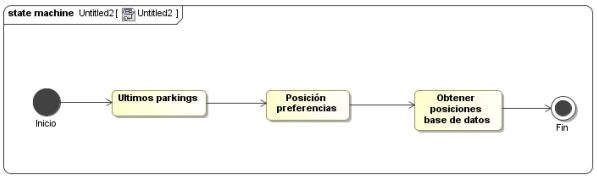


Figura 3.6: Diagrama de casos obtener últimos parkings

3.3.3. Prototipo iOS

En esta fase del desarrollo se genera un prototipo para tener una guía de diseño y para tener fijado un claro objetivo del desarrollo de la aplicación. Hay que tener en cuenta

las reglas de diseño que vienen fijadas desde Apple[1] donde nos indican las principales características de interfaz que debe tener una aplicación para que esta sea aprobada en el proceso de revisión del App Store.

Pantalla principal En esta primera pantalla del prototipo podemos ver cómo se sitúan los botones, alineados uno encima del otro y cómo se le presenta al cliente con una apariencia de un dispositivo Iphone. Como es el primer prototipo no es necesario que esté incluido el diseño de la aplicación, tan sólo con una muestra de la funcionalidad principal de la misma es suficiente. La funcionalidad de cada uno es:

- **Botón estoy aquí**: Este botón almacenará la posición actual utilizando la señal GPS o los datos que obtenga de la red a través del dispositivo móvil.
- **Botón ver mapa**: Este botón nos llevará a la pantalla secundaria.

[1] `http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html`

Figura 3.7: Pantalla principal iOS

Pantalla secundaria Esta pantalla consta de tres botones. En el caso del botón ätrásës obligatorio para iOS porque no tiene un botón volver vía hardware como disponen los terminales Android. Según las guías de diseño, este botón debe estar en la parte superior izquierda de la pantalla. La funcionalidad de los botones son:

- **Botón ir a mi posición**: Este botón situará el mapa en tu posición actual.
- **Botón ir a mi coche**: Este botón en principio te indicaba la ruta hacía tu último parking desde la posición actual.

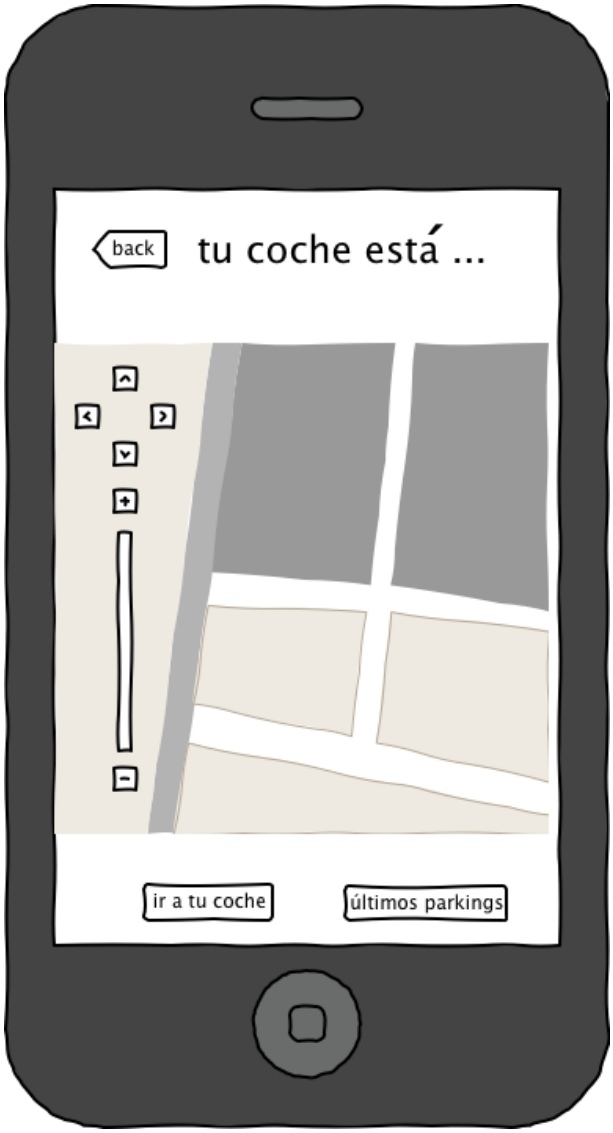


Figura 3.8: Pantalla secundaria iOS

3.3.4. Prototipo Android

Al igual que en el caso de iOS el prototipo de Androide debe ser similar a éste, ya que la aplicación va a tener la misma funcionalidad y la única diferencia es el tipo de interfaz usuario ya que, en esta plataforma, no existe un proceso de revisión pero, al igual que iOS, tienen una guía de consejos de diseño[2].

Pantalla principal En esta primera pantalla del prototipo podemos ver cómo se sitúan los botones, alineados uno encima del otro y, como ocurría con el prototipo iOS, se presenta

[2] `http://developer.Android.com/design/get-started/principles.html`

al cliente una apariencia Android. A diferencia de iOS no es necesario incluir un botón para volver a la pantalla anterior ya que Android, por defecto, tiene un botón físico para esta función. Al ser el primer prototipo no hay añadido ningún elemento de diseño final, tan sólo se muestra funcionalidad.

- **Botón estoy aquí**: Este botón almacenará la posición actual utilizando la señal GPS o los datos que obtenga de la red a través del dispositivo móvil.
- **Botón donde esta mi coche**: Este botón nos llevará a la pantalla secundaria

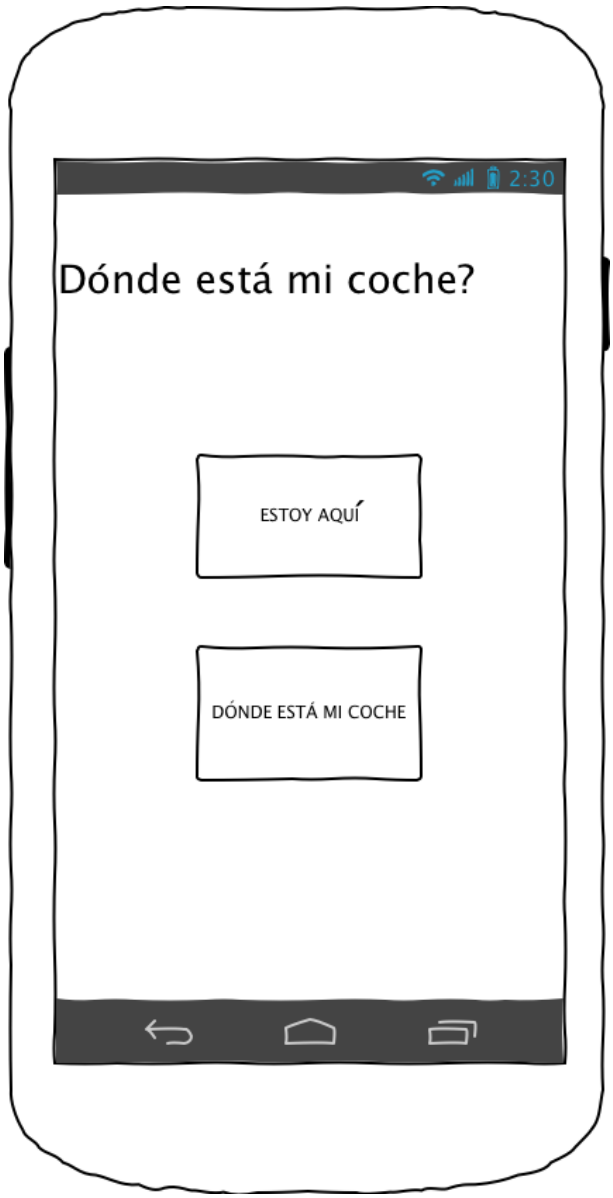


Figura 3.9: Pantalla principal Android

Pantalla secundaria 3.10 Esta pantalla sólo consta de dos botones ya que el de volver hacía atrás no esta como hemos comentado en el párrafo anterior.

- **Botón ir a mi posición**: Este botón situará el mapa en tu posición actual.
- **Botón ir a mi coche**: Este botón indica la ruta hacía tu último parking desde la posición actual.

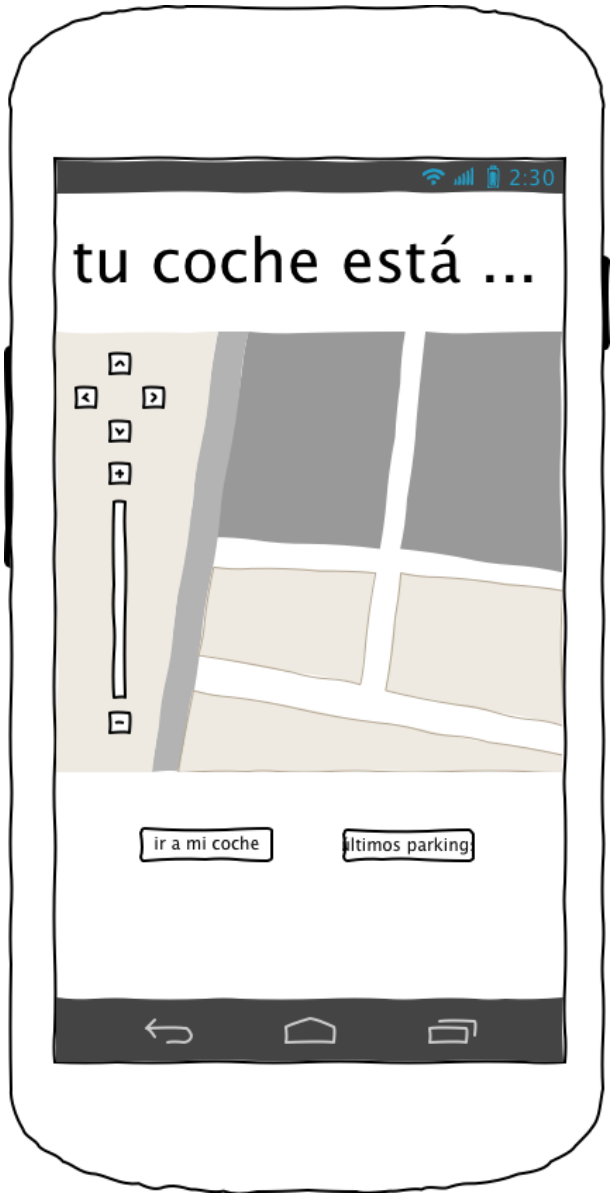


Figura 3.10: Pantalla secundaria Android

3.4. Diseño

A continuación detallaremos la estructura de clases que se van a utilizar para este proyecto, teniendo en cuenta que la estructura se repetirá para ambas plataformas, cada una de ellas con sus particularidades. Pero ambas van a tener definidas una clase para la pantalla principal y otra clase para la pantalla secundaria como mínimo.

3.4.1. Diseño iOS

A la hora de diseñar la estructura de clases en iOS hay que tener en cuenta que este sistema funciona a través de controladores. En este caso usaremos lo que se denominan *view controllers*. Estos controladores generan una serie de métodos por los que la aplicación va pasando según evoluciona en el tiempo y permiten la inserción de cualquier elemento visual, como botones, imágenes, cuadros de texto o mapas. El ciclo de vida de un view controller está formado por los siguiente métodos:

- **ViewDidLoad**: Es el primer método que se lanza al entrar la aplicación en el view controller. En este método se iniciaran las llamadas relacionadas con la obtención de datos de servicios, se comprobará la conectividad y se iniciará la carga de imágenes y de elementos estáticos.

- **ViewWillAppear**: Es el método que se lanza cuando aparece la interfaz del view controller se suele utilizar para actualizar datos que queremos que el usuario vea.

- **VIewDidApperar**: Es el método por el cual pasa la aplicación una vez que ha aparecido la vista. En este método se hacen las modificaciones que queremos que el usuario aprecie, ya que la pantalla ya está cargada.

- **ViewWillDisappear**: Se lanza cuando la pantalla va a desaparecer, en este método se suelen cerrar las conexiones de red, bases de datos, etc. Se prepara para la salida de la pantalla.

- **ViewDidDisappear**: Se ejecuta cuando la pantalla desaparece pero aún está cargada en memoria.

- **ViewDidUnload**: Es el último método que se lanza y es cuando la pantalla desaparece de la pila de memoria. En iOS anteriores a la versión 5 cuando la gestión de memoria era manual se utilizaba para liberar la memoria que se había reservado durante la ejecución del view controller.

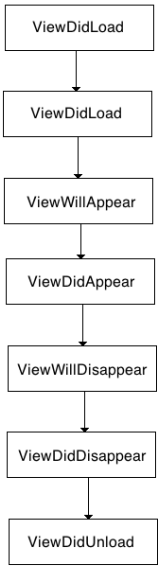


Figura 3.11: Ciclo de vida iOS

Teniendo en cuenta el ciclo de vida de este sistema operativo, se deben separar las diferentes pantallas en un view controller diferente, permitiendo encapsular el código, lo que permite que la división de tareas sea más fácil y, a la hora de transformar el diseño en código hace que está tarea sea más sencilla. Tras conocer el ciclo de vida de un view controller ahora es necesario saber cómo se muestran en la pantalla del dispositivo los elementos que deseemos. En este caso a partir de la versión 5 de iOS, proporciona al desarrollador una herramienta muy potente de creación de interfaces de usuario que viene junto a xCode se llama storyboard. Con él podemos insertar todas las pantallas que queramos tan sólo arrastrando los view controllers del menú e insertando en cada uno de ellos todos los elementos de interfaz gráfica que deseemos: botones, tablas e imágenes. A la hora del diseño de la interfaz de una aplicación móvil en iOS hay que tener en cuenta que solo hay tres resoluciones (640 x 1136) ,(640 x 960), (320 x 480) y solo tres dimensiones de

pantalla. Esto nos permite implementar las interfaces de una manera más sencilla ya que sólo tenemos que tener en cuenta esos tres factores. Incluso, si solo queremos diseñarlas para IPhone 4S y anteriores también podemos hacerlo, el único inconveniente es que en el IPhone 5 se nos verían dos bandas negras ajustando la pantalla al tamaño de una de los anteriores. Otra ventaja es que nos permite consumir menos recursos gráficos, ya que serían para tres pantallas aunque en la mayoría de los casos sólo se hacen recursos para dos ya que la resolución del IPhone 5 y 4S es la misma y se usan degradados para no tener que hacer más imágenes, lo que aligera la aplicación hablando en términos de coste de memoria.

3.4.2. Diseño Android

En el caso del sistema operativo Android hay que tener en cuenta que, al estar basado en Java, está pensado para estar estructurado en clases. La diferencia con iOS es que aquí los view controllers se llaman *activities*. Estas actividades, al igual que los view controllers, tienen una serie de métodos por los cuales van pasando según se van ejecutando lo que denominaremos ciclo de vida de una activity:

- **OnCreate**: Es el primer método que se lanza al entrar la actividad. En él se inicializan los atributos que van a estar cargados en memoria.
- **OnStart**: Después se pasa a este estado. En él se suelen llamar a métodos que inician la carga de datos y obtienen información que posteriormente queramos tratar.
- **OnResume**: Este método se ejecutará cada vez que la aplicación vuelva a estar activa, es decir, una vez haya sido desbloqueada. En él se suele hacer una recarga de datos y contiene el flujo principal de la aplicación.
- **OnPause**: Se inicia cuando la aplicación va a entrar en pausa. Se suele preparar la aplicación para una posible salida de ella y para almacenar las variables en memoria.
- **OnStop**: Se lanza cuando la aplicación se para.
- **OnDestroy**: Se ejecuta cuando la actividad se destruye, es decir, cuando se pasa de una actividad a otra y se ejecuta un finish() de la misma. Se libera toda la memoria de la aplicación y desaparece de la pila.

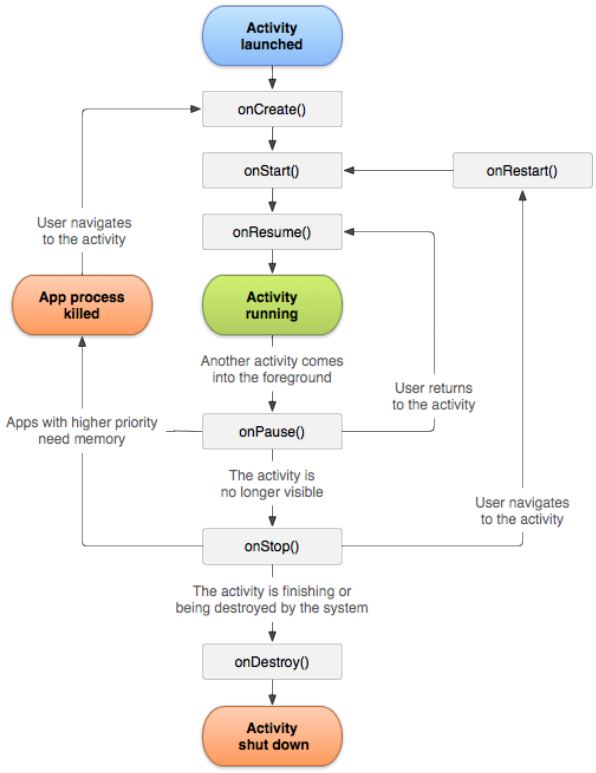


Figura 3.12: Ciclo de vida Android

Después de ver el ciclo de vida de la aplicación ya podemos saber dónde iniciar la carga de las variables, dónde realizar el tratamiento de datos y en qué métodos almacenar los mismos. A la hora de diseñar la interfaz gráfica hay que tener en cuenta que en este sistema operativo hay una gran gama de pantallas, partiendo de todas las dimensiones y de todas las densidades de las mismas. Por este motivo, a la hora de tener en cuenta los recursos gráficos de Android, agrupamos las imágenes en varios tamaños teniendo en cuenta la densidad de pantalla de cada grupo. Estos grupos son:

- **ldpi** Low dots per inch: Son las pantallas de menos resolución. Las imágenes suelen estar escaladas en torno a los 120dpi, el tamaño es de 420dp x 320dp.

- **mdpi** Medium dots per inch: Las imágenes están escaladas en torno a los 160dpi y el tamaño es de 470dp x 320dp.

- **hdpi** Hight dots per inch: Las imágenes están escaladas en torno a los 240dpi y el tamaño es de 640dp x 480dp.

- **xhdpi** Extra hight dots per inch: Las imágenes están escaladas en torno a los 320dpi y el tamaño es de 960dp x 720dp.

A la hora de la integración del diseño es mucho más complicado ya que no se pueden utilizar posiciones absolutas. Al haber tanta diversidad entre una pantalla y otra los elementos se descolocan, por lo que hay que posicionarlos de manera relativa, lo que complica muchísimo la colocación de los mismos. En Android las interfaces gráficas se definen mediante ficheros XML independientes. Dentro de estos ficheros XML se generan los layouts. Estos layaouts son las interfaces y pueden ser lineales (linear layout) o relativos (relative layout). Aparte de estos dos tipos de layouts, también hay otros elementos como fragmentos, botones e imágenes que se utilizan para completar las interfaces.

3.4.3. Diseño de la base de datos

La base de datos es un componente que van a tener en común ambos sistemas. Se va a utilizar para almacenar las posiciones que vayamos recogiendo con el sensor GPS para después obtenerlas para hacer el cálculo de los vehículos más cercanos. Esta base de datos será de tipo Sqlite que es un formato reducido de Sql pensado para plataformas móviles. Esta es compatible tanto para iOS como para Android. La base de datos está compuesta por una única tabla donde se almacenan las posiciones por latitud y longitud. El tipo de dato para estos campos será float.

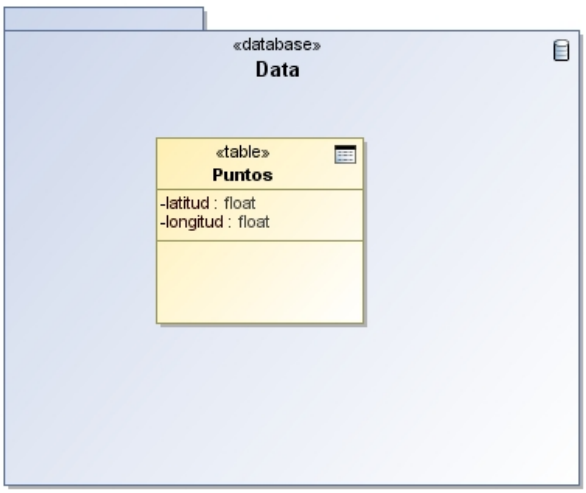


Figura 3.13: Diagrama de base de datos

3.5. Implementación

En esta sección nos centraremos en analizar la implementación del código donde veremos cómo se declaran las variables, los métodos y el acceso a la interactuación con los botones entre otras cosas. Antes de empezar hay que tener en cuenta que a la hora de implementar la aplicación de prueba varios aspectos. Primero tenemos que ser conscientes de que va a ser necesario el uso de la geolocalización para situarnos con nuestro dispositivo móvil y que también será necesario el uso de memoria para almacenar las posiciones que se vayan tomando.

3.5.1. Implementación iOS

Aquí veremos cómo se implementa en la plataforma de iOS en su versión 5. Para comenzar hablaremos de la estructura del código de iOS. Como está basado en C todos los archivos contienen una cabecera donde se declaran los atributos, variables y donde también están declarados los elementos de interfaz de usuario. En este caso toda aplicación se lanza desde el appdelegate correspondiente es la clase que controla el inicio y el fin de la aplicación. El ciclo de vida es éste:

- **didFinishLaunchingWithOptions**: Este método es el primero que se carga y sirve para recuperar variables como tokens de inicio de sesión entre otras cosas.

- **applicationWillResignActive**: Se lanza cuando la aplicación pasa de activa a inactiva. Esto ocurre cuando son cortes temporales como llamadas de teléfono, mensajes o cuando el usuario cierra la aplicación y entra en transición a segundo plano.

- **applicationDidEnterBackground**: Se ejecuta cuando la aplicación ha entrado en segundo plano y se utiliza para liberar memoria, almacenar datos de usuario y almacenar datos del estado de la aplicación para, después, cargarlos cuando esté activa.

- **applicationWillEnterForeground**: Se llama cuando pasa de ejecutarse en backgroud a un estado inactivo.

- **applicationDidBecomeActive**: Restaura todas las tareas que estuviesen pausadas mientras la aplicación haya estado inactiva. Si la aplicación ha estado en background, opcionalmente, actualiza la interfaz de usuario.

- **applicationWillTerminate**: Se llama cuando la aplicación está a punto de terminar.

Después de ver cómo funciona la carga, vamos a pasar a ver cómo se declaran las variables, cómo se implementan los botones, los métodos y los controles de interfaz. Los atributos de cada clase los declararemos en el archivo de cabeceras con extensión **.h**. Hay que tener en cuenta que a la hora de declarar los atributos de la clase hay dos campos.

En estos declararemos las propiedades de la variable. Estas propiedades se usan para indicar al compilador cómo se tratan estas mismas en la memoria y para la declaración automática de los get y los set (estos métodos se usan para obtener y asignar las variables para mantener la encapsulación). Las propiedades son las siguientes:

- **readonly**: Esta propiedad sólo declara el método get.

- **readwrite**: Esta propiedad declara los métodos get and set.

- **nonatomic**: Esta propiedad hace que la variable no sea atómica ,es decir, que a la hora de que accedan a esa variable, otros hilos de ejecución pueden acceder a esta misma variable. Si ha sido parcialmente escrita puede haber un problema de consistencia de datos. La ventaja del uso de esta variable es que el acceso es mucho más rápido.

- **atomic**: Esta propiedad hace que la variable sea de tipo atómico, es decir, no va a haber problema en el acceso a esta, aunque accedan a ella diferentes hilos.

- **retain**: Esta propiedad es necesaria cada vez que el atributo es un puntero a un objeto que conservará en memoria. Este tipo de variables requiere que luego se liberen.

- **assign**: Esta propiedad hace que los valores se asignen a los atributos. Se debe usar para atributos que no sean objetos.

- **copy**: Esta propiedad se usa cuando el atributo es un objeto mutable y se quiere que permanezca intacto, es decir, que el objeto no se modifique al cambiar el objeto que se le pasa como parámetro.

Todas estas opciones nos permiten elegir qué tipo de propiedad se adapta mejor para declarar nuestro atributo, al igual que permite que manejemos la reserva de espacio y la destrucción y el control de acceso sobre estos. Tras la declaración de variable es el turno de ver cómo se declara un botón y cómo se captura el evento del mismo, una vez que éste es pulsado. A la hora de declarar un botón se puede hacer de dos formas introduciendo el código a mano o mediante el storyboard arrastrando el botón al archivo de cabecera correspondiente y seleccionando el tipo de propiedad, si queremos que el botón ejecute una acción. Aquí podemos ver cómo está declarado un botón en el archivo de cabecera.

```
@property (nonatomic,assign) float latitud;
```

Figura 3.14: Declaración de variable iOS

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

Figura 3.15: Declaración de botón iOS

El método que hay que generar o que se genera sólo mediante el storyboard, el cual se lanza cada vez que se pulsa el botón que hemos declarado:

```
- (IBAction)btImHere:(id)sender {
    [self setposition];
}
```

Figura 3.16: Declaración de botón iOS

Después de declarar los dos botones de la aplicación de prueba. Se verá como declarar un método en iOS. Lo primero que hay que decidir es si el método será público o privado. En caso de ser un método público, habrá que declararlo en el fichero de cabeceras. En el caso de ser un método privado, deberá estar declarado en la parte de la clase reservada para las interfaces.

La declaración del método en la clase es parecida a la de cualquier lenguaje de programación, muy similar a la de C.

```
@interface MapViewController ()
-(void)calculateRoute:(float)latitudOrigen longitudOringen:(float)longitudOringen;
-(void)pintar:(float)latitudO longitudO:(float)longitudO coche:(bool)isCar ;
```

Figura 3.17: Declaración de método privado en la interfaz

```
-(void)calculateRoute:(float)latitudOrigen longitudOringen:(float)longitudOringen{
```

Figura 3.18: Declaración de método privado en la clase

Realizar la llamada a un método es un poco confuso, ya que para ello es necesario el uso de corchetes y el nombre del método, es decir, el acceso a los métodos de una clase se realiza con los corchetes. En caso de ser otra clase sería: nombre de la clase espacio y el nombre del método, después, dos puntos para separar el nombre del primer valor que recibe como parámetro y el resto de parámetros se reciben con el nombre del parámetro seguido de dos puntos y el nombre de la variable que recibe como segundo parámetro. Esto se realiza sucesivamente, según el número de estos que tenga la función.

```
[self pintar:lat longitudO:longs coche:true];
```

Figura 3.19: Llamada función iOS

Una vez declarados los métodos se mostrará el uso del GPS en la aplicación. Para implementar esta funcionalidad es necesario que nuestra clase herede de *CLLocationManagerDelegate* la cual nos genera unos métodos por defecto que permiten iniciar la localización. Crearemos una variable de tipo *CLLocationManager* que será la que disparará un método que controla la actualización de la posición.

didUpDateToLocation: Este método lo que hace es ver si hay un cambio de posición. En caso de que lo haya muestra las dos posiciones: la posición nueva y la posición anterior. Una vez hemos obtenido la posición, aunque sólo sería necesaria almacenarla en la base de datos, lo haremos también en la memoria. De este modo, podremos ver otra de las características más comunes de las aplicaciones. Para indicar que este método se va a lanzar, es necesario indicar que se va iniciar el posicionamiento. Esto se realiza con la variable *locationManager*, donde la indicaremos que el método que va a recoger la llamada es la clase misma, le indicamos un filtro (en este caso no será ninguno) y seleccionamos la precisión que deseamos, que en este caso será la mejor presión posible. Hay que tener

en cuenta que, cuanto mayor sea la precisión que queramos obtener, mayor será el tiempo necesario para que el terminal obtenga dicha precisión.

```
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation
```

Figura 3.20: Metodo de lectura de posición

Tras obtener la posición, el siguiente paso es ver cómo se almacenan, en la memoria interna, los datos obtenidos. Para ello es necesario almacenar dichos datos en las preferencias de usuario para lo que emplearemos la clase *NSUserDefaults*. Esta clase utiliza, para el guardado de datos, una relación clave-valor, es decir, que a cada clave se le asigna un valor ya sea un NSString, NSinteger o un float como en el ejemplo. Después de realizar la asignación se llama al método "sincronizar"para que se produzca la escritura de los datos en la memoria del dispositivo. Para obtener los datos de la memoria tan sólo hay que hacer una llamada con el método que nos proporciona la llamada para nuestro tipo de dato y el valor de la clave que le asignamos al guardarla.

```
-(void)guardar{
    [[NSUserDefaults standardUserDefaults] setFloat:latiude forKey:cllatitude];
    [[NSUserDefaults standardUserDefaults] setFloat:longitude forKey:cllongitude];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

-(void)obtener{
    latitude=[[NSUserDefaults standardUserDefaults] floatForKey:cllatitude];
    longitude=[[NSUserDefaults standardUserDefaults] floatForKey:cllongitude];

}
```

Figura 3.21: Obtener datos de las preferencias

El uso de base de datos es muy frecuente en la programación, el uso de datos persistentes es muy habitual porque elimina llamadas a servicios web para la obtención de datos, lo que provoca una mayor velocidad de la aplicación y su posible funcionamiento sin conexión a internet. Para la creación de la base de datos se ha utilizado *CoreData*. Es un framework de iOS para el manejo de la bases de datos. Este framework nos proporciona una interfaz visual desde la cual podemos generar las entidades, los atributos de las mismas y definir la relaciones entre las mismas.

Aparte de tener la vista detallada, *CoreData* también ofrece una vista más gráfica, en forma de diagrama, donde se pueden ver las relaciones que comentábamos anteriormente.

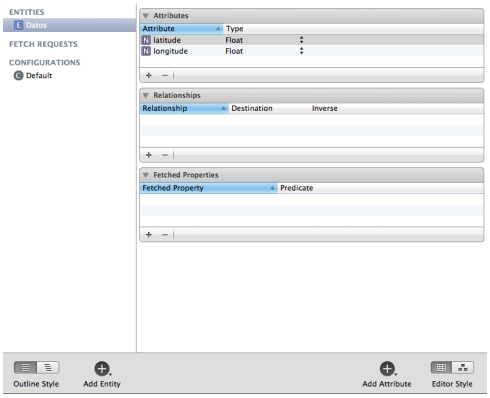


Figura 3.22: Interfaz CoreData

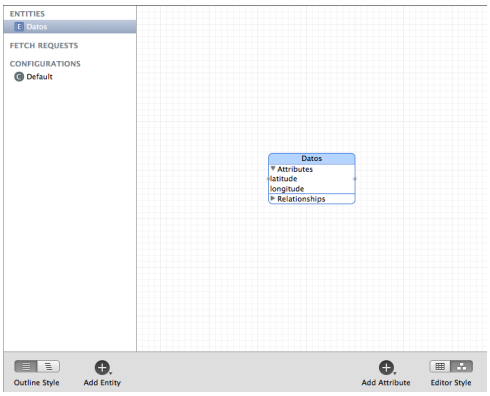


Figura 3.23: Interfaz CoreData II

Para acceder a la base datos que hemos creado con *CoreData* necesitaremos dos atributos en nuestra clase *Appdelegate* de tipo *NSManagedObjectContext y NSManagedObjectModel*. En ellos almacenaremos el contexto donde está situada la base de datos y el objeto de base de datos. Es necesario crear en esta clase estos métodos:

- **applicationDocumentsDirectory:** En este método se le indica el directorio de la aplicación. Es donde se va a realizar la búsqueda de la base de datos.
- **managedObjectModel:** es donde se define el objeto de la base de datos.
- **persistentStoreCoordinator:** es el método que se encarga de coordinar las lecturas/escrituras de la base datos, de tal manera que se asegura que la base de datos se persistente. Aquí se le indica el nombre de la base datos que queremos utilizar.
- **saveContext:** Se utiliza para guardar el contexto. Efectúa la escritura de la base de datos en disco.

- **managedObjectContext:** Es la función que devuelve el objeto de base de datos que está manejado por *persistentStoreCoordinator*

```
- (NSManagedObjectModel *)managedObjectModel
{
    if (__managedObjectModel != nil)
    {
        return __managedObjectModel;
    }
    NSURL *modelURL = [[NSBundle mainBundle] URLForResource:@"database" withExtension:@"momd"];
    __managedObjectModel = [[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL];
    return __managedObjectModel;
}

- (NSPersistentStoreCoordinator *)persistentStoreCoordinator
{
    if (__persistentStoreCoordinator != nil)
    {
        return __persistentStoreCoordinator;
    }

    NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppendingPathComponent:@"wimc.sqlite"];

    NSError *error = nil;
    __persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:[self managedObjectModel]];
    if (![__persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL options:nil error:&error])
    {
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }

    return __persistentStoreCoordinator;
}

- (void)saveContext
{
    NSError *error = nil;
    NSManagedObjectContext *managedObjectContext = self.managedObjectContext;
    if (managedObjectContext != nil)
    {
        if ([managedObjectContext hasChanges] && ![managedObjectContext save:&error])
        {
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
            abort();
        }
    }
}
```

Figura 3.24: Creación de objetos bbdd iOS

```
- (NSManagedObjectModel *)managedObjectModel
{
    if (__managedObjectModel != nil)
    {
        return __managedObjectModel;
    }
    NSURL *modelURL = [[NSBundle mainBundle] URLForResource:@"database" withExtension:@"momd"];
    __managedObjectModel = [[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL];
    return __managedObjectModel;
}

- (NSPersistentStoreCoordinator *)persistentStoreCoordinator
{
    if (__persistentStoreCoordinator != nil)
    {
        return __persistentStoreCoordinator;
    }

    NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppendingPathComponent:@"wimc.sqlite"];

    NSError *error = nil;
    __persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:[self managedObjectModel]];
    if (![__persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL options:nil error:&error])
    {
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }

    return __persistentStoreCoordinator;
}

- (void)saveContext
{
    NSError *error = nil;
    NSManagedObjectContext *managedObjectContext = self.managedObjectContext;
    if (managedObjectContext != nil)
    {
        if ([managedObjectContext hasChanges] && ![managedObjectContext save:&error])
        {
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
            abort();
        }
    }
}
```

Figura 3.25: Creación de objetos bbdd iOS II

Estos métodos los usaremos para manejar los objetos de base de datos que hemos creado pero para realizar inserciones y otro tipo de acciones es necesario utilizar otros métodos. Para ello es necesario en el caso de la inserción hay que instanciar un objeto de

tipo *Datos*. Este objeto ha sido generado automáticamente por *CoreData* y lo obtendremos con *managedObjectContext*. Una vez hayamos almacenado en este objeto los datos llamaremos al método de Appdelegate save para guardar los datos en la base de datos.

```
Datos *data = (Datos *)[NSEntityDescription insertNewObjectForEntityForName:@"Datos"
    inManagedObjectContext:appDelegate.managedObjectContext];

data.latitude=[NSNumber numberWithFloat:latitud];
data.longitude=[NSNumber numberWithFloat:longitud];

NSError *error = nil;

if (![appDelegate.managedObjectContext save:&error]) {
    // Handle the error.
    NSLog(@"un error al guardar: %@", error);
    abort();
}
```

Figura 3.26: Inserción en bbdd iOS

Para obtener los datos de la base de datos, al igual que para la inserción hay que obtener el objeto *managedObjectContext* y en la variable request indicarle el tipo de query que deseamos hacer. Posteriormente los resultados serán devueltos en un *NSMutableArray* .Appdelegate save para guardar los datos en la base de datos.

```
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
NSFetchRequest *request = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"Datos"
    inManagedObjectContext:appDelegate.managedObjectContext];
[request setEntity:entity];
NSError *error = nil;
NSMutableArray *mutableFetchResults = [[appDelegate.managedObjectContext
    executeFetchRequest:request error:&error] mutableCopy];
```

Figura 3.27: Select bbdd iOS

Una parte muy importante de la geolocalización es representar estos datos en el dispositivo. Para ello, en la mayoría de los casos, se utilizan mapas. En iOS el uso de estos es bastante sencillo; la clase que utilicemos deberá heredar de $<MKMapViewDelegate, CLLocationManagerDelegate>$ La primera es para el uso de los mapas y, la segunda, como hemos visto en los puntos anteriores, para el uso de la geolocalización. Debemos declararnos un atributo de tipo *MKMapView*, que es la que almacena el mapa. Este atributo lo deberemos enlazar con el storyboard, al igual que hacemos con los botones. Aparte de éste, es necesario declarar un *NSSMutableArray* para almacenar los marcadores y luego los atributos, donde se almacenaran la latitud y longitud. Todo esto, como hemos visto anteriormente, se declara en el archivo de cabecera.

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

@interface MapViewController : UIViewController <MKMapViewDelegate,CLLocationManagerDelegate>{
    IBOutlet MKMapView *mapa;
    float latitud;
    float longitud;
    CLLocationManager *locationManager;
    NSMutableArray *annotations;


}

@property (nonatomic, retain) NSMutableArray *annotations;
@property (nonatomic, retain) MKMapView *mapa;
@property (nonatomic,assign) float latitud;
@property (nonatomic,assign) float longitud;
@property (nonatomic, retain) CLLocationManager *locationManager;
@property (nonatomic,retain) NSMutableArray *_path;
- (IBAction)btRoute:(id)sender;
- (IBAction)btlastParkings:(id)sender;
```

Figura 3.28: Declaración mapa iOS

Para añadir un marcador hay que declarar una variable de tipo *MKPointAnnotation*, la cual le asignaremos unas coordenadas con una variable de tipo *CLLocationCoordinate2D*, a la cual asignaremos los valores de latitud y longitud que tenga nuestro vehículo primero, el título que queramos que tenga el marcador y posteriormente un subtítulo, que en este caso serán los valores de latitud y longitud. Tras haber rellenado los campos que deseamos del marcador, lo añadiremos al mapa con la función *addAnnotation*.

```
CLLocationCoordinate2D theCoordinate;
theCoordinate.latitude= latitud0;
theCoordinate.longitude= longitud0;

MKPointAnnotation *myAnnotation = [[MKPointAnnotation alloc] init];
myAnnotation.coordinate=theCoordinate;
if (isCar)
    myAnnotation.title= NSLocalizedString(@"coche", @"");
else
    myAnnotation.title= NSLocalizedString(@"tu posicion", @"");
myAnnotation.subtitle = [NSString   stringWithFormat:@"%f %f",myAnnotation.coordinate.
    latitude, myAnnotation.coordinate.longitude];

[mapa addAnnotation:myAnnotation];
```

Figura 3.29: Anadir un marcador iOS

Para el pintado de la ruta es necesario obtener los puntos de un servicio web. Para ello, en primer lugar, veremos cómo se implementa una llamada asíncrona en iOS. La llamada asíncrona consiste en no dejar la aplicación bloqueada a la espera de la respuesta del servicio web, para lo que se le proporciona la dirección del servicio web a la cual queramos llamar. En este caso es *http://maps.googleapis.com/*. Posteriormente, se le añadirán el resto de parámetros como la latitud y la longitud, origen y destino mediante un método POST. Si la petición se ha realizado con éxito, se ejecutará el método, *parseResponse* se encargará de parsear la respuesta y transformarlo en un polyline para que el mapa lo pueda pintar.

```
AFHTTPClient *_httpClient = [AFHTTPClient clientWithBaseURL:[NSURL
                                                    URLWithString:@"http://
                                                    maps.googleapis.com/"]];
[_httpClient registerHTTPOperationClass: [AFJSONRequestOperation class]];
NSMutableDictionary *parameters = [[NSMutableDictionary alloc] init];
[parameters setObject:[NSString stringWithFormat:@"%f,%f",
                      latitudOrigen,longitudOringen]
               forKey:@"origin"];
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];

[self pintar:appDelegate.getlatitude longitudO:appDelegate.getlongitude coche:true];
[parameters setObject:[NSString stringWithFormat:@"%f,%f",
                      appDelegate.getlatitude   ,appDelegate.getlongitude]
               forKey:@"destination"];
[parameters setObject:@"true" forKey:@"sensor"];
NSMutableURLRequest *request = [_httpClient requestWithMethod:@"GET" path:
                               @"maps/api/directions/json" parameters:parameters];
request.cachePolicy = NSURLRequestReloadIgnoringLocalCacheData;
AFHTTPRequestOperation *operation = [_httpClient
                                    HTTPRequestOperationWithRequest:request
                                    success:^(AFHTTPRequestOperation *operation, id
                                        response) {
                                        NSInteger statusCode = operation.response.
                                            statusCode;
                                        if (statusCode == 200) {
                                            [self parseResponse:response];
                                        } else {
                                        }
                                    } failure:^(AFHTTPRequestOperation *operation, NSError
                                        *error) { }];
[_httpClient enqueueHTTPRequestOperation:operation];
```

Figura 3.30: Llamada asincrona iOS

Lo último que queda por ver del desarrollo de la aplicación de prueba es la navegación entre pantallas. Para realizar esta navegación es tan sencillo como ir al storyboard, pulsar la tecla control sobre un botón declarado en la interfaz y arrastrar la línea resultante hasta la pantalla donde queramos navegar.

Figura 3.31: Transición de pantallas iOS

Para el funcionamiento de la localización, el uso de Internet y CoreData, es necesario añadir una serie de *framworks* para poder heredar de las clases anteriormente mencionadas. Sin insertar estos, se generará un error de compilación.

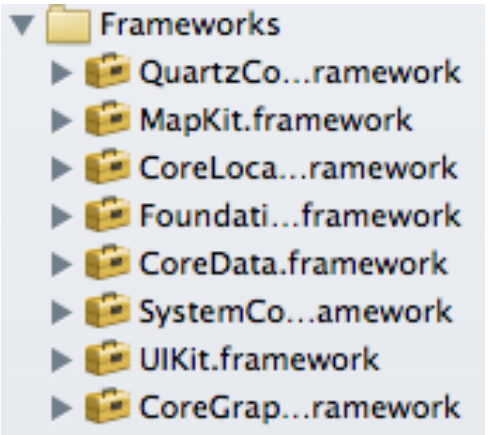


Figura 3.32: Frameworks iOS

A la hora de desarrollar la aplicación es muy importante tener en cuenta cómo generar ésta en varios idiomas. En el caso de iOS, lo que se utiliza para localizar los textos es un archivo de recursos llamado *localizable.strings*. En este archivo meteremos los strings, se generará un archivo con una extensión diferente para cada idioma que queramos insertar. En el caso de las imágenes es igual pero, de manera gráfica, hay que pulsar en la imagen y luego en make localizable.

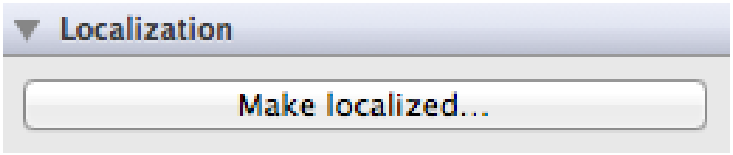


Figura 3.33: Localizar imágenes iOS

Por último, la subida de la aplicación en el App Store, debe cumplir unos requisitos que son dos iconos: uno de 1024x1024 y otro de 512x512 para el lanzador de la aplicación, las capturas de pantalla y los textos de descripción que aparecerán en la tienda.

3.5.2. Implementación Android

En este apartado veremos cómo se desarrolla sobre la plataforma de Google. Hay que tener en cuenta que, como hemos visto previamente, el lenguaje de programación es Java. Este lenguaje estructura el código en clases e inserta una serie de librerías que nos dan soporte para poder desarrollar aplicaciones en Android. También hay varios plugin en .Net que permiten la programación en esta plataforma. Para empezar a programar hay que instalar Eclipse y su correspondiente plugin que lo que hace es instalar el SDK y el ADB. El SDK lo que hace es instalar las librerías y el ADB instala el simulador para que podamos probar nuestras aplicaciones sin necesidad de tener un terminal. Nada más empecemos a realizar una aplicación Android, la parte del código más importare es el *AndroidManifest.xml*. En este fichero se definen los permisos que se van a utilizar, que serán los necesarios para la conexión a Internet, el acceso a la localización y el uso de mapas. En este archivo también se especifican las versiones de Android, el nombre de la aplicación, la actividad principal y el resto de actividades. Las actividades que aparezcan en la aplicación y no estén declaradas en el manifest no se ejecutarán, darán un error de ejecución. El manifest será el encargado de comunicar al usuario lo que la aplicación necesita para funcionar.

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="es.urjc.whereismycar.permission.MAPS_RECEIVE" />
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Figura 3.34: Declaración de manifest Android

```
<permission
    android:name="es.urjc.whereismycar.permission.MAPS_RECEIVE"
    android:protectionLevel="signature" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar" >
    <activity
        android:name="es.urjc.whereismycar.MainActivity"
        android:label="@string/app_name"
        android:screenOrientation="portrait" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name="es.urjc.whereismycar.MyCarActivity"
        android:screenOrientation="portrait" />

    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="AIzaSyArPTcBrxo0KhhPl9Js3x1WmytdVT2QSE4" />
```

Figura 3.35: Declaración de manifest Android II

Una vez visto cómo se declaran los permisos de la aplicación y la actividad principal, se enseñará cómo se realiza la declaración de los botones. Para declarar un botón en Android, aparte de como lo hemos visto en el apartado de diseño, en el respectivo de interfaz mediante un *Button*, es necesario declararlo en el código para su uso. Lo primero que haremos será declararlo como un atributo de la clase donde indicaremos el nombre de la variable que queremos darle al botón y, luego, el tipo de dato.

```
public class MainActivity extends Activity implements OnClickListener {
    private Button btWhereIs;
    private Button btIamHere;
```

Figura 3.36: Declaración botón Android

No sólo basta con declarar el botón, sino que también deberemos asignar el método que se lanzará al pulsar encima del botón. En esta clase hemos decidido que se la misma clase la que maneje los eventos de pulsado haciendo que implemente los métodos. Esto se hará utilizando la función *onclick*. Se capturarán todas las acciones de los botones en los que hayamos asignado la clase misma en el método *onClickListener(this)* y, posteriormente en la función, controlaremos qué vista ha activado el evento y haremos con ella la acción que sea necesaria.

```
@Override
public void onClick(View v) {

    switch (v.getId()) {
    case R.id.button2:
        Intent intent = new Intent(this, MyCarActivity.class);
        startActivity(intent);
        break;

    case R.id.button1:
        setCurrentLocation();
        break;

    default:

        break;
    }

}
```

Figura 3.37: Declaración onClickListener

Después de declarar los dos botones de la aplicación de prueba nos disponemos a analizar cómo se declara un método en Android. Lo primero que hay que decidir es si el método será público o privado; en caso de ser un método público, bastará con poner el

prefijo public y, en el caso de que el método sea privado, deberemos poner “private”, como prefijo, en la cabecera.

```
public void onProviderDisabled(String provider) {
```

Figura 3.38: Declaración de método privado en la interfaz

La declaración del método en la clase es parecida a la de cualquier lenguaje de programación muy similar a la de C.

```
private void setCurrentLocation() {
```

Figura 3.39: Declaración de método público en la clase

Para realizar la llamada a un método es tan sencillo como escribir el nombre de la función y meter entre paréntesis las variables necesarias. En caso de que la función esté en otra clase, el procedimiento será el mismo pero con la variable seguida de un punto y luego el nombre de la función.

```
setCurrentLocation();
```

Figura 3.40: Llamada función Android

Una vez que se declaran los métodos en el programa, estudiaremos el uso del GPS en la aplicación. Para implementar esta funcionalidad es necesario crearnos una clase, ya sea local o en un fichero aparte. Esta clase heredará de *LocationListener* la cual añade unos métodos, donde se disparan, dependiendo de los datos que reciba del sensor GPS. Crearemos una variable de tipo *MyLocationListener* que será de la cual leeremos los datos de latitud y longitud. El método donde se controla la actualización de la posición mediante el GPS es:

```
public void onLocationChanged(Location argLocation) {
    // TODO Auto-generated method stub
    CurrentLatitude = argLocation.getLatitude();
    CurrentLongitude = argLocation.getLongitude();
}
```

Figura 3.41: Método de lectura de posición Android

Este método se lanza sólo si ha habido un cambio de posición. En el caso que este cambio se dé, la nueva posición estará en la variable *argLocation*. Asignaremos los valores de latitud y longitud de dicha variable a los atributos de la clase *currentLatitude* y currentLongitude. Tras obtener la posición, el siguiente paso es ver cómo se almacena en la memoria interna los datos obtenidos. Para ello, es necesario almacenar dichos datos en las preferencias de usuario, para lo cual, utilizaremos la clase *Preferences*. Esta clase se ha creado para tener todas las llamadas a las preferencias en un mismo sitio, hemos declarado los métodos como estáticos para así no tener que declarar la variable para que podamos utilizarlos tan solo haciendo referencia a la clase. El método de almacenamiento se realiza mediante clave-valor, donde se le asignará una clave a cualquier valor que queramos almacenar u obtener. En preferencias podemos guardar tipos primitivos de datos. Estas acciones se hacen mediante los métodos *getSharedPreferences* donde obtenemos las preferencias y *.getFloat* donde obtenemos un float y un valor por defecto, en caso de que no haya nada almacenado.

```
public static float getLongitude(Context myContext) {
    if (longitude == 0) {
        SharedPreferences settings = myContext.getSharedPreferences(
                PREFS_NAME, 0);
        longitude = settings.getFloat("longitude", -1);
    }
    return longitude;
}

public static void setLongitude(Context myContext, float vlongitude) {


    try {
        SharedPreferences settings = myContext.getSharedPreferences(
                PREFS_NAME, 0);
        SharedPreferences.Editor editor = settings.edit();
        editor.putFloat("longitude", vlongitude);
        editor.commit();
    } catch (Exception e) {
        Log.e("ERROR", e.getMessage());

    }

}
```

Figura 3.42: Datos preferencias Android

Tras el almacenamiento en las preferencias, otra característica de la aplicación, al igual que en iOS, es el almacenamiento en base de datos. Para este propósito, se utilizará una base de datos sqlite. Por este motivo, crearemos una clase llamada *DbHandler* donde generaremos las tablas y llamaremos para hacer las inserciones, selects y borrados de la base de datos. Para la creación de la base de datos es tan sencillo como hacer esta llamada:

```
private static final String DATABASE_NAME = "whereIsmycarDb";
private static String TABLE_POSITIONS = "tpositions";
private static String CREATEDB = "Create table " + TABLE_POSITIONS
        + " (latitude double ,longitude double)";

public DbHandler(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
    // TODO Auto-generated constructor stub
}

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREATEDB);
    // TODO Auto-generated method stub

}
```

Figura 3.43: Creación base de datos Android

Para almacenar las posiciones en la base de datos sólo hay que declarar una variable de

tipo *DbHandler* y llamar al método add.

```
db.addPosition(CurrentLatitude, CurrentLongitude);
```

Figura 3.44: Insertar posición Android

Para obtener los puntos almacenados de la base de datos sólo hay que declarar la variable como para añadirlos y llamar al método *getLastParking*. Este método nos devolverá un *ArrayList* de *LatLng* donde obtendremos la latitud y la longitud.

```
ArrayList<LatLng> parking = db.getLastParking(CurrentLatitude,
        CurrentLongitude, 5);
```

Figura 3.45: Obtener posiciones base de datos Android

Para mostrar el mapa, primero tendremos que conseguir un código para que podamos verlo en el terminal. Necesitaremos uno para desarrollo y otro para cuando realicemos la subida al *Google Play Store*. Este código se obtiene en `https://code.google.com/apis/console/`. Una vez se tiene el código hay que introducirlo en el manifest.

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyArPTcBrxo0KhhPl9Js3x1WmytdVT2QSE4" />
```

Figura 3.46: Código mapa manifest Android

Después de haber añadido el código, lo que hay que hacer, inmediatamente, es añadir en la interfaz un fragmento con el mapa. Se debe declarar en el código, asociándolo a un fragmento y, a continuación, obtener el mapa.

```
<fragment
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Figura 3.47: Declaración mapa en la interfaz Android

```
SupportMapFragment fm = (SupportMapFragment) getSupportFragmentManager()
        .findFragmentById(R.id.map);

mapView = fm.getMap();
```

Figura 3.48: Declaración mapa en el código Android

Una vez declarado el mapa, veremos cómo se añaden los marcadores. Estos marcadores se usan para marcar la posición del vehículo y la posición actual. Para añadir un marcador, tan sólo hay que declarar una variable de tipo *Marker* rellenar los campos de *title, position e icon* y añadirlo a la variable *mapView* llamando a la función add.

```
Marker marke2r = mapView.addMarker(new MarkerOptions()
        .position(p)
        .title("Su vehículo")
        .icon(BitmapDescriptorFactory
                .fromResource(R.drawable.marker2)));
```

Figura 3.49: Añadir un marcador Android

Para el cálculo de la ruta es necesario efectuar una llamada asíncrona, es decir, una llamada que no deje bloqueado el hilo de ejecución. Para realizar esta llamada hay que declararse una tarea asíncrona. En ella efectuaremos la llamada al servicio web y recibiremos los datos que queremos obtener de él. Una vez obtenidos los datos se obtienen de la respuesta y se genera un *polyline* que, posteriormente, se añade al mapa.

```
private class GetRoute extends AsyncTask<Integer, Integer, Document> {

    @Override
    protected Document doInBackground(Integer... params) {
        // TODO Auto-generated method stub
        color = Color.BLUE;
        mMapView01 = mapView;

        LatLng fromPosition = new LatLng(CurrentLatitude, CurrentLongitude);
        LatLng toPosition = new LatLng(carLatitude, carLongitude);

        md = new GMapV2Direction();

        return md.getDocument(fromPosition, toPosition,
                GMapV2Direction.MODE_WALKING);
    }
```

Figura 3.50: Llamada asincrona Android

```
protected void onPostExecute(final Document result) {
    Document doc = result;
    if (doc != null) {
        ArrayList<LatLng> directionPoint = md.getDirection(doc);
        PolylineOptions rectLine = new PolylineOptions().width(3)
                .color(Color.RED);

        for (int i = 0; i < directionPoint.size(); i++) {
            rectLine.add(directionPoint.get(i));
        }
        mapView.clear();
        mapView.addPolyline(rectLine);
        double results = (carLatitude + CurrentLatitude) / 2;

        double resultlo = (carLongitude + CurrentLongitude) / 2;

        LatLng pos = new LatLng((double) Preferences.getLatitude(act),
                (double) Preferences.getLongitude(act));

        Marker marker = mapView.addMarker(new MarkerOptions()
                .position(new LatLng(carLatitude, carLongitude))
                .title("Su vehículo")
                .icon(BitmapDescriptorFactory
                        .fromResource(R.drawable.marker2)));

        Marker marke2r = mapView.addMarker(new MarkerOptions()
                .position(pos)
                .title("Su posición")
                .icon(BitmapDescriptorFactory
                        .fromResource(R.drawable.marker2)));

        mapView.animateCamera(CameraUpdateFactory.newLatLngZoom(
                new LatLng(results, resultlo), 6.0f));

    }
```

Figura 3.51: Llamada asincrona Android II

A la hora de la navegación entre pantallas en Android lo que se utilizan son *Intent*. Éstos reciben, como parámetro, la actividad y la clase a la cual queramos transitar. Se pueden pasar parámetros en estos mediante el uso de extras.

```
Intent intent = new Intent(this, MyCarActivity.class);
startActivity(intent);
```

Figura 3.52: Transición pantallas Android

A la hora de localizar la aplicación en Android hay que declararse el archivo donde se declaran los strings en la carpeta *values*. Cada una de ellas, con la extensión del idioma. A la hora de llamarlo basta con realizar una llamada al método *getStringFromResource*. Para el caso de las imágenes es similar tan sólo hay que crearse una carpeta con la resolución y el sufijo del idioma que queramos localizar. Android solo escogerá la imagen según el idioma y el tamaño del dispositivo.

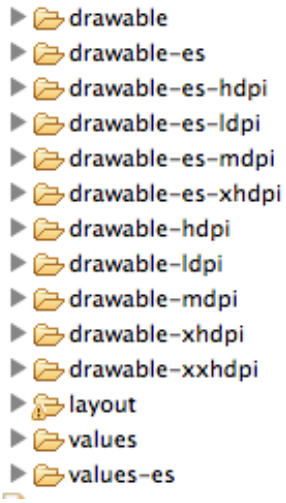


Figura 3.53: Localizacion Android

Para la subida en el Google Play Store tan sólo es necesario el archivo de la aplicación, el icono de la aplicación en 512 x 512 una imagen destacada en 1024 x 500, una imagen promocional 180 x 120 y, al menos, dos capturas de pantalla 320 pixeles como mínimo en los laterales. Esto, unido a los textos de la aplicación, hará que la misma aparezca en pocas horas en el Google Play Store.

3.6. Tecnologías utilizadas

Para la realización del trabajo, las tecnologías utilizadas han sido Java, XML, Objetive C. El compilador de Java que se ha usado es Eclipse en su versión 3.8.0 y Android Developer Tools versión 21.1.0. El compilador de Objective C ha sido xCode en su versión 4.5.2. Se han decidido estos porque son los que se usan habitualmente para los proyectos. Respecto a las librerías utilizadas en el proyecto son:

iOS

- **Afnetworking** (versión 1.0): librería utilizada para realizar las tareas asíncronas. Se utiliza en prácticamente todos los proyectos de iOS en el mercado por ser fácil de usar. Aparte de realizar llamadas asíncronas de tipo *POST* y GET también realiza cacheo de imágenes.
- **CoreData**: se utiliza para el manejo de bases de datos. Implementa una interfaz gráfica y permite que los accesos a la base de datos sean simples.

Android

- **android-support-v-4** (versión 4.0): librería utilizada para la compatibilidad de elementos de la version 4 de android con las anteriores.

Software del trabajo También se ha utilizado el siguiente software para la realización del trabajo.

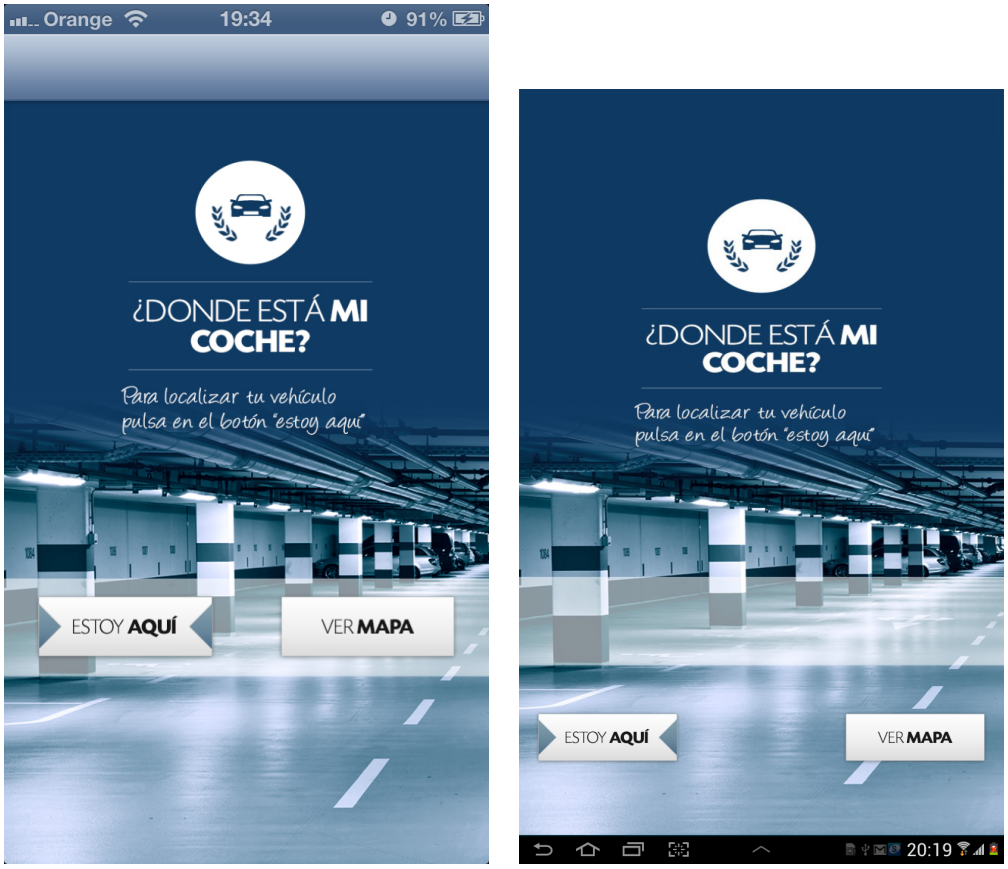
- **MagicDraw UML** (version 16.8): herramienta que se ha utilizado para la generacion de los diagramas del proyecto.
- **TextShop** (versión 3.11): editor de latex con el cual se ha realizado la memoria.
- **Git**(versión 1.8.3.1): herramienta para el control de versiones.

3.7. Evaluación

En esta sección veremos la evaluación del proyecto, es decir, comprobaremos si su funcionamiento es correcto. Lo haremos a través de ejemplos gráficos y ejemplos analíticos que nos ayudarán a comprender y verificar que el simulador funciona correctamente.

3.7.1. Ejemplos gráficos

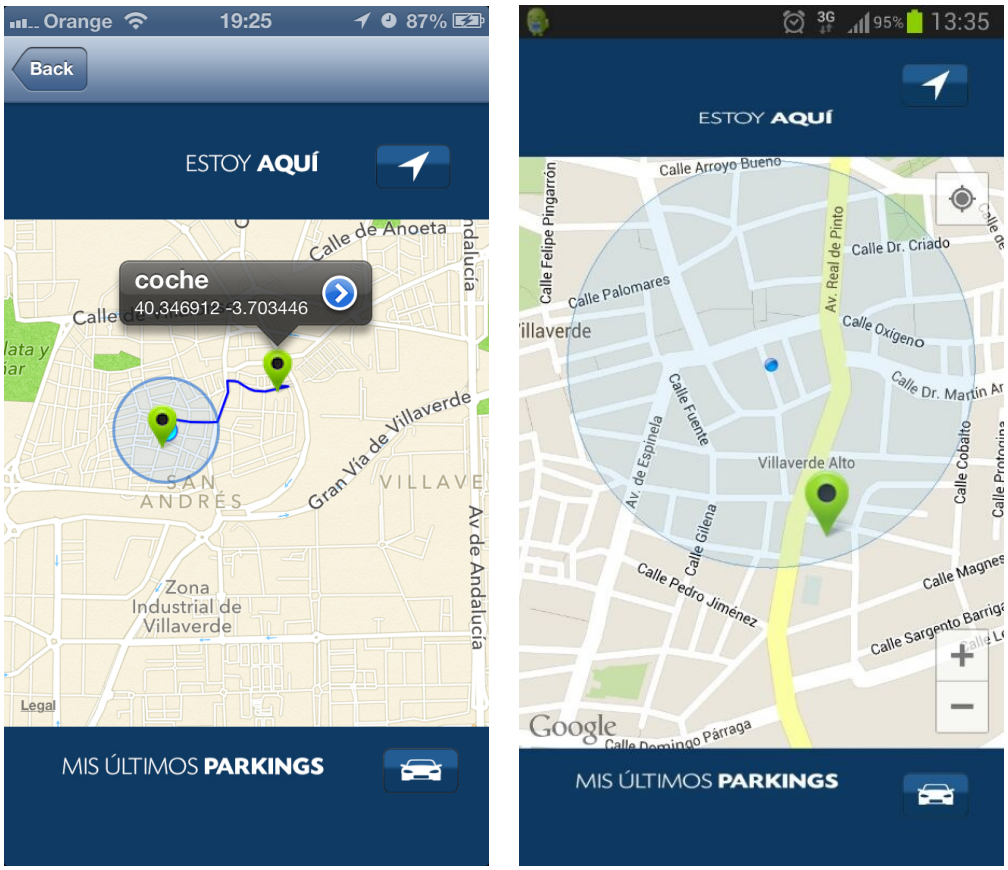
A continuación, podremos ver las capturas de pantalla de las aplicaciones iOS y Android. En la primera de ellas, se puede observar cómo son las interfaces de usuario en ambas plataformas. Coincidiendo con las pantallas vistas en el prototipo pero con el diseño final incluido. Podemos ver cómo están los botones *estoy aquí* y *ver mapa*.



(a) (b)

Figura 3.54: Primera pantalla

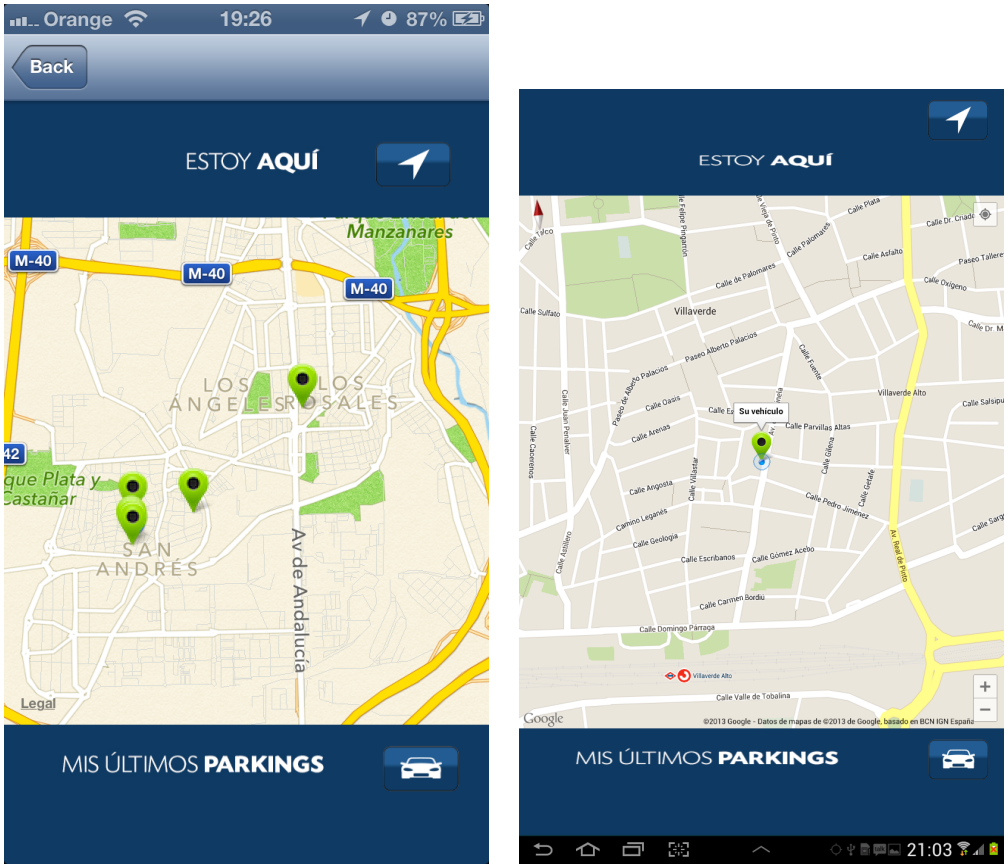
Después de ver la primera pantalla, aquí se puede ver la funcionalidad de la segunda, mostrando la ruta donde se encuentra el vehículo y, en caso de que no exista una ruta posible, se puede ver que no muestra nada.



(a) (b)

Figura 3.55: Segunda pantalla

Y, por último, se muestran los últimos parkings cercanos. Esta acción se realiza obteniendo los puntos de la base de datos y elimando del listado los que no cumplan con la condición de que la distancia sea menor de 5 kilómetros.



(a) (b)

Figura 3.56: Segunda pantalla parkings

3.7.2. Ejemplo analítico

En esta parte veremos cómo se comporta la aplicación basada en los datos que proporciona el sensor GPS.

latitud	longitud
40.306.740	-3.732.234
40.349.430	-3.817.212
40.344.341	-3.709.556
40.344.765	-3.709.776
40.344.654	-3.709.795
40.355.759	-3.742.676
40.390.682	-3.628.417
40.344.570	-3.709.739
40.346.657	-3.709.529
40.291.264	-3.743.719
40.354.977	-3.692.331
40.354.977	-3.692.331
40.322.632	-3.711.977
40.322.720	-3.712.059
40.346.912	-3.703.446
40.214.367	-3.467.196
40.235.985	-3.476.629

Figura 3.57: Tabla de todas las posiciones almacenadas

Éstas serían todas las posiciones almacenadas en la base datos que cumplen el requisito de que la distancia sea menor a 5 kilómetros. Como podemos observar estos datos lo confirman. El cálculo de la distancia se realiza usando la siguiente fórmula :

lat1=latitud orginen.

lat2=latitud destino.

long2 = longitud origen.

lat2 = longitud destino.

dLat = lat2-lat1

dLon = lon2-lon1

a= $\sin(dLat/2) * \sin(dLat/2) + \cos(lat1) * \cos(lat2) * \sin(dLon/2) * \sin(dLon/2)$

$c = 2 * \arctan(\sqrt{a}, \sqrt{1-a})$

$R = 6371$

$distancia = R * c * 10$

latitud	longitud	distancia
40.306.740	-3.732.234	4.970.668
40.344.341	-3.709.556	0.050055
40.344.765	-3.709.776	0.003741
40.344.654	-3.709.795	0.010382
40.355.759	-3.742.676	3.902.175
40.344.570	-3.709.739	0.019014
40.346.657	-3.709.529	0.217691
40.354.977	-3.692.331	2.272.778
40.322.632	-3.711.977	2.503.041
40.322.720	-3.712.059	2.494.145
40.346.912	-3.703.446	0.750274

Figura 3.58: Tabla de todas las posiciones cercanas

Se puede ver que, cada punto que aparece en el mapa, es una de la posición que se ve en la tabla anterior. Después de ver estos datos verificamos que la aplicación funciona como esperábamos ya que, las marcas en la pantalla, coinciden con los datos obtenidos de la tabla.

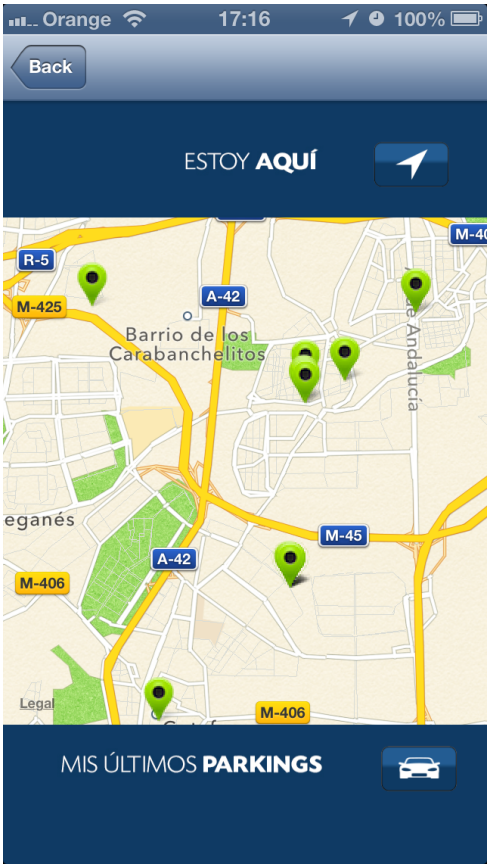


Figura 3.59: Parkings

Capítulo 4

Conclusiones

En este apartado se expondrán las conclusiones del proyecto, se explicará cómo se han alcanzado los objetivos, se realizará una pequeña evaluación personal sobre el proyecto, donde se intentará plasmar la experiencia de realizar un proyecto de estas características y, por último, se mostrará el apartado de líneas futuras donde se exponen posibles futuras mejoras.

4.1. Objetivos cumplidos

Llegando a este punto se han de definir si todos los objetivos, que se marcaron en el comienzo del proyecto, se han llevado a cabo. Para ello, iremos repasando uno a uno esos objetivos e iremos comentando la forma en que se han cumplido, durante la memoria. En primer lugar, hemos podido ver cómo se optó por el modelo de prototipo para iniciar el proyecto. Se han creado los prototipos y los diagramas correspondientes para su funcionamiento. En segundo lugar, una vez hecho el prototipo, se desarrollaron las aplicaciones en las dos plataformas y, en el aparto de implementación, se explicaron las características más importantes de cada una. También, se vieron desde la declaración de los métodos hasta el ciclo de vida de la aplicación, terminando con la subida a sus respectivos stores, comprobando, con datos reales, que las aplicaciones funcionan correctamente. En tercer lugar, en lo que respecta a la integración continua, hay que destacar que, para el desarrollo de todo el proyecto, se ha usado el sistema de control de versiones Git y los proyectos tienen creados un repositorio en `https://github.com/buitren9/whereismycar`

y `https://github.com/buitren9/whereismycariOS` porque el apartado de control de versiones, también está cumplido. Por último, la comparación de desarrollo se verá en el siguiente punto, por lo que podremos dar todos los objetivos por cumplidos.

4.2. Evaluación personal

Después de observar los requisitos de la aplicación, su prototipado y su implementación, en este punto se va a realizar una comparativa en el desarrollo de esta aplicación en ambas tecnologías. Para que sea más sencillo de entender dividiremos esta comparativa en tres partes:

4.2.1. Diseño y prototipo

En este aspecto los pasos a seguir por las dos tecnologías son idénticos, ya que para ambas hay que hacer un análisis de requisitos, tanto el modelo de desarrollo como el diagrama de base de datos son los mismos. Tan sólo hay que tener en cuenta las limitaciones de cada plataforma a la hora de desarrollarlas, debido a que existen muchas funcionalidades que en Android se pueden desarrollar y en iOS no. Un ejemplo: a la hora de compartir archivos, fotos y música por bluetooth, en el sistema operativo de Apple no se puede realizar. Aparte de estas limitaciones técnicas, también existen unas limitaciones a nivel de diseño ya que ambas plataformas tienen sus propias guías de diseño que deben respetarse, aunque, en el caso de Android no sea restrictivo, en el caso de iOS sí que puede ser un motivo para que la aplicación no supere el proceso de revisión del App Store.

4.2.2. Implementación

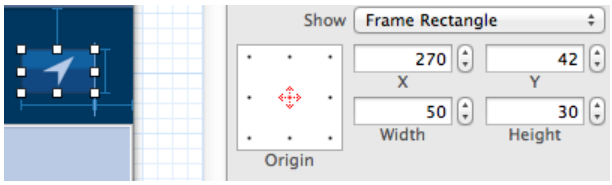
En cuanto al nivel de implementación hay que destacar que la interfaz gráfica es mucho más sencilla de desarrollar en iOS, ya que sólo tenemos que implementar esta interfaz para dos dimensiones de pantalla. El StoryBoard permite que esto sea más fácil ya que tan solo hay que arrastrar a la vista los botones, y situarlos en el lugar que queramos, estos mantendrán el mismo aspecto cuando carguemos la aplicación en todos los terminales. En el caso de Android la interfaz debe ser visible para las todas las posibles pantallas y resoluciones. Este inconveniente nos fuerza al uso de posiciones relativas para mantener el mismo aspecto en los dispositivos lo que hace que aumente la dificultad a la hora de implementarlo ya que no basta con coordenadas, hay que posicionar los elemento en relación a otros elementos o a posiciones en la vista: centrado, arriba, abajo, etc. Por otra parte, el diseño se realiza mediante ficheros XML, ya que si se realiza por la interfaz

gráfica que proporciona Eclipse introduce código innecesario que hace que las interfaces vayan más lentas. Por consiguiente, en el tema de diseño de interfaces es mucho mejor iOS que Android.

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="3"
    android:background="@drawable/fondoimhere" >

    <ImageView
        android:id="@+id/boton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_marginRight="26dp"
        android:layout_marginTop="18dp"
        android:adjustViewBounds="true"
        android:clickable="true"
        android:src="@drawable/boton_geo" />
</RelativeLayout>
```

(a)



(b)

Figura 4.1: Posicionamiento de botones

Centrándonos más en el código y observando una a una las llamadas, los métodos y la creación de la base de datos desde el punto de vista del programador, Android, al estar hecho en Java, es más familiar. Esto es debido a que, actualmente, todos los desarrolladores parten de este lenguaje para iniciarse en el mundo de la programación. Esto unido a que la declaración de las variables y de los métodos se realizan en el mismo fichero, hace que sea más cómodo de utilizar.

Otro aspecto negativo para iOS es el lenguaje. Objetive C es un lenguaje complejo ya que el continuo uso de punteros y de referencias muchas veces resulta muy engorroso a la hora de iniciarse en la programación, sobre todo si no se está acostumbrado a trabajar en C. Otro de los inconvenientes de este lenguaje es que, pese a parece C++, las llamadas y la declaración de los métodos son totalmente diferentes. También los tipos de datos son diferentes ya que no se utilizan Strings y en cambio se usan NStrings. Los tipos de datos

de iOS sólo se usan en este lenguaje, y como todo está basado en objetos, el concepto de array que se tiene de programación en C desaparece y se crea un nuevo tipo de dato. Este nuevo dato es el *NSMutableArray* que es un array genérico para cualquier dato. El acceso a éstos se realiza de una forma más lenta y menos intuitiva que en Java ya que para acceder a un elemento en vez de usar el método *get* y una posición, se utiliza *objectAtIndex* lo que desvirtúa un poco su uso.

Otro factor muy importante es que la comunidad de usuarios es mucho más amplia en Android que en iOS, ya que la primera está soportada por Java, que es un lenguaje muy común entre desarrolladores y sin embargo xCode no tiene tanto apoyo. También es cierto que, las herramientas que proporciona xCode para el desarrollo son bastante más potentes que las que proporciona Eclipse. Es verdad que con el nuevo editor que ha lanzado Google, *Android Studio*, esta diferencia se recorta, aunque no demasiado, ya que al ser una primera versión aún tiene que mejorar. En el aspecto de la interfaz gráficas sí que ha mejorado ya que permite modificar los layouts y verlos al mismo tiempo. xCode es un editor muy potente y las principales ventajas es que se pueden ver los ViewController según los estas desarrollando. Del mismo la interfaz gráfica de Coredata simplifica mucho la creación de las tablas aunque ralentiza la creación de las mismas. Tras ver las ventajas e inconvenientes de este apartado podemos afirmar que el desarrollo en Android es mucho más fluido debido a la sencillez del lenguaje y debido a ello su aprendizaje requiere menos dificultad.

4.2.3. Despliegue

En este caso, gracias a la no existencia del proceso de revisión, Android es mucho mejor que iOS, ya que la aplicación puede ser muy buena, pero si no se ajusta a las normas de estilo de Apple, si tarda en cargar más de diez segundos, si hay aplicaciones parecidas, si es dependiente siempre de la conectividad a Internet o si no realizas las compras mediante su forma de pago (App-purchase) pueden ser motivo por el cual tu aplicación no pase este proceso. Este porcentaje de rechazo está en torno al 20 % en iOS y el 1 % en Android. Los inconvenientes son que en Android al no darse esta restricción puedes encontrarte muchas aplicaciones con poca funcionalidad que no funcionen bien en el Google Play Store. Aún con ello, el usuario es el último responsable de la instalación de la aplicación y para evitar

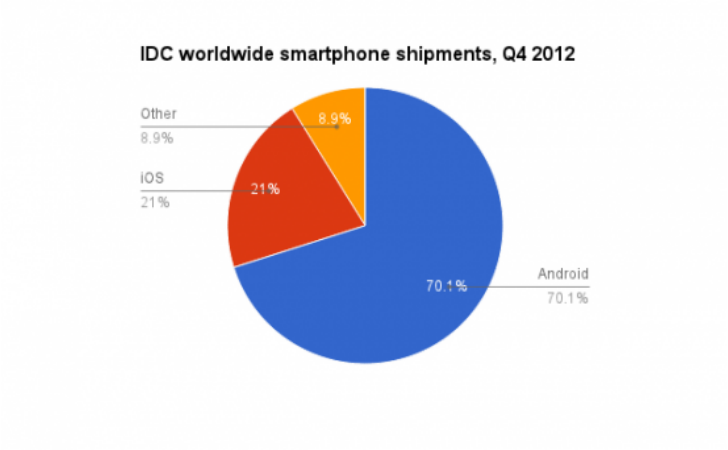
bajarse una aplicación que no funcione bien, están las puntuaciones y opiniones en la página de descarga.

Otra desventaja que tiene iOS respecto a Android es la distribución de las aplicaciones. En el caso de Android se pueden distribuir sin pasar por el Google Play Store tan sólo distribuyendo el ejecutable de la aplicación, en el caso de iOS esto no se puede hacer ya que solo existe el medio de distribución ad-hoc y para ello es necesario proporcionar el UID del dispositivo, darlo de alta en la cuenta de desarrollador de Apple y luego distribuir el ejecutable. Por último hay que destacar el precio de la licencia de desarrollador en Android, se trata de una licencia de por vida de 25$ y en el caso de iOS es de 99$, renovable anualmente.

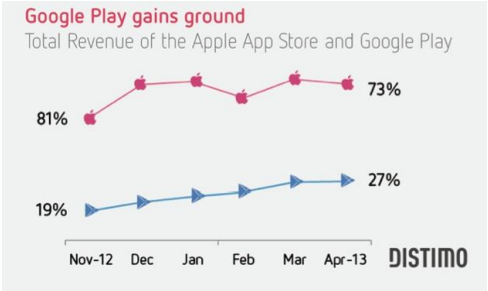
4.2.4. Conclusión

Haciendo un repaso de todo el trabajo, como conclusión, podemos decir que para un desarrollador que esté empezando y quiera crear sus propias aplicaciones sin tener que hacer una gran inversión inicial, la plataforma que mejor se ajusta a este perfil es Android. En primer lugar, porque su licencia de desarrollo tan sólo cuesta 25$ y, a nivel de hardware, con un ordenador común es suficiente para instalar el software necesario.

En segundo lugar, porque se pueden encontrar terminales para el uso de pruebas desde 89,99; en el caso de iOS un Iphone tiene un coste de 389. A nivel de venta de dispositivos el último trimestre de 2012 Android vendió el 70 % del mercado lo que significa que habrá más usuarios de Android que de iOS. Esto se une a que el número de aplicaciones en el Google Play Store está creciendo exponencialmente estos últimos años. Hay ya 600.000 en el Google Play Store y 700.000 en App Store; esta diferencia hace 3 años era abismal pero en la actualidad se encuentran prácticamente al mismo nivel. En tercer lugar, respecto a la ganancia con la venta de aplicaciones, en ambos markets el desarrollador sólo percibe el 70 % de la venta de cada aplicación y los usuarios de iOS están más habituados a la hora de pagar por sus apps y esto refleja que los ingresos de iOS sean mucho mayores que los de Android. También es cierto que los números de Android están creciendo muy rápido.



(a)



(b)

Figura 4.2: Ventas de Smartphones y ganancias

A nivel de desarrollo, para la realización de la aplicación iOS, se ha tardado 25 días y 22 para la Android. Hay que tener en cuenta que mis conocimientos de iOS y Android eran de nivel básico, lo que me ha permitido desarrollar de manera más rápida las aplicaciones profundizado en varios conceptos que desconocía de los lenguajes. Como se ha visto en el punto de implementación, en el caso de iOS es más complejo de aprender ya que, al usar Objetive C, y desconocer totalmente este lenguaje ya que no se estudia en la universidad. En Android, al estar basado en Java, su sintaxis es más conocida y, con sólo aprender el uso de las librerías y el conocimiento de los ciclos de vida de la aplicación, es suficiente para empezar a escribir las primeras aplicaciones.

Entrando en un nivel más técnico, podemos decir que, por la estructura del código, por como estructuramos la aplicación y por el uso de ficheros de la misma, sumado a que haya que añadir un archivo de cabeceras cada vez que se declara una clase, hace que el desarrollo en iOS sea más lento. Por otro lado, a la hora de depurar un programa iOS,

es algo más complicado ya que hay que usar *GDB*. Esto supone tener que aprender otro programa para poder depurar, lo que dificulta aún más el aprendizaje. Este factor, unido al económico, me fuerza a tener que optar por Android como lenguaje de programación para iniciarme en el mundo de la tecnología móvil. En la realización de este trabajo de fin máster he adquirido bastante conocimientos de ambas tecnologías. Entre ellas, la declaración de variables en iOS de una manera óptima, el uso de base datos, de la localización del dispositivo, el ciclo de vida en ambas plataformas, la investigación sobre datos económicos y número de aplicaciones en las respectivas tiendas.

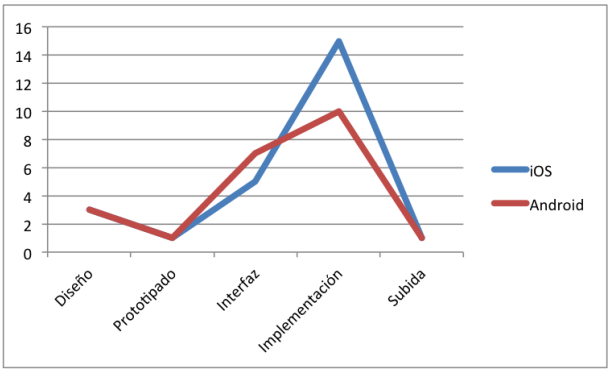


Figura 4.3: Comparativa desarrollo tiempos

Licencia	Ordenador	Monitor	Terminal	Total	
25	399	99	89,99	612,99	Android
99	649	99	389	1236	iOS

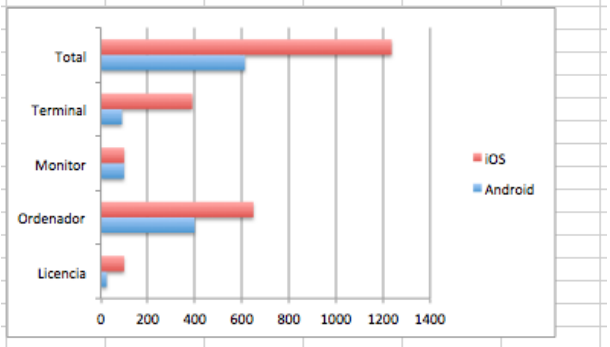


Figura 4.4: Tabla/Gráfica de coste inicial en el desarrollo

4.3. Líneas futuras

El trabajo se puede completar de varias formas ya que se puede profundizar en dos campos. Estos son, por un lado, los de la aplicación de prueba y, por el otro, añadir funcionalidad más compleja para realizar la comparación de esta en ambas plataformas. Estas mejoras no se han realizado por falta de tiempo y porque haría crecer el volumen de este trabajo considerablemente. Estas mejoras son:

Aplicación de prueba

- Almacenar la posición del vehículo una vez se ha abandonado, utilizando el bluetooth. Esto permitiría al usuario tener que olvidarse de almacenar su posición, ya que el dispositivo lo haría por él.
- Utilizar la realidad aumentada para identificar el vehículo. Añadiendo está funcionalidad y usado en software *Vuforia* permite identificar el vehículo con la cámara del dispositivo, lo que hace que localizar el vehículo resulte más sencillo.
- Almacenar la hora del parking para que muestre posibles sitios donde se aparcó, filtrando por distancia y hora.
- Añadir interactuación con usuarios donde puedas compartir sitios libres para que otros usuarios puedan usarlos.

Comparación

- Realizar una comparativa con el uso de animaciones. Ver las diferencias de estas en los dos plataformas.
- Incluir los métodos de compartir y ver cómo se comportan de forma nativa en ambas plataformas.
- Profundizar el proceso de publicación en ambas plataformas añadiendo capturas de pantallas y datos reales.
- Comparativa en la publicidad de ambas plataformas. Sus sistemas nativos como *adMob* y *iAds*.

Completando esta comparativa se podría hacer un libro de como iniciarse en el mundo de la programación móvil. Este libro podría dar una introducción a ambos lenguajes, facilitar datos de rendimiento económico de las aplicaciones y ayudar al programador que lenguaje elegir según los objetivos que busqué, es decir si se plantea obtener beneficios económicos a largo plazo o a corto plazo y también dependiendo de la inversión inicial que desee realizar.

Bibliografía

Estadísticas

[1] http://www.idc.com/getdoc.jsp?containerId=prUS23946013

[2] http://gs.statcounter.com/#mobile_os-ww-monthly-201205-201305-bar

[3] http://www.digitalbuzzblog.com/infographic-2012-mobile-growth-statistics

[4] http://www.visionmobile.com/product/developer-economics-2012

[5] http://www.cultofmac.com/175065/inside-the-app-economy-making-big-money-is-far-fr

[6] http://www.elandroidelibre.com/2011/01/los-markets-de-android-y-apple-ventajas-e-
html

[7] http://blogs.20minutos.es/clipset/ios-contra-android-quien-gana-la-batalla/
datosdeventas

Estudios previos

[8] http://www.passion4teq.com/articles/ios-android-development-comparison-1/

[9] http://reviews.cnet.com/8301-3638_7-57589074/
ios-vs-android-the-game-dev-edition/

[10] http://mobiledevices.about.com/od/kindattentiondevelopers/tp/
Android-Os-Vs-Apple-Ios-Which-Is-Better-For-Developers.htm

Latex

[11] http://www2.dis.ulpgc.es/~lalvarez/teaching/pi/latex/TutorialLatex.pdf

[12] http://jci.codemonkey.cl/charlas/tutorial_latex.pdf

[13] http://elblogdelatex.blogspot.com.es/2006/03/inclur-bibliografa.html

Programación iOS

[14] Aprende iOS: Primeros pasos; Juan M. Cigarrán.

[15] https://developer.apple.com/

[16] http://stackoverflow.com/

Programación Android

[17] El Gran Libro de Android; Jesús Tomás Gironés

[18] http://developer.android.com/index.html

[19] http://stackoverflow.com/

Diagramas

[20] http://es.wikipedia.org/wiki/Diagrama_de_estado

[21] http://www2.uah.es/jcaceres/capsulas/DiagramaCasosDeUso.pdf