ESCUELA UNIVERSITARIA POLITÉCNICA DE TERUEL

DEPARTAMENTO DE INFORMÁTICA E INGENIERÍA DE SISTEMAS

INGENIERIA TÉCNICA EN INFORMÁTICA DE GESTIÓN

TRABAJO FIN DE CARRERA

APARCADROID: APLICACIÓN PARA LA GESTIÓN AUTOMÁTICA Y EFICIENTE DEL ESTACIONAMIENTO EN LAS CIUDADES ESPAÑOLAS

Manuel Antón Simarro
Septiembre 2014

ESCUELA UNIVERSITARIA POLITÉCNICA DE TERUEL

DEPARTAMENTO DE INFORMÁTICA E INGENIERÍA DE SISTEMAS

INGENIERIA TÉCNICA EN INFORMÁTICA DE GESTIÓN

TRABAJO FIN DE CARRERA

APARCADROID: APLICACIÓN PARA LA GESTIÓN AUTOMÁTICA Y EFICIENTE DEL ESTACIONAMIENTO EN LAS CIUDADES ESPAÑOLAS

Autor: Manuel Antón Simarro
Directores: Francisco José Martínez Domínguez
Jesús Gallardo Casero

TRIBUNAL:

Presidenta: Piedad Garrido Picazo

Secretario: Francisco J. Martínez Domínguez

Vocal: Fernando Naranjo Palomino

CALIFICACIÓN:.....

FECHA: 23 de septiembre de 2014

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En este proyecto se presenta una aplicación basada en el uso de teléfonos móviles basados en Android, que permitirá una gestión inteligente de las plazas de aparcamiento libres de una ciudad en tiempo real.

De este modo, los conductores interesados podrán ver dónde hay una plaza libre cercana de forma rápida y sencilla, con el consiguiente ahorro de tiempo y combustible. Además, nuestro sistema permite aplicar conceptos de las conocidas como redes sociales en el ámbito de las redes vehiculares.

Las plazas libres se guardarán en la base de datos almacenando la longitud y la latitud (coordenada de donde se encuentra la plaza) y, la fecha y hora en que fue ocupada o liberada dicha plaza. De esta forma, se puede conocer la posición exacta del estacionamiento y la hora en que esa plaza cambió de estado (siendo liberada u ocupada). Cuando un usuario abra la aplicación deberá tener conexión a Internet (por Wi-Fi o red de datos del dispositivo) y, para tener una mayor precisión del posicionamiento, deberá conectar también el GPS del terminal.

Los experimentos realizados nos demuestran el correcto funcionamiento de la aplicación, y el consiguiente ahorro de tiempo y comodidad del usuario, sobre todo en entornos con muchos vehículos, donde suele ser complicado encontrar plazas de aparcamiento.

Palabras clave: Redes vehiculares, Android, GPS, gestión del aparcamiento, reducción del consumo de combustible, reducción de las emisiones.

---

1. INTRODUCCIÓN

1.1. Motivación

Uno de las peores situaciones que se viven al volante a lo largo del día es el momento de buscar aparcamiento. Para los conductores que no tienen el privilegio de disponer de una plaza de garaje o no pueden permitirse un parking público la búsqueda de un hueco para dejar el coche es más que una pesadilla. Dar vueltas y vueltas con el coche para encontrar un hueco no sólo consigue que hagamos un gasto innecesario de gasolina, sino que puede provocar atascos, el aumento de las emisiones de gases de efecto invernadero y hasta provocar accidentes.

Esta búsqueda de aparcamiento supone una pérdida de tiempo incómoda para los conductores, además de provocar el 30% [1] de los atascos que se producen en las ciudades y, un aumento considerable de las emisiones de gases contaminantes. Además, genera considerables pérdidas de tiempo, de productividad y contribuye a una gestión ineficiente de los servicios de una ciudad.

Basándose en la encuesta realizada a 8.042 conductores de 20 ciudades de cinco continentes, IBM ha elaborado un índice de aparcamiento que mide el gasto económico y emocional que supone buscar aparcamiento [1]. Chicago, seguida de Los Ángeles y Toronto son las ciudades que menos incidencias acusan, mientras que Nueva Delhi, Bangalore y Pekín asumen el mayor porcentaje de quejas de los conductores, en cuanto al tiempo invertido en encontrar estacionamiento y el impacto en la salud y la productividad. Madrid, se encuentra en la posición duodécima entre las veinte ciudades analizadas.

La frustración y los problemas generados a la hora de buscar una plaza de aparcamiento son denominador común de todas las ciudades incluidas en el estudio. Por ejemplo, los conductores en Nairobi invierten un promedio de 31’7 minutos en su búsqueda de aparcamientos, mientras que los ciudadanos de Bangalore, Beijing, Buenos Aires, Madrid, Ciudad de México, París o Shenzhen invierten más de la media mundial (20 minutos) buscando una plaza donde estacionar su vehículo. En concreto, el 16% de los conductores madrileños y el 17% de los residentes en Milán y Pekín invierten entre 31 y 40 minutos buscando un lugar de estacionamiento. El 80% en Shenzhen, el 74% en Pekín, el 76% en Nairobi y el 69% en Madrid reconocieron que no llegaron a su destino porque se dieron por vencidos, fruto del largo tiempo de búsqueda de una plaza. Por el contrario, el 63% de los encuestados en Chicago, el 62% de Estocolmo, 58% de Montreal y 57% de Toronto, rara vez experimentan esta frustración.

Otro de los efectos colaterales generados fruto de la frustración es el alto nivel de agresividad en las discusiones y el aumento del número de las mismas. A nivel mundial, un 27% de los conductores, reconoce haber participado en el último año en una discusión con otro conductor por una plaza de aparcamiento. Nueva Delhi, Bangalore, Nairobi y Milán son las ciudades que más encienden los ánimos de los conductores, mientras que Chicago, Los Ángeles y Estocolmo son las ciudades en las que más calmados se encuentran, y menos les afecta la búsqueda de aparcamiento.

De este modo, un sistema de gestión automática del estacionamiento puede minimizar e incluso llegar a eliminar los problemas anteriormente citados, pudiendo reducir en gran medida las emisiones de gases contaminantes, así como las discusiones, accidentes y otros incidentes que se pueden producir durante la búsqueda de aparcamiento.

1.2. Objetivos

El principal objetivo de este proyecto es desarrollar una aplicación Android para la gestión automática y eficiente del estacionamiento en las ciudades españolas. Esta aplicación permitirá una gestión inteligente de las plazas de aparcamiento libres en una ciudad en tiempo real. De esta forma, los conductores interesados podrán ver dónde existe un aparcamiento cercano, sin tener que dar vueltas, con el consiguiente ahorro de tiempo y combustible.

A su vez, se ha desarrollado una aplicación servidor para poder compartir las localizaciones de plazas libres y que estén accesibles a todos los usuarios del sistema en cualquier momento. Este servidor no necesitará de una aplicación de administración, ya que gestionará las plazas libres de forma automática junto con la colaboración de los usuarios.

De acuerdo con lo anterior, se pueden distinguir de una manera clara los siguientes subobjetivos:

- Desarrollar un servidor central al cual se conectarán las aplicaciones cliente.
- Implementar una base de datos central del sistema a la cual se conectará. únicamente la aplicación servidor.
- Diseñar un protocolo de comunicación.
- Implementar un sistema de registro de usuario.
- Implementar un sistema de autentificación.
- Permitir la visualización de las plazas libres que se encuentran alrededor.
- Permitir ocupar y liberar plazas.
- Permitir consultar información de cada plaza de aparcamiento.
- Permitir abrir el navegador para ser guiado hasta la plaza de estacionamiento deseada.

- Se podrá cerrar sesión.
- Se podrá consultar y editar el perfil de usuario.
- Se podrá consultar el historial de aparcamientos recientes del usuario.

1.3. Visión general del documento

El presente documento consta de los apartados indicados a continuación:

- ***Conceptos previos***: apartado en el que se contextualiza la aplicación y se explican conceptos clave para entender su funcionamiento y filosofía. Se realizará una introducción al sistema operativo Android, así como una explicación de su funcionamiento, arquitectura y evolución en cada versión.
- ***Estado del arte:*** se exponen diferentes aplicaciones presentes en el mercado que ofrecen el mismo servicio o servicios complementarios al trabajo fin de carrera que estamos tratando.
- ***Análisis (Documento de Especificación de requisitos):*** en este apartado se desarrollan todos los requisitos de la aplicación lo que formaría un documento que sigue las directrices que se marcan en el estándar del IEEE *Recommended Practice for Software Requirements Specifications* IEEE 830-1998.
- ***Planificación***: aquí se encuentra la planificación inicial prevista y la planificación real.
- ***Cálculo de costes***: en este apartado se realiza estimación de costes que se ocasionarán durante el desarrollo de la aplicación. Dicha estimación se realizará mediante el método de Puntos Función.
- ***Diseño***: contiene los distintos diagramas UML creados en la fase de diseño y que serán utilizados para la fase de implementación
- ***Implementación:*** descripción de las herramientas y tecnologías utilizadas durante el proceso de implementación del proyecto. Además se comentan otros aspectos de interés relacionados con dicho proceso.
- ***Pruebas:*** en este apartado se detallan las diferentes pruebas realizadas con el sistema para comprobar su correcto funcionamiento.
- ***Conclusiones:*** apartado para la reflexión y el análisis sobre la realización del proyecto y los resultados obtenidos.
- ***Ampliaciones futuras:*** en esta sección se describen las posibles directrices a seguir para la mejora y ampliación de las características de la aplicación.
- ***Bibliografía:*** se incluyen las referencias bibliográficas utilizadas a lo largo de la redacción del presente proyecto.
- ***Anexos:*** en esta sección se incluye todo aquello que no tiene cabida en otros apartados, como el manual de usuario.

1.4. Agradecimientos

En primer lugar agradecer a toda la comunidad de desarrolladores tanto de Android como de Java por la increíble cantidad de tutoriales y foros de ayuda que existen en Internet, ya que, me han ayudado en distintas fases del desarrollo de la aplicación.

También me gustaría agradecer su ayuda a Francisco J. Martínez y Jesús Gallardo, mis directores del proyecto, y a Piedad Garrido por toda la paciencia que han tenido conmigo y la ayuda recibida durante el desarrollo del proyecto y, como no, también agradecer a mis compañeros Jesús Fuentes, Vicente Torres, Javier Barrachina, Julio Sangüesa y Manuel Fogué por el magnífico ambiente de trabajo creado en el laboratorio y los excelentes consejos recibidos.

Por último, pero no menos importante, agradecer a mi familia todo el apoyo recibido a lo largo de la carrera, y también a mi novia Laura, por animarme en los momentos de "bajón" y conseguir que siguiera trabajando duro durante el desarrollo de este Trabajo Final de Carrera.

2. CONCEPTOS PREVIOS

En este apartado se detallan algunos conceptos importantes para contextualizar el proyecto. En primer lugar se define que es Android, ya que, todo el proyecto se dirige hacia los usuarios de dispositivos móviles con este Sistema Operativo. En segundo lugar se explica el sistema de posicionamiento A-GPS disponible en la mayoría de terminales móviles de la actualidad. Es conveniente conocer las características de este sistema de posicionamiento ya que será el elemento hardware encargado de triangular nuestra posición, y así poder mostrar nuestra localización en el mapa. A continuación se expondrá "Google Maps" que es el servicio de mapas elegido para nuestra aplicación. Por último se definirá "PostGIS" que es un complemento muy importante de la base de datos utilizada en el proyecto.

2.1.Introducción a Android

Android [2] es un sistema operativo basado en Linux, abierto, libre, gratuito y multiplataforma. Inicialmente fue desarrollado por Android Inc., que posteriormente, en 2005, fue comprada por Google; en la actualidad, este sistema operativo lo desarrollan los miembros de la Open Handset Alliance (consorcio de compañías hardware, software y telecomunicaciones comprometidas con la promoción de estándares abiertos para dispositivos móviles) liderada por Google.

Aunque fue diseñado principalmente para dispositivos móviles (teléfonos inteligentes, tablets, etc.), también está siendo implantado en multitud de dispositivos electrónicos y vehículos. Fue presentado en 2007 junto a la creación de la Open Handset Alliance. A pesar de ello, el primer terminal comercial no se vendió hasta Octubre de 2008 y fue el HTC Dream.

Android además es multitarea y permite a los desarrolladores acceder las funcionalidades principales del dispositivo mediante aplicaciones, que pueden ser reemplazadas, por otras oficiales o de terceros desarrolladas a través de las herramientas proporcionadas por Google. El sistema consta de 12 millones de líneas de código escritas en XML, C, C++ y Java. La mayoría de dicho código se libera bajo la licencia Apache, lo que permite a la comunidad de desarrolladores realizar cambios y mejoras en el sistema operativo.

2.1.1. Arquitectura

En primer lugar, destacar que Android es un Sistema Operativo que posee un núcleo monolítico, es decir, su núcleo es grande y complejo, ya que, engloba todos los servicios del sistema operativo. Esto implica que tenga un mayor rendimiento pero, por el contrario, cualquier cambio a realizar en cualquier servicio requiere la recompilación de dicho núcleo y el reinicio del sistema para poder aplicar los cambios.

En el siguiente diagrama se muestran, agrupados por capas, los principales componentes que forman el Sistema Operativo Android. Al ser un sistema monolítico, cada una de las capas mostradas en el diagrama necesita elementos de la capa inferior para realizar sus funciones.

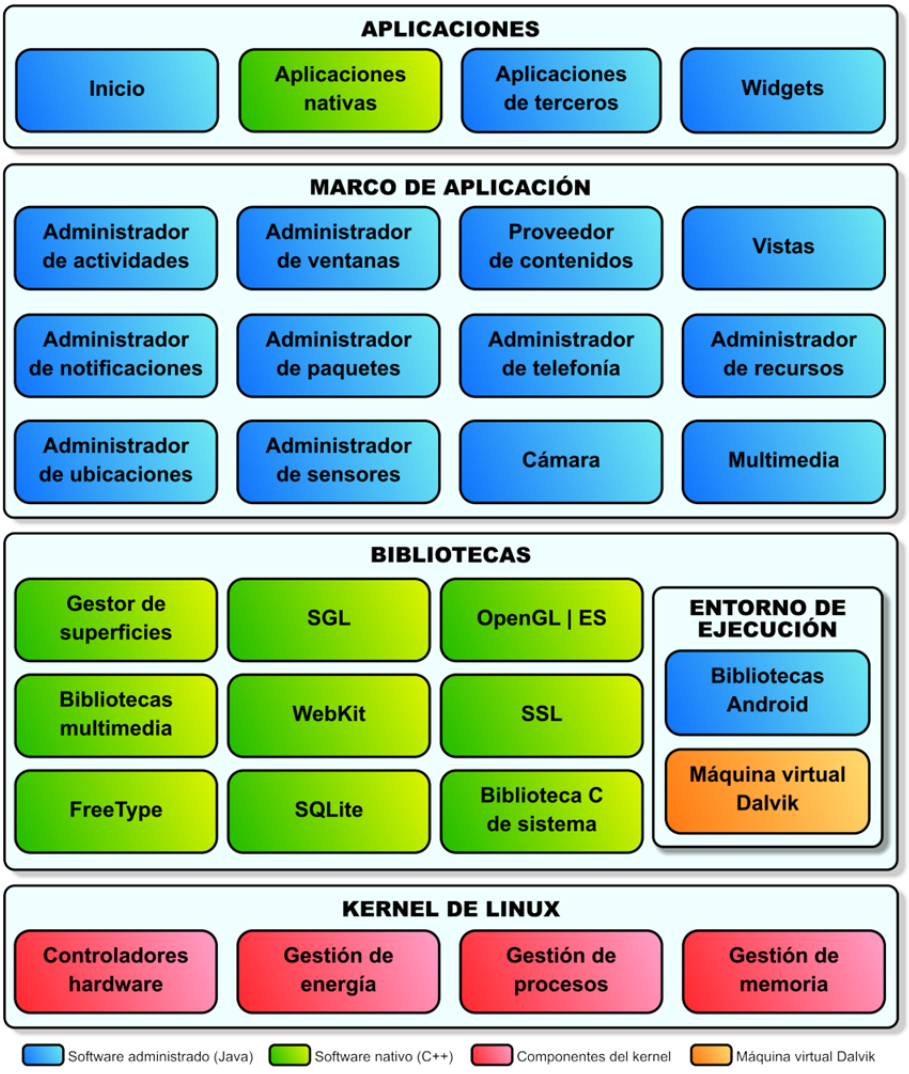


Figura 1: *Arquitectura Android. Fuente: lacolumna.wordpress.com Agosto 2014.*

Kernel

Android utiliza un *Kernel* o núcleo Linux que cambia dependiendo de la versión de Android, y es similar al que se incluye en cualquier distribución de Linux, solo que se adapta al hardware en el que se ejecutará.

Dicho núcleo crea una capa de abstracción para los elementos hardware a los que tienen que acceder las aplicaciones que permite acceder a dichos componentes sin necesidad de conocer sus características hardware concretas. Además, este núcleo se utiliza para controlar el funcionamiento de los servicios base del sistema como son: seguridad, gestión de memoria, gestión de procesos, *stack* de red y modelo de drivers.

Librerías

Es una capa situada sobre el núcleo y contiene las librerías nativas de Android. Dichas librerías nativas están escritas en C o C++ y son compiladas para el hardware específico del dispositivo. Su misión es proporcionar funcionalidad a las aplicaciones, de forma que se puedan realizar de forma más eficiente.

Entorno de ejecución

Esta capa se encuentra al mismo nivel que la anterior debido a que también incluye librerías. El principal componente de esta capa es la máquina virtual Dalvik, que es la encargada de ejecutar todas las instrucciones no nativas de Android, realizando una compilación en tiempo de ejecución.

Desde la versión 4.4 Kitkat se introduce la nueva máquina virtual ART (el usuario puede seleccionar cuál de las dos desea que ejecute sus aplicaciones) que mejora el rendimiento notoriamente y que en futuras versiones sustituirá a la antigua Dalvik

Marco de aplicación

Esta capa está formada por todos los servicios que utilizan las aplicaciones para realizar su misión. La mayoría de estos servicios son librerías Java que acceden a los recursos de capas inferiores a través de la máquina virtual.

Aplicaciones

Es la última capa, en ella se encuentran instaladas todas las aplicaciones del dispositivo. Las nativas (programadas en C o C++) y las instaladas (normalmente programadas en Java).

2.1.2. Versiones

La primera versión (1.0) de Android, denominada Apple Pie, fue liberada el 23 de Septiembre de 2008, saliendo su actualización a la versión 1,1 el 9 de Febrero de 2009. A pesar de ello, no fue hasta la versión 1.5 cuando el sistema no adquirió cierta relevancia.

Android 1.5 Cupcake

Liberada el 30 de Abril de 2009 y basada en el núcleo 2.6.27 de Linux, incluía nuevas mejoras en la interfaz, incorporando widgets y transiciones animadas entre pantallas. Además se añadió la posibilidad de grabar y reproducir videos, dando la opción de subirlos a YouTube y las imágenes a Picasa.

Esta versión, también daba soporte para Bluetooth A2DP, posibilitando la conexión automática de un auricular bluetooth a distancia.

Android 1.6 Donut

Siendo liberada el 15 de Septiembre de 2009, incorporaba el núcleo Linux 2.6.29. Sus principales novedades fueron que incluía un Market mejorado, con el cual era más fácil encontrar aplicaciones, y que integraba en una misma interfaz la galería de foto y video. También mejoraba la búsqueda de voz, haciéndola más rápida y flexible, y la experiencia Web que incorporaba marcadores e historiales. A su vez se proporcionó soporte a CDMA/EVDO, 802.1x, VPN y a dispositivos con pantallas WVGA, más grandes, que comenzaban a aparecer en el mercado.

Android 2.0/2.1 Eclair

La versión 2.0 fue liberada el 26 de Octubre de 2009, mientras que el 3 de Diciembre del mismo año se liberó la 2.0.1 y el 12 de Enero de 2010 la 2.1. A pesar de estar basadas en el mismo núcleo que la versión 1.6, supusieron una gran renovación en la interfaz de usuario, dándole mucha más potencia y soporte para distintos tamaños de pantalla. Además se volvió a mejorar el navegador, cambiando la interfaz y dándole soporte para HTML5. También se incluyó una nueva versión de GoogleMaps (v3.1.2), fondos de pantalla

animados, se dio soporte para el zoom digital y el flash en la cámara, para bluetooth 2.1 y para pantallas multitáctil. A pesar de todas las mejoras, también se logró optimizar la velocidad del hardware.

Android 2.2 Froyo

Esta versión se liberó el 20 de Mayo de 201, y está basada en el núcleo 2.6.32 de Linux. Su principal mejora fue la optimización del sistema, buscando incrementar su rendimiento, reduciendo la fragmentación de la memoria e intentando mejorar la ejecución de aplicaciones. Las mejoras más significativas de esta versión fueron la actualización del Market incluyendo actualizaciones automáticas y la inclusión de soporte para Adobe Flash 10.1 y para pantallas de alta definición. También se incluyen Wi-Fi hotspot, tethering por USB y la compilación JIT(compilación en tiempo de ejecución), además de cambiar el motor de la aplicación Browser por el de Google Chrome e incluir la sincronización remota del calendario.

Android 2.3 Gingerbread

Siendo liberada el 6 Diciembre de 2010, está basada en el núcleo 2.6.35.7 de Linux. En ella se vuelve a producir una actualización del diseño de la interfaz ofreciendo, de este modo, soporte para pantallas extra grandes. Otras mejoras fueron el soporte para tecnologías VoIP, decodificación de audio AAC, compatibilidad con otros tipos de sensores y múltiples cámaras; además de ofrecer nuevos efectos de audio, mejores gráficos para diseñadores de juegos y un control de energía mejorado.

Android 3.0/3.1/3.2 Honeycomb

Fue una versión concebida pensando, exclusivamente, en el ecosistema de los tablets. Su API presentaba muchas diferencias con respecto las anteriores, por lo que las aplicaciones creadas con ésta podían ser incompatibles con dispositivos que tenían versiones de Android 2.X.

Esta versión fue liberada el 5 de Enero de 2011, estando basada en el núcleo 2.6.36 de Linux. Susu principales mejoras fueron en el interfaz, que tenía un nuevo diseño, escritorios 3D y widgets rediseñados. También incorporó mejoras en el sistema multitarea y navegador web, además de añadir soporte para videochat mediante Google Talk y variedad de accesorios USB.

Android 4.0 Ice Cream Sandwich

Tras Honeycomb se consideró que tener un sistema operativo para tablets diferente y, en muchos casos, que generaba incompatibilidades con las

aplicaciones, era un error. Con la aparición de Android 4.0 se volvió a unificar la versión para móviles y tablets y, por tanto, se consiguió que las aplicaciones funcionaran indistintamente ya fuera en un teléfono inteligente o en un tablet. Esto fue posible gracias a la creación de un Framework único para las aplicaciones.

Fue liberada el 20 de Octubre de 2011, estando basada en el núcleo Linux 3.0.8. Incorporaba una interfaz más limpia, con nuevos botones y, estrenaba su nuevo tipo de fuente llamado "Roboto". Además mejoraba los widgets, corrector de texto, el sistema multitarea, el gestor de tráfico de datos de Internet, introducía un nuevo software para la cámara con nuevas utilidades, y la posibilidad de realizar aceleración por hardware, pudiendo ser la interfaz dibujada por la GPU.

Aparte de las anteriores, incorporó nuevas funcionalidades como el reconocimiento de voz del usuario, el reconocimiento facial para el desbloqueo de la pantalla, soporte para contenedores MKV, un sistema de ficheros más fácil de manipular y, más herramientas para la gestión de aplicaciones, pudiéndolas cerrar y, de este modo, liberar memoria.

Android 4.1/4.2/4.3 Jelly Bean

La primera versión de Jelly Bean fue liberada el 9 de Julio de 20012. Las mejoras de esta versión se centran en ofrecer una mejor experiencia de usuario.

Dichas mejoras son en la fluidez, rapidez y suavidad del sistema, redistribución de los elementos en el escritorio, dictado por voz disponible offline, nuevos idiomas no occidentales en el teclado, mejoras en la accesibilidad y en la cámara. También se mejora el Android Bean, pudiendo enviar fotografías y vídeos con tan solo tocar los dispositivos, se mejoran las notificaciones, búsquedas y Google Now. La última mejora importante es que se introduce la actualización de aplicaciones inteligentes, de modo que, cuando exista una actualización, no será necesario descargar toda la aplicación, sino solo la parte especifica que sea necesaria

Android 4.4 Kitkat

Se trata de la última versión de Android que ha salido al mercado. Esta versión fue liberada el 31 de octubre de 2013 y supuso un gran paso adelante en cuanto a la optimización de recursos, ya que, permite su ejecución en terminales con tan sólo 512MB de memoria RAM. Esto se consigue mediante la simplificación de sus procesos, para reducir el uso de memoria, por lo cual, la

plataforma se vuelve más eficiente y rápida. Además incluye nuevas APIs y herramientas que ayudan a los desarrolladores a crear aplicaciones.

También, además de la amortización de recursos, se han introducido importantes mejoras, de seguridad en las transacciones NFC, el nuevo administrador de impresión nativo. Por otro lado, se han añadido nuevas funcionalidades, como son la integración con el almacenamiento en la nube (pudiendo acceder a las carpetas y archivos como si estuvieran en el dispositivo), Google Now siempre escuchando en la pantalla de inicio, nueva agenda de Google que prioriza los contactos de las personas con las que más hablamos y un identificador de llamadas inteligente, que buscará coincidencias con los teléfonos de empresas de una lista de Google Maps, si no lo tenemos almacenado en la agenda. Otras novedades importantes son que unifica todos los servicios de mensajería en la aplicación Hangouts, introducción de un nuevo teclado que permite añadir *emojis* en cualquier aplicación, soporte para conseguir un ahorro de energía en la utilización de los sensores y, también, soporte para el detector y contador de pasos, modo de pantalla completa .

Por último, introduce mejoras en la accesibilidad, seguridad y conectividad, ya que, añade subtítulos en los videos y ha actualizado la configura de SELinux de "permisiva" a "hacer cumplir", añadido nuevos algoritmos de cifrado. En cuanto a la conectividad, se añade soporte a nuevos perfiles de Bloetooth (HID over GATT, MAP, AVRCP 1.3), a emisores de IR(infrarrojos) y, también, Wif-Fi TDLS, facilitando la transmisión de archivos multimedia.

2.1.3. Fragmentación

Uno de los primeros aspectos a tener en cuenta a la hora de desarrollar aplicaciones Android es el nivel de API que se va a utilizar. Esto es debido a que cada versión de Android tiene una API distinta, en la cual se ha desarrollado más librerías, se han actualizado otras o, incluso, se han descartado en beneficio de otras. Esta diferencia de APIs ocasiona el problema de la fragmentación en Android y por ello, cuando se desarrolla una aplicación, no tiene por qué funcionar en todas las versiones. Por esto, se debe prestar atención y elegir concienzudamente con que API se desea desarrollar una aplicación.

Los datos que se muestran en las tablas y diagramas del presente apartado, han sido obtenidos de la página web oficial Android Developers en Agosto de 2014, dentro del apartado Dashboards [3].

En la siguiente tabla, se muestra la correspondencia entre la versión de Android y el nivel de API:

Version	Nombre	Nivel de API
1.5	Cupcake	3
1.6	Donut	4
2.1	Eclair	7
2.2	Froyo	8
2.3.3-2.3.7	Gingerbread	10
3.x.x	HoneyComb	11-13
4.0	Ice Cream Sandwich	14
4.0.3 – 4.0.4		15
4.1.x	Jelly Bean	16
4.2.x		17
4.3		18
4.4	KitKat	19

Tabla 1: *Versiones de Android*

Como se ha mencionado anteriormente, desde su creación y gracias a su evolución, existen diversas versiones de Android, de modo que, existen terminales con todas las versiones disponibles. El porcentaje de dispositivos que usan una u otra versión varia constantemente quedando la distribución, a día de hoy, del siguiente modo:

Version	Nombre	Distribución
2.2	Froyo	0.7%
2.3.3-2.3.7	Gingerbread	13.5%
4.0.3 – 4.0.4	Ice Cream Sandwich	11.4%
4.1.x	Jelly Bean	27.8%
4.2.x		19.7%
4.3		9.0%
4.4	KitKat	17.9%

Tabla 2: *Distribución según versión de Android*

Los datos de la tabla anterior, se pueden trasladar a un gráfico de sectores y, de este modo, ver más claramente las proporciones de uso de cada versión.

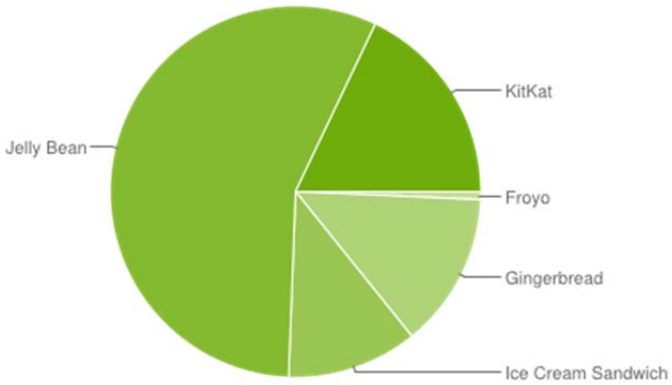


Figura 2: *Distribución según versión de Android*

Las versiones anteriores a Android 2.2 Froyo no están reflejadas en los anteriores datos (proporcionados por Google), ya que, dichos datos se obtienen a través de la aplicación de Play Store y ésta no soporta versiones anteriores. De todos modos, en Agosto de 2013, las versiones anteriores a Android 2.2 suponían alrededor de un 1% del total de dispositivos.

En cuanto a tamaños de pantalla y densidad de píxeles de esta, también existe una gran diversidad dentro del ecosistema Android. Por ello, y para

simplificar la manera en que se diseña los interfaces de usuario para las diferentes pantallas, Android divide los rangos de tamaños y densidades. La siguiente tabla refleja cómo se distribuye el mercado Android en cuanto a tamaños y densidades de pantalla se refiere:

	ldpi (120dpi)	mdpi (160dpi)	tvdpi (213dpi)	hdpi (240dpi)	xhdpi (320dpi)	xxhdpi (480hdpi)	Total
Pequeña	6.8%						**6.8%**
Normal		11.4%		34.5%	19.4%	15.3%	**80.6%**
Grande	0.6%	4.5%	1.7%	0.6%	0.6%		**8.0%**
Extra Grande		3.9%		0.3%	0.4%		**4.6%**
Total	**7.4%**	**19.8%**	**1.7%**	**35.4%**	**20.4%**	**15.3%**	

Tabla 3: *Distribución según tamaño de pantalla y densidad de píxeles*

Del mismo modo que hemos hecho en el caso de las versiones, en el siguiente gráfico de sectores podemos ver las proporciones existentes en cuanto a los tamaños y densidades de pantalla.

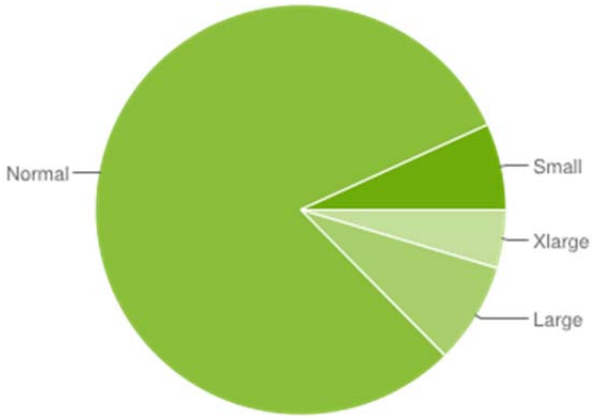


Figura 3: *Distribución según tamaño de pantalla*

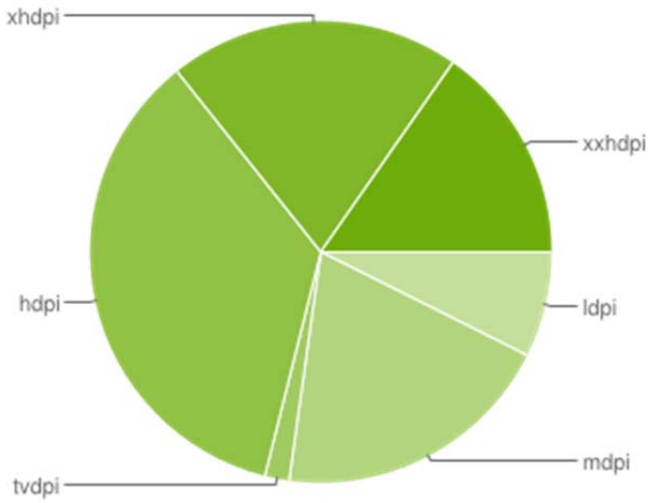


Figura 4: *Distribución según densidad de píxeles*

2.1.4. Distribución del Mercado

Dentro del mercado de la telefonía móvil, existen diversos sistemas operativos, pero en los últimos años, desde la aparición de los teléfonos inteligentes se ha librado una dura batalla comercial por conseguir dominar la cuota de mercado. Esto se puede ver reflejado en la siguiente figura, que muestra la evolución de las ventas a nivel mundial, de las principales plataformas móviles.

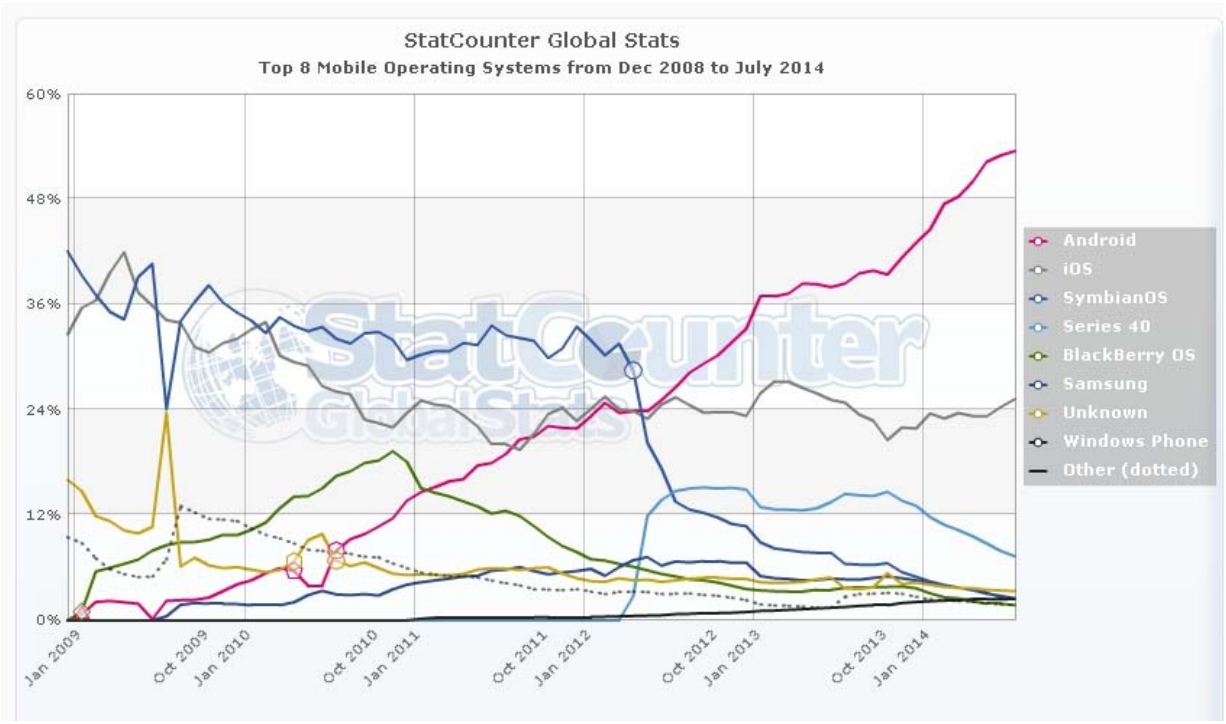


Figura 5: *Distribución de los Sistemas Operativos móviles a nivel mundial. Fuente: statcounter.com Agosto2014.*

En la figura anterior, se observa cómo Android, desde su aparición, ha experimentado un crecimiento casi continuo hasta alcanzar el liderazgo absoluto en cuanto a cuota de mercado, llegando a alcanzar el 53,4% de la cuota total de mercado.

En cambio, las plataformas dominadoras hasta ese momento han sufrido una suerte desigual. Symbian que gracias a ir de la mano de Nokia había dominado el mercado durante varios años, ha sufrido una caída estrepitosa, disminuyendo su cuota desde el 42% de diciembre de 2009 hasta un 2% en Julio de 2014. Por el contrario, iOS, que hasta el momento solo se encontraba con la competencia de BlackBerry, fue perdiendo cuota de mercado hasta enero de 2011, momento en que se estabilizó y se mantiene constante alrededor del 25% de cuota de mercado.

En el caso de España, la situación es algo diferente, como se puede observar en la siguiente figura:

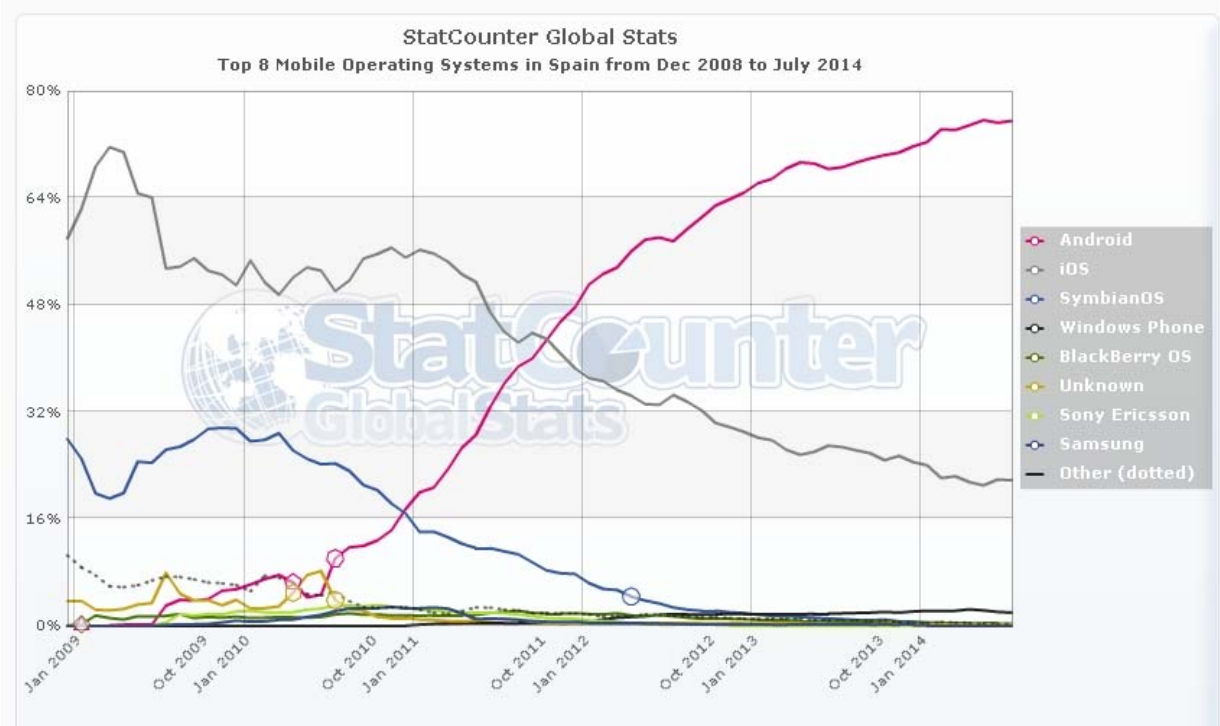


Figura 6: *Distribución de los Sistemas Operativos móviles en España. Fuente: statcounter.com Agosto2014.*

Se observa la gran pérdida de cuota de mercado del sistema operativo Symbian, y como se ha reducido de una manera más drástica que a nivel mundial, cayendo hasta el 2% de cuota en Octubre de 2012, un año y 9 meses antes que se alcanza la misma cuota a nivel mundial.

En nuestro país hasta finales de 201, el dominador del mercado era iOS, alcanzando su cuota máxima en Marzo de 2009 con un 71,6%. A partir de éste momento, su cuota de mercado se fue reduciendo y, en Octubre de 2011, se marcó un punto de inflexión, ya que, Android que experimentaba una fuerte tendencia creciente lo igualó en cuota de mercado. A partir de este momento, comenzó el dominio incuestionable de Android, alcanzando su máximo histórico en Mayo de 2014 con un 75,6% de la cuota y continuando su tendencia dominadora.

2.2. A-GPS

Se trata de un sistema de posicionamiento global, el acrónimo A-GPS [3] proviene del inglés Assisted Global Possitioning System, es decir, GPS asistido que se suele usar en teléfonos inteligentes, tablets y demás dispositivos móviles.

Un sistema GPS (Global Positioning System) está formado por una red de 30 satélites denominada NAVSTAR, los cuales están situados en una órbita a unos 20.000km. de la tierra. Esta red es propiedad del Gobierno de los Estados Unidos y está gestionado por su Departamento de Defensa. Éste GPS convencional presenta dificultades a la hora de proporcionar posiciones precisas cuando disponemos de una señal débil, por ejemplo, cuando se atenúa dicha señal por encontrarse con obstáculos como estar dentro de edificios o debajo de árboles. Además, la primera vez que los receptores GPS se encienden en tales condiciones, algunos sistemas no asistidos, no son capaces de descargar información de los satélites GPS y, por lo tanto, los hace incapaces de triangular la posición hasta recibir una señal clara durante al menos un minuto.

Los receptores A-GPS pueden solucionar estos problemas de dos formas, mediante el acceso a un Servidor de Asistencia en línea (modo on-line) o fuera de línea (modo off-line). Los modos en línea acceden a los datos en tiempo real, por lo que necesitan tener una conexión de datos activa mientras que los sistemas fuera de línea permiten utilizar datos descargados previamente (a través de GPRS, Ethernet, WIFI, ActiveSync o similar).

En cualquier caso, el A-GPS utiliza los datos obtenidos de un servidor externo y los combina con la información de celda o antena de telefonía móvil para conocer la posición aproximada y saber qué satélites tiene sobre él.

Todos los datos referentes a los satélites se obtendrán del servidor externo (modo on-line) o del fichero previamente descargado (modo off-line), y dependiendo de la posición proporcionada por la red de telefonía, el GPS dispondrá de los datos de unos satélites u otros y junto con los datos obtenidos a través del receptor convencional de GPS hará que se triangule la posición de una manera notablemente más rápida y precisa.

2.3. Google Maps

Se trata de un servidor de aplicaciones de mapas en la web. Este servicio ofrece imágenes de mapas desplazables, así como fotografías por satélite del mundo e incluso la ruta entre diferentes ubicaciones o imágenes a pie de calle (Google Street View).

Google Maps [5] está disponible como aplicación (Google Earth para equipos de escritorio y Google Maps para dispositivos móviles) y como sitio web. Ambas alternativas permiten a los usuarios desplazarse a través de los mapas, hacer zoom en el sitio que desee, visualizar imágenes de satélite de gran calidad, mapeos vectoriales, mapas con calles y rutas. También permite realizar búsquedas de calles, servicios, transportes, etc.

Desde el aspecto del marketing online, Google Maps permite anuncios sobre el mapa, mostrando la ubicación del negocio, así como cierta información importante acerca de él. Además, el ecosistema de Google Maps, cuenta con distintas API que permite la fácil integración de sus mapas en cualquier sitio web, también en aplicaciones Android e iOS, así como, la implementación de sus distintas funcionalidades.

Fue anunciado por primera vez en el Google Blog el 8 de febrero de 2005 y sólo disponía de soporte para Internet Explorer y Mozilla Firefox. Posteriormente, el 25 de ese mismo mes se añadió el soporte para Opera ySafari. Estuvo en fase beta durante seis meses antes de convertirse en parte de Google Local el 6 de octubre de 2005.

2.4.PostGIS

Es un módulo que añade soporte de objetos geográficos al Sistema Gestor de Bases de Datos PostgreSQL [6], permitiéndole gestionar datos espaciales para su utilización en Sistemas de información Geográfica. Se publica bajo la licencia Publica General de GNU [7].

A día de hoy, dispone de una interfaz de usuario con herramientas para la gestión de datos, soporta funciones básicas de topología, transformación de coordenadas, validación de datos, programación de APIs (Application Programming Interface), almacenamiento de información raster, etc. Además, incorpora herramientas para la realización de cálculos de rutas, gestión de redes, superficies 3D y objetos complejos como curvas y splines.

PostGIS es un producto que ha demostrado su eficiencia y ser muy superior que la extensión geográfica de MySQL, y también, estar a la altura de la base versión geográfica de la base de datos de Oracle.

3. ESTADO DEL ARTE

3.1.Introducción

Actualmente con el auge de las tecnologías y dispositivos móviles disponemos de una gran cantidad de aplicaciones que nos ayudan a hacer nuestras vidas más fáciles. La búsqueda de aparcamiento es un campo en el que nos pueden facilitar mucho la tarea estas tecnologías, y a pesar de ello en España no está suficientemente explotado este mercado. En el siguiente apartado realizaremos un breve recorrido por algunas aplicaciones de este tipo existentes a nivel internacional y otras a nivel estatal.

3.2.Aplicaciones similares disponibles en el mercado.

3.2.1. OpenSpot

Es una aplicación Android de Google cuya finalidad es facilitarnos la búsqueda de aparcamiento. Actualmente solo se encuentra disponible en Estados Unidos, Canadá y en los Países Bajos para terminales con Android 2.0 o superior[8].

Esta aplicación funciona como una red social de modo que, un conductor, a la hora de retirar su vehículo se conecta a *OpenSpot* y comparte la ubicación de la plaza, de tal forma que otros usuarios que buscan aparcamiento pueden verla en sus terminales. Se notificarán las plazas que se encuentren a distancia igual o menor a 1,5Km.

La aplicación contiene un mapa en el cual, se muestran las plazas libres mediante un sistema de colores según el espacio de tiempo transcurrido desde que éstas quedaron libres. De tal modo que el rojo marca los nuevos espacios libres; el naranja, los estacionamientos que no han sido ocupados durante más de cinco minutos, y en color amarillo, los espacios libres durante más de diez minutos.

Superados estos tres niveles, después de veinte minutos, estas plazas se desclasifican, ya que, es considerado demasiado tiempo como para que aún sigan disponibles.

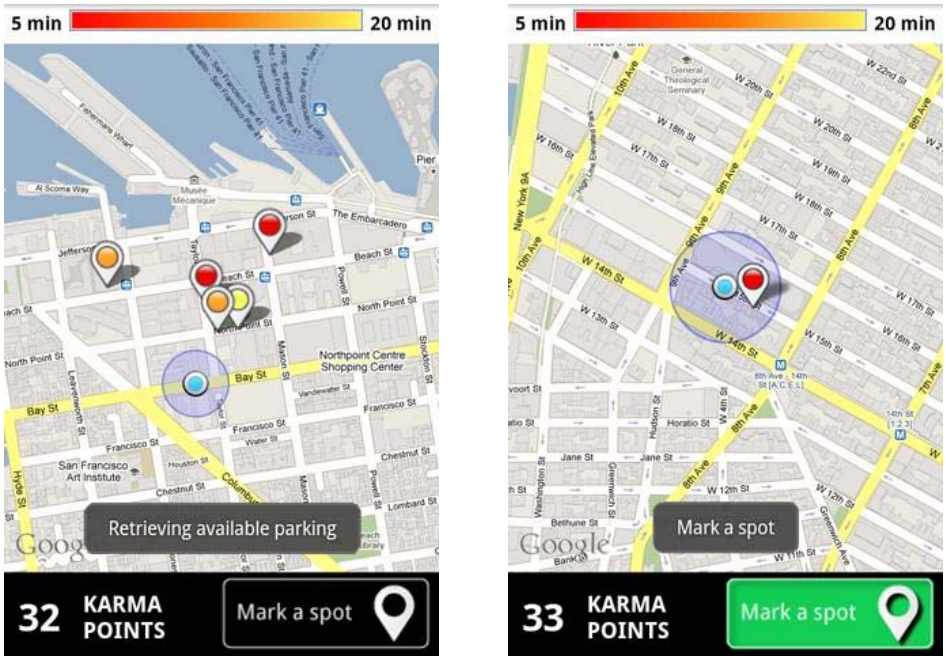


Figura 7: *Capturas de la aplicación Open Spot*

3.2.2. SFPark

Se trata de un sistema implementado en la ciudad de San Francisco para ayudar a encontrar aparcamiento rápidamente mediante la utilización de la aplicación para dispositivos Android o iOS. Este sistema dispone de 7.000 plazas normales y 12.250 plazas de garaje en ocho zonas distintas de la ciudad; además regula el precio de los estacionamientos en función de la demanda de aparcamiento. [9]

Para conseguir que funcione SFpark, se utilizan unos sensores que avisan cuándo la plaza queda libre, mostrando la información en tiempo real tanto en la aplicación para dispositivos móviles como en la web del servicio.

No está diseñada para utilizarse mientras se conduce, la aplicación muestra una advertencia cuando se ejecuta y vuelve a mostrarlo si detecta que se superan los 16 kilómetros por hora.

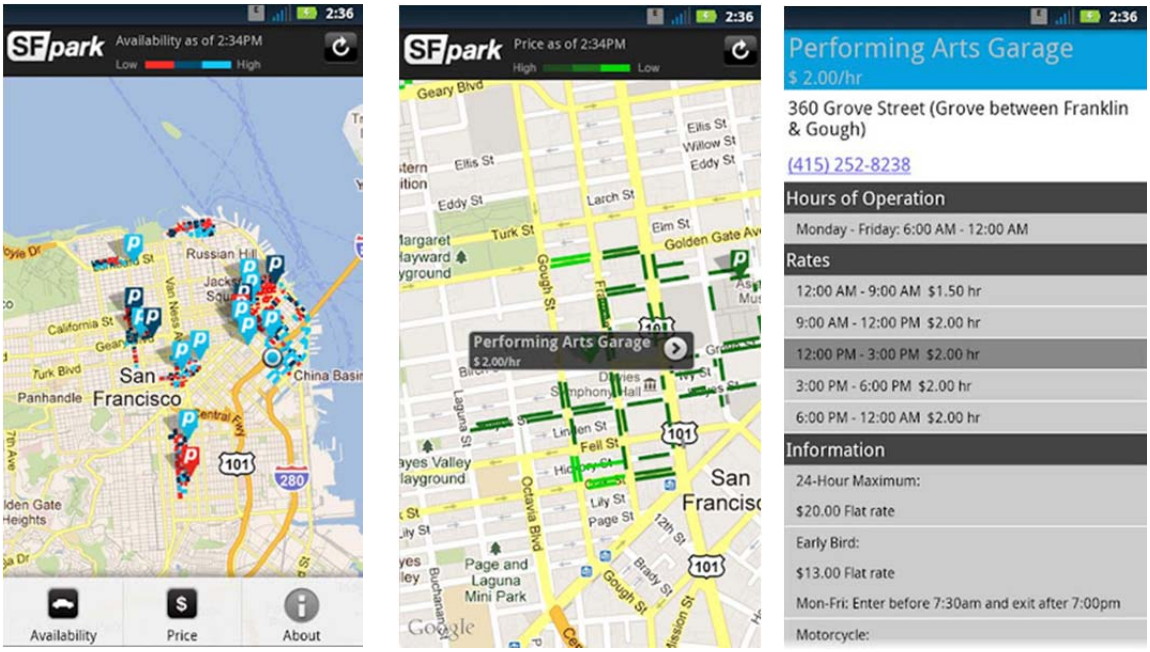


Figura 8: *Capturas de la aplicación SFPark*

3.2.3. Aparcamientos España

Es una aplicación completamente dedicada a la búsqueda de aparcamiento. Utiliza Google Maps y el GPS del dispositivo para realizar las siguientes tareas: [10]

1. Buscar todos los aparcamientos en las proximidades.
2. Navegar por el mapa para encontrar otro aparcamiento.
3. Iniciar automáticamente el navegador para llegar al aparcamiento.
4. Uso del buscador para encontrar aparcamientos en cualquier lugar.

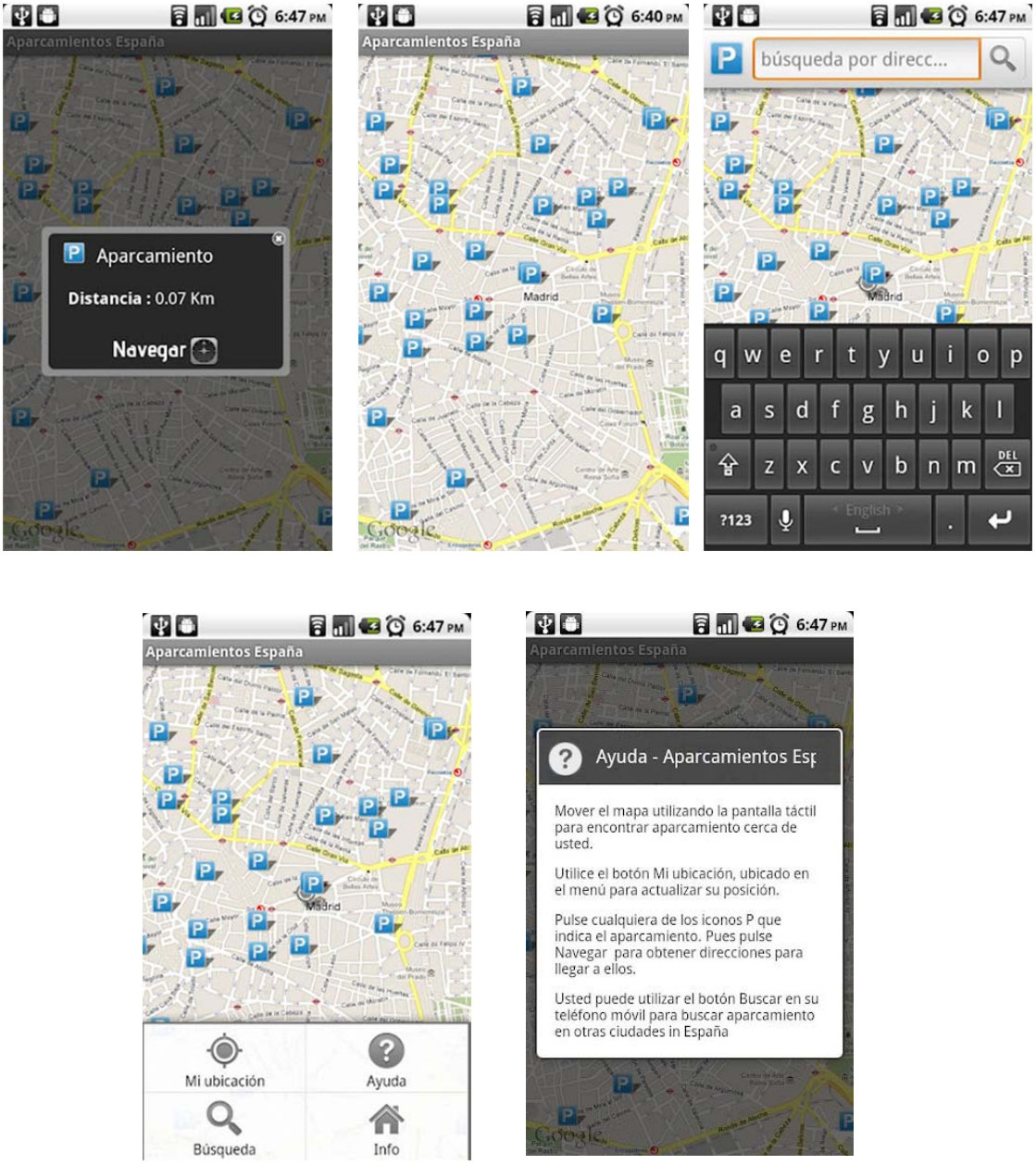


Figura 9: *Capturas de la aplicación Aparcamientos España*

3.2.4. AA Parking

Aplicación impulsada por Parkopedia para la búsqueda de aparcamiento en el Reino Unido e Irlanda [11]. Ofrece cobertura tanto a estacionamientos de pago como estacionamientos libres. Ofrece las siguientes características:

1. Búsqueda el estacionamiento más barato o gratuito usando la ubicación actual o introduciendo una dirección.
2. Indicaciones para llegar a la entrada del aparcamiento.
3. Muestra la disponibilidad de espacios libres en tiempo real (únicamente donde esté disponible).
4. Uso de filtros: Street Parking, Parking gratuito, Aparcamiento cubierto, Aparcamiento vigilado, etc.
5. Búsqueda de horarios de apertura y precios.
6. Buscar Park and Ride (lugares de estacionamiento que permiten hacer fácilmente trasbordo entre transporte privado y público).

Se trata de una aplicación de pago, que está disponible en Android Play Store por 2.35€.

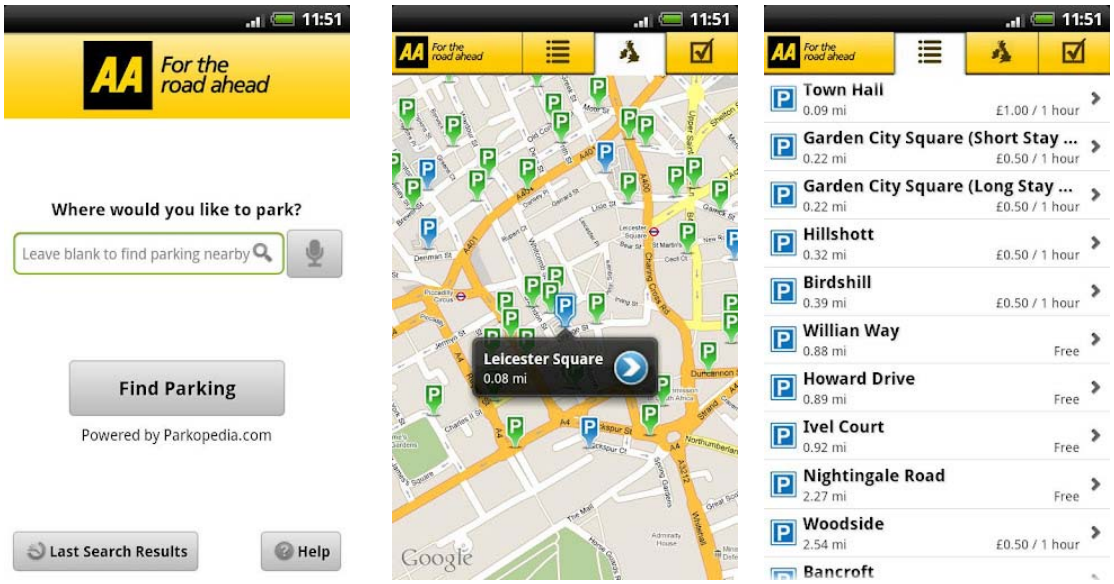


Figura 10: *Capturas de la aplicación AA Parking*

3.2.5. Parking Finder

Se trata de un sistema GPS que es capaz de buscar plazas de aparcamiento, y también, de localizar dónde hemos estacionado el coche [12].

Este dispositivo combina el GPS con el 3G bajo una plataforma Android. También puede ser utilizado como sistema antirrobo, asistente en la conducción, o como sistema de pago en las zonas de aparcamiento regulado.

Las características técnicas de este dispositivo son:

1. GPS.
2. Conexión de datos 3G.
3. Pantalla multitáctil para navegar por los menús. Además, incluye soporte para gestos, lo que facilita la navegación por los menús y acciones como realizar zoom.
4. Lector de huellas dactilares para identificación de usuarios.
5. La utilización del Sistema Operativo Android, permite el uso del software Google Maps que se encuentra integrado, así como añadirle funcionalidades.
6. Chip RFID que permite identificar el vehículo en el aparcamiento.
7. Lector de tarjetas de memoria SD/SDHC para almacenar datos (mapas, puntos de interés o perfiles de configuración).
8. Brújula digital.
9. Altavoces para escuchar las indicaciones.
10. Integración con el vehículo que permite el uso de sensores de aparcamiento, cámara de video posterior y señales de alarma.

Es un dispositivo hardware que todavía se encuentra en fase de prototipo.

Figura 11: *Prototipo Parking Finder*

3.2.6. SPark

Sistema de aparcamiento inteligente basado en redes vehiculares VANET (acrónimo en inglés que refiere al tipo de comunicación que utiliza a los vehículos como nodos de la red) [13]. Ofrece al usuario información en tiempo real sobre las plazas libres y le guía hasta éstas mediante signos actualizados dinámicamente.

Este sistema obtiene la disponibilidad de las plazas de aparcamiento usando los sensores instalados a lo largo del parking. Sin embargo, instalar sensores en un gran parking puede ser muy caro. Además dichos sensores, con el tiempo, pueden cometer imprecisiones y hasta dejar de funcionar. Por lo tanto, es deseable tener un sistema seguro y con una buena relación calidad/precio para informar acerca de las plazas libres de aparcamiento y guiar a los conductores hasta ellas.

Con el avance y el extenso alcance de las tecnologías de comunicación inalámbricas, muchas multinacionales automovilísticas y de las telecomunicaciones se preparan para equipar cada coche con un *On Board Unit* (OBU) que es un sistema de comunicaciones integrado y que permite a los vehículos que las equipan comunicarse entre si, así como con las infraestructuras instaladas en las carreteras para el mismo propósito.

Por lo tanto, se hace posible monitorizar la ocupación de los aparcamientos, guiar a los conductores a las plazas libres y proporcionar protección antirrobo en grandes aparcamientos a través de comunicaciones vehiculares.

El esquema de SPARK se caracteriza por utilizar los *Road Side Units* (RSU) de las plazas de aparcamiento para realizar la vigilancia y la gestión del parking entero utilizando la tecnología de comunicación VANET.

En primer lugar, el esquema de SPARK puede proporcionar un servicio de navegación por los parkings en tiempo real, con esto, los conductores pueden encontrar rápidamente las plazas libres.

En segundo lugar, SPARK proporciona protección antirrobo basada en sistemas inteligentes VANET. Todos estos vehículos estacionados están vigilados mediante las RSUs, de manera que si uno es retirado ilegalmente, dichas RSUs detectan la anomalía rápidamente.

En tercer lugar, el esquema de SPARK puede proporcionar información a los vehículos en movimiento acerca de la localización de las plazas libres, de modo que los conductores puedan elegir rápida y adecuadamente la plaza libre en la que desean estacionar.

Por último, este esquema asegura la privacidad de las OBUs que es el requerimiento básico de seguridad en las comunicaciones VANET.

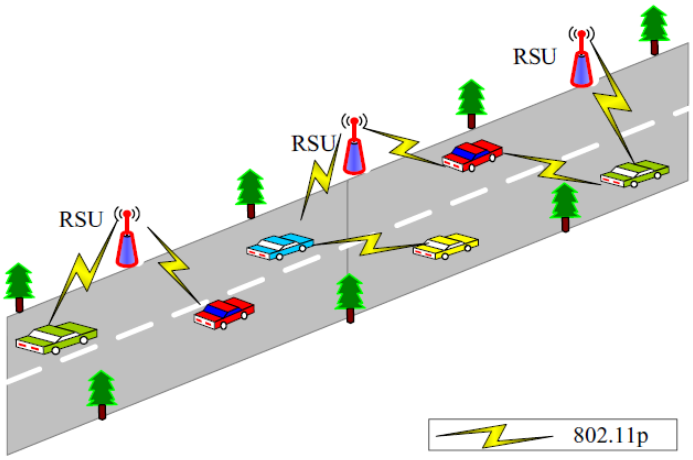


Figura 12: *Esquema funcionamiento SPark*

3.2.7. ePark

Se trata de un sistema que permite la búsqueda de plazas de aparcamiento libres en la vía pública. Este sistema lo forman una red de sensores instalados en las plazas de aparcamiento y una aplicación para dispositivos móviles. Actualmente, el sistema sólo dispone de infraestructura en zonas de estacionamiento regulado en Madrid (zona azul) [14].

La aplicación móvil permite a los usuarios buscar plazas de aparcamiento libre en las zonas en las que se dispone de los sensores necesarios para cubrir la demanda. Además, mediante la aplicación para dispositivos móviles se podrán realizar las siguientes acciones adicionales:

- Pagar la tarifa del estacionamiento regulado.
- Alargar el tiempo de estacionamiento sin volver al vehículo.
- Anular denuncias.
- Consultar estacionamientos.
- Posibilidad de estacionar de manera gratuita gracias a los tickets ofrecidos por otros usuarios.

Se trata de una iniciativa desarrollada por un estudiante de la Universidad Pontificia de Comillas (Icade).

3.3.Comparativa

Después de haber expuesto las distintas alternativas, en cuanto a aplicaciones de búsqueda de aparcamiento, en este apartado se presenta una tabla comparativa de dichas aplicaciones, resumiendo y comparando sus principales características.

Característica / Dispositivo	OPEN SPOT	SF PARK	APARCAMIENTOS ESPAÑA	AA PARKING	PARKING FINDER	SPARK	ePark
CARACTERÍSTICAS GENERALES							
Empresa Responsable	Google	SFMTA SD Park	Compound Eyes	Parkopedia	Martín Chamarro	IEEE	Gonzalo Boada
Lugares	Estados Unidos, Canadá y Países Bajos	San Francisco	España	Reino Unido e Irlanda	-	Lugares con la Infraestructura necesaria	Lugares con la infraestructura necesaria. Por ahora Madrid.
Tipo de plazas	Las notificadas por el usuario	Normales y Garaje	-	Pago y Libre	-	Pago	Pago (Zona Azul)
Tipo de dispositivo	Android 2,0 o superior	Android e iOS	Android	Android	Dispositivo GPS con Android	Dispositivos compatibles con VANET	Android, iOS, Win. Phone, Firefox OS y BlackBerry
Coste Aplicación	Gratuita	Gratuita	0,50€	2,35€	Sin determinar (todavía prototipo)	Depende del Vehículo	Gratuita
BÚSQUEDA DE APARCAMIENTO							
Localización de plazas libres	GPS y 3G	Sensores instalados en las plazas	GPS y 3G	GPS y 3G	GPS y 3G	Redes VANET	Sensores instalados en las plazas
Sistema Antirrobo	No	No	No	No	Sí	Sí	No
Búsqueda de Dirección	No	No	Sí	Sí	Sí	No	-
Inicia navegador automáticamente	Sí	No	Sí	No	Sí	Si (indica posición de las plazas libres dentro del parking)	-
Regula precio estacionamientos	No	Sí	No	No	No	No	No
OTRAS CARACTERÍSTICAS							
Integración con el Vehículo	No	No	No	No	Sí	Sí, con OBU	No
Necesita infraestructura Específica	No	Sensores en las plazas de aparcamiento	No	No	Dispositivo GPS	RSU (Roadside Unit) y OBU (On Board Unit)	Sensores en las plazas de aparcamiento
Permite el pago del estacionamiento	-	No	-	Sólo consulta de tarifas	-	No	Sí

Tabla 4: *Comparativa de distintas aplicaciones de gestión de plazas de aparcamiento*

En cuanto a las aplicaciones presentadas en la tabla comparativa destacan sobretodo SPARK y Parking Finder. Son las dos más completas, ya que implementan la mayoría de las características importantes para este tipo de

sistemas de búsqueda de aparcamiento. Además, ambas, proporcionan un sistema antirrobo del que las demás aplicaciones carecen y resulta un valor añadido. El mayor problema de estos sistemas es que necesitan de dispositivos específicos para la recepción de la señal y búsqueda de aparcamiento, ya que SPARK necesitaría de un vehículo compatible con redes VANET y Parking Finder necesitaría de la compra de un dispositivo GPS específico diseñado para tal cometido.

La aplicación desarrollada en el presente proyecto, permite la utilización del sistema en cualquier dispositivo Android, sin necesidad de disponer de sensores en las plazas (ePark y SFPark los necesitan) ni cualquier otro tipo de infraestructura específica (SPark necesita una red VANET), sólo necesita la colaboración de los usuarios.

Otro punto a favor de nuestra aplicación con respecto al resto es que, a pesar de estar pensada para ser utilizada en ciudades españolas, no tiene ningún tipo de limitación geográfica, por lo que se podría utilizar en cualquier lugar del mundo siempre y cuando tengamos cobertura GPS y acceso a Internet.

4. ANÁLISIS

Este apartado del documento trata sobre la fase de análisis de la aplicación. El contenido del mismo consiste en un documento de especificación de requisitos. A lo largo del desarrollo del apartado se hará referencia a él como "documento".

4.1.Introducción

El presente documento contiene una Especificación de Requisitos de Software (ERS) de la aplicación informática *Aplicación para la Gestión Automática y Eficiente del Estacionamiento en las Ciudades Españolas* desarrollada en el Trabajo Fin de Carrera (TFC) del mismo título. La estructuración de dicha Especificación de Requisitos de Software ha sido realizada siguiendo las directrices que se marcan en el estándar IEEE Recommended Practice for Software Requirements Specifications IEEE 830-1998 [15].

4.1.1. Propósito

El propósito de este documento es facilitar el desarrollo de la "Aplicación para la Gestión Automática y Eficiente del Estacionamiento en las Ciudades españolas" (en lo sucesivo, la aplicación), y su posterior implementación, presentando sus funcionalidades así como las posibles restricciones. Este documento va dirigido tanto a desarrolladores como a los clientes del producto.

4.1.2. Alcance

El objetivo es obtener un sistema software que permita una gestión automática y eficiente de los espacios de aparcamiento dentro de las ciudades españolas. De este modo los usuarios registrados podrán compartir y ocupar plazas de aparcamiento, así como, consultar el historial de plazas ocupadas recientemente.

Además del software, se realizará también la especificación del hardware necesario para su ejecución con unos tiempos de respuesta que se consideren aceptables.

Se requerirá que toda la documentación referente al proceso de diseño sea conforme al estándar UML.

4.1.3. Definiciones, acrónimos y abreviaturas

En este apartado se proporcionan las definiciones de todos los términos, acrónimos y abreviaturas necesarios para interpretar adecuadamente el presente Documento de Especificación de Requisitos de software. De este modo, tenemos los siguientes términos:

Vehículo: Medio de transporte de personas o cosas. Particularmente puede ser un automóvil que se define como un vehículo que puede ser guiado para marchar sin necesidad de carriles y lleva un motor; generalmente de explosión, que lo pone en marcha.

GPS (*Global Positioning System*): sistema de posicionamiento global, es un sistema que permite determinar, mediante la triangulación con satélites, la posición de un objeto o persona en el mundo.
Plaza de aparcamiento: superficie, terreno o local acotado destinado a estacionar, aparcar o guardar vehículos a motor.

Autenticar: es el acto de establecimiento o confirmación de algo (o alguien) como auténtico. Concretamente nos referimos, en el presente documento, al acto de verificar la identidad y los datos de acceso de un usuario.

Email: o correo electrónico, es un servicio de red que permite a los usuarios enviar y recibir mensajes y archivos mediante sistemas de comunicación electrónicos.

Google Maps: Herramienta desarrollada por Google que nos permite la visualización de mapas, así como la interacción con ellos.

Android SDK: Android Software Development Kit, es un conjunto de herramientas de desarrollo de software que le permiten a un programador crear aplicaciones para dispositivos Android.

PostgreSQL: es un Sistema Gestor de Bases de Datos (SGBD) relacional orientado a objetos y libre (está publicado bajo la licencia BSD).

PostGIS: es un módulo que añade soporte para objetos geográficos al Sistema Gestor de Bases de Datos PostgreSQL.

XML (*Extensible Markup Languaje):* lenguaje de marcas extensible en castellano, es un metalenguaje que sirve para almacenar datos de forma estructurada.

4.1.4. Referencias

Se indican a continuación los documentos o estándares que se han seguido para el análisis de la aplicación:

"IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications", IEEE Computer Society, 1998.

4.1.5. Resumen

El resto del documento está estructurado de la siguiente manera:

- **Descripción global:** apartado en el que se proporciona una visión general acerca de las funcionalidades de la aplicación.

- **Características del Producto:** Descripción general del sistema a desarrollar, con el fin de indicar los factores que afectan al producto y sus requerimientos. Proporciona las principales características a cumplir por el producto.

- **Requisitos específicos:** en este apartado se describen las funcionalidades de la aplicación de un modo más técnico que en el primer apartado, ya que va dirigido principalmente a los desarrolladores.

4.2.Descripción global

En esta sección se describirán los factores generales que afectan al producto y a sus requisitos.

4.2.1. Perspectiva del producto

Para que la "Aplicación para la Gestión Automática y Eficiente del Estacionamiento en las ciudades españolas" lleve a cabo su cometido, es necesario el uso, por parte de esta, de una serie de componentes software que dan como resultado una aplicación cliente y otra aplicación servidor. La estructura general del sistema es la siguiente:

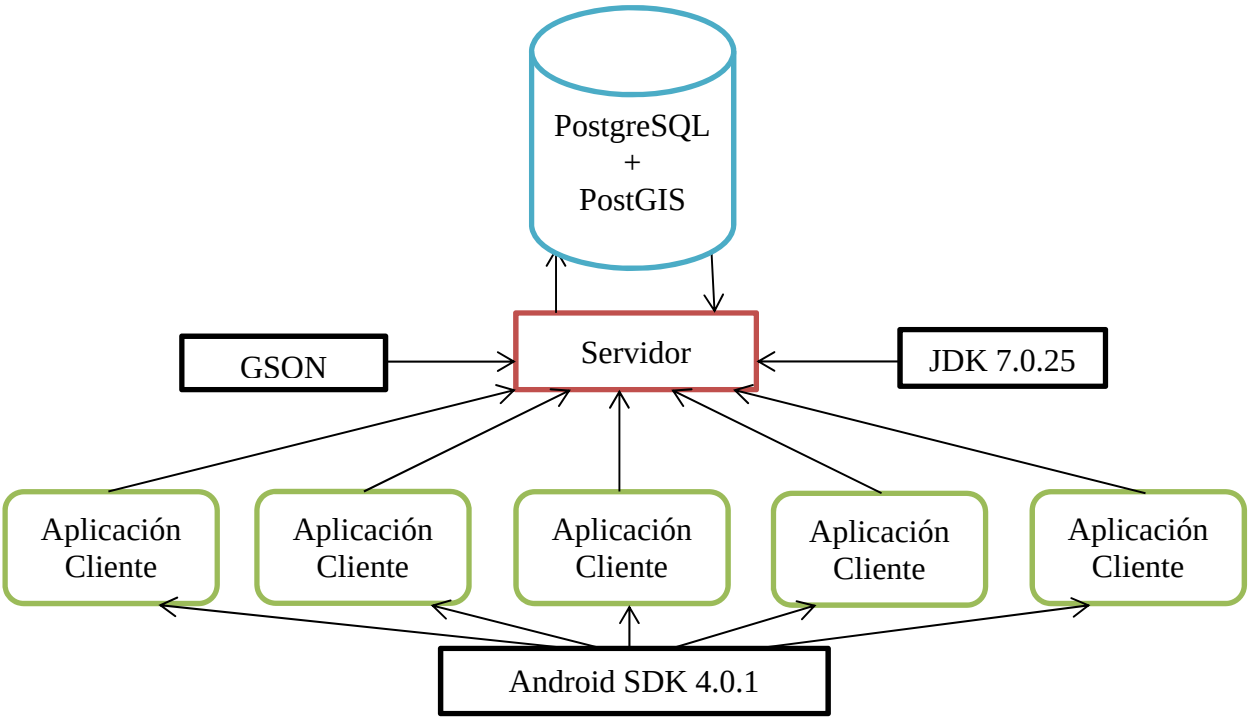


Figura 13: *Estructura general de la aplicación*

Como se observa en el diagrama anterior, cada una de las aplicaciones cliente interactuará con la aplicación servidor a través de sockets, permitiendo así, solicitar los datos requeridos además de interactuar con la base de datos.

Aparte de la aplicación servidor, también interactuará con la base de datos propia, que está incluida en el dispositivo móvil, esta base de datos sirve de apoyo, y en ella se pueden albergar datos relativos a las plazas ocupadas recientemente o al usuario que ha realizado login en ese terminal.

Por otro lado, la aplicación servidor es la encargada de recibir peticiones del cliente, y de devolverle los datos necesarios.

Interfaces del sistema

La "Aplicación para la Gestión Automática y Eficiente del Estacionamiento en las Ciudades españolas" dispondrá de una aplicación para dispositivos móviles Android, lo cual, permitirá acceder a todas las funcionalidades del sistema.

Interfaces de usuario

En cuanto a interfaces de usuario, consideraremos los siguientes aspectos:

- La aplicación está diseñada para ser ejecutada en dispositivos móviles con un tamaño de pantalla medio o grande, y una densidad de píxeles media o alta, para los cuales se garantiza una correcta visualización.
- Se utilizarán criterios de usabilidad tales como fuentes adecuadas, colores con suficiente contraste entre ellos y que no sean pesados para la vista…
- Se requerirá Android 4.0.1 o superior.

Interfaces hardware

El sistema deberá tener en cuenta las diferentes interfaces hardware que se utilizarán, tal y como se refleja en el siguiente diagrama de despliegue:

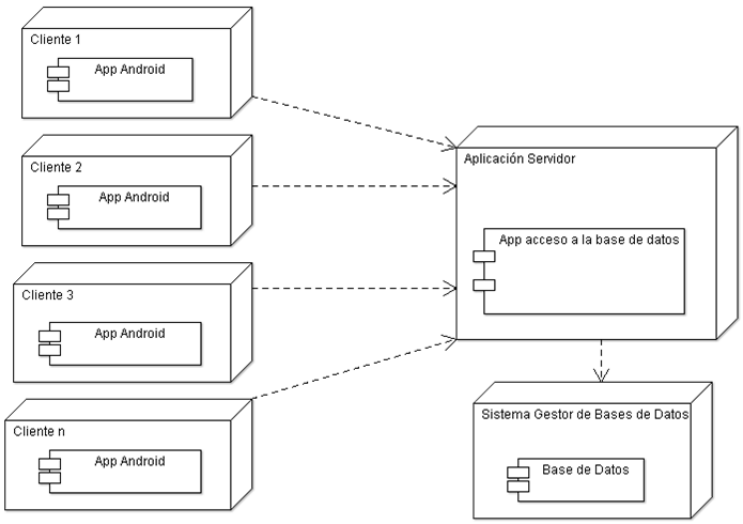


Figura 14: *Diagrama de despliegue*

Los usuarios se conectarán a un único servidor a través de la aplicación cliente. Esta conexión se realizará mediante sockets que transmitirán información encriptada. A su vez, el servidor accederá a una base de datos en la que se almacenará toda la información relativa a las plazas de aparcamiento y

los usuarios. Aunque en el diagrama se ha colocado en un equipo aparte, el almacenamiento persistente podrá estar ubicado en el mismo equipo que la aplicación servidor.

Este será un sistema de alta disponibilidad que contará con todos los servicios duplicados en 2 servidores distintos con una tarjeta de red cada uno conectados a dos redes TCP/IP distintas y un SAI (Sistema de Alimentación ininterrumpida que permitirá ejecutar las aplicaciones con cortes de tensión) que evitará la falta de energía en el servidor.

Interfaces software

En cuanto a la interfaz software, dependerá de si hablamos del cliente o del servidor, por ello tenemos:

- Aplicación cliente:
  - Sistema operativo Android 4.0.1 o superior.
- Aplicación servidor:
  - JRE 7
  - Conectividad con la base de datos PostgreSQL, que puede estar en la misma o en distinta máquina que la aplicación servidor.

Interfaces de comunicación

En lo referente a las comunicaciones, tendremos dos tipos:

- Una comunicación entre el cliente y el servidor, cuando los clientes solicitan datos y el servidor les envía la respuesta. Esta comunicación entre cliente y servidor se realizará usando el protocolo de comunicación TCP/IP y se establecerá a través de sockets. Además la información será encriptada antes de enviarla.

- El servidor deberá poder comunicarse con la base de datos, la cual podrá encontrarse en el mismo equipo que el servidor o no, por lo que deberá poderse acceder a ella de forma remota.

Restricciones de memoria

El equipo donde se ejecute la aplicación servidor (teniendo la base de datos en otra máquina), deberá tener al menos 4GB de memoria RAM y 100MB de espacio libre en el disco duro como mínimo para que se ejecute de manera. En el caso de que la base de datos se encuentre en el mismo servidor, el espacio en disco necesario variará dependiendo del tamaño de dicha base de datos.

En el caso de la aplicación cliente, los terminales no tienen restricción de memoria RAM, ya que, con cualquier terminal de hoy en día, incluso con los de gama de entrada, se podría ejecutar la aplicación de forma fluida. En cuanto a almacenamiento interno se refiere, serán necesarios 5MB libres, como mínimo.

Casos de uso

En el siguiente diagrama se muestran los casos de uso de los usuarios de la aplicación:

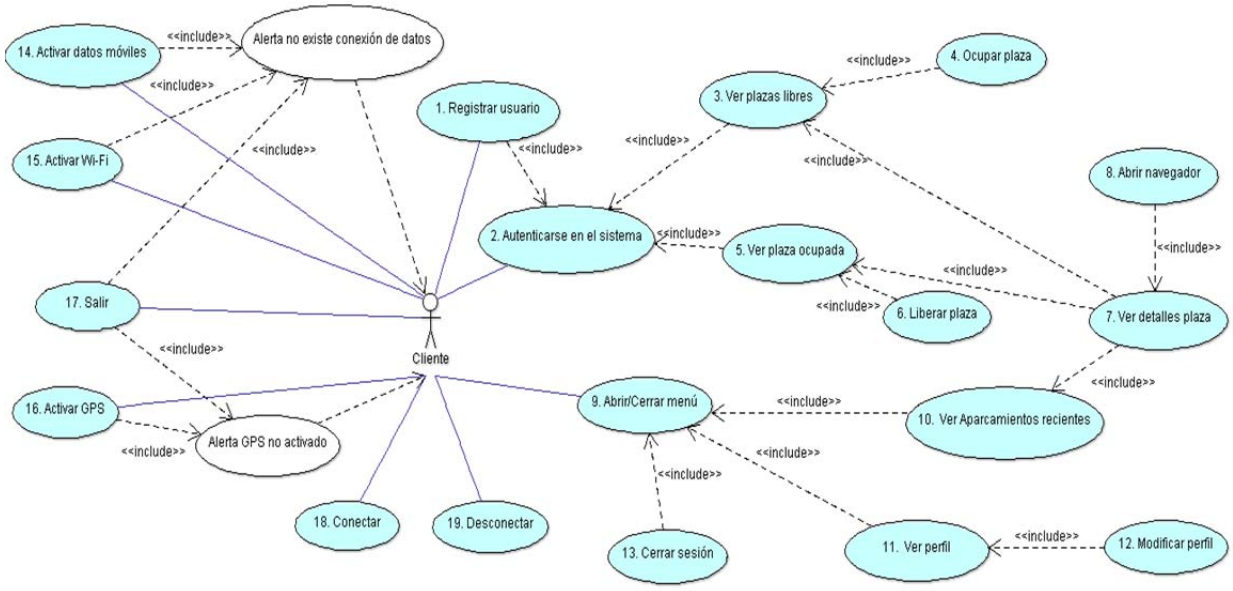


Figura 15: *Diagrama de casos de uso*

Cada caso de uso esta numerado para facilitar su referencia.

CU 1. Registrar usuario

La interfaz de usuario deberá ofrecer la posibilidad de que los usuarios puedan registrarse en el sistema. Para ello deberán rellenar un formulario con sus datos personales (nombre y apellidos), nombre de usuario, un email de contacto y la contraseña.

CU 2. Autenticarse en el sistema

Los usuarios no podrán interactuar con el sistema sin haber superado previamente el proceso de autenticación. Para ello, la interfaz de usuario ofrecerá un formulario que posibilite a los usuarios, una vez registrados, autenticarse en el sistema tantas veces como deseen.

CU 3. Ver plazas libres

Los usuarios que se hayan autenticado en el sistema y, no estén estacionados, podrán visualizar las plazas libres a su alrededor e interactuar con ellas.

CU 4. Ocupar plaza

Los usuarios deberán poder ocupar una plaza de aparcamiento siempre que estén en la sección que indica el CU 3.

CU 5. Ver plaza ocupada

Los usuarios que hayan superado la autentificación en el sistema y, estén estacionados, visualizarán el lugar donde han realizado el estacionamiento.

CU 6. Liberar plaza

Los usuarios deberán poder liberar la plaza de aparcamiento que ocupan, para ello deberán estar estacionados (CU 5).

CU 7. Ver detalles plaza

Al igual que en el CU 10, desde los casos de uso CU 3, CU 5 y CU 9.1, el usuario podrá visualizar todos los detalles de una plaza tocando con el dedo la "chincheta" que la simboliza

CU 8. Abrir Navegador

Partiendo de los casos de uso CU 3, CU 5 y CU 9.1, el usuario podrá abrir el navegador predeterminado del dispositivo para ser guiado hasta la plaza seleccionada.

CU 9. Abrir/Cerrar menú

Tras haberse autenticado en el sistema (CU 2), un usuario puede abrir y cerrar el menú de la aplicación en cualquier momento, cuando desee acceder a los supuestos contemplados en los casos de uso 10, 11, 12 y 13.

CU 10. Ver Aparcamientos recientes

Se mostrará un mapa en el que aparecerán todas las plazas que se han ocupado recientemente.

CU 11.Ver perfil

Se mostrará una vista con todos los datos del perfil, exceptuando la contraseña.

CU 12. Modificar perfil

Se mostrará un formulario en el que el usuario podrá modificar sus datos personales.

CU 13. Cerrar sesión

En el momento que lo desee, un usuario autenticado podrá cerrar su sesión en ese dispositivo. De este modo quedará marcado el usuario como inactivo y se cargará la pantalla de inicio de sesión para poder autenticarse con el mismo u otro usuario.

CU 14. Activar datos móviles

Tras haber sido informado de que no dispone de conexión a Internet, mediante un mensaje de alerta, el usuario podrá abrir la pantalla de configuración Redes Móviles, y activar esta conectividad.

CU 15 Activar Wi-fi

Al igual que en el caso de uso número 12, el usuario será informado de que no dispone de conectividad a Internet, y se le ofrecerá la opción de abrir la pantalla de configuración del Wi-fi, pudiéndolo activar y conectarse a una red para disponer de conexión a Internet.

CU 16. Activar GPS

De forma similar a los mensajes de alerta que originan los casos de uso 12 y 13, se informará al usuario de que es necesaria la conexión a internet mediante un mensaje de alerta y, a su vez, se le dará la posibilidad de acceder a los *Ajustes de Ubicación* para activarlo.

CU17. Salir

Aparte de las opciones citadas en los casos de uso 12, 13 y 14, en los mensajes de alerta, también se dará al usuario la opción de salir de la aplicación. Esta opción se proporciona por si no dispone de una tarifa de datos o conexión Wi-fi, o simplemente, no quiere hacer uso de ellas o del GPS.

CU18. Conectar

La aplicación cliente se conectará al servidor cada vez que necesite realizar una operación (autenticarse, obtener plazas, ocupar y liberar plazas, actualizar el perfil…).

CU19. Desconectar

La aplicación cliente cerrará la conexión con el servidor tan pronto como haya realizado la operación requerida tras la conexión (CU16).

Debido a que los usuarios solo interactúan con el servidor a través de la aplicación cliente, se considera que dicho servidor no tendrá casos de uso con respecto al usuario.

4.2.2. Características del Producto

Características Generales

CAR_GEN_01: El sistema deberá poder manejar un número indeterminado de usuarios.
CAR_GEN_02: El sistema dispondrá de una base de datos para almacenar las plazas que se han notificado como libres y los usuarios registrados en el sistema.
CAR_GEN_03: La base de datos estará alojada en un servidor central.
CAR_GEN_04: Las aplicaciones clientes accederán a los datos a través de una aplicación servidor.
CAR_GEN_05: La aplicación deberá estar disponible las 24 horas del día 365 días al año.
CAR_GEN_06: El sistema deberá estar siempre conectado a Internet.
CAR_GEN_07: Los usuarios de este sistema deberán poder acceder a él identificándose mediante su nombre de usuario y su contraseña.
CAR_GEN_08: Los nombres de usuario deberán de ser únicos.
CAR_GEN_09: Las contraseñas que utilice cada usuario deberán tener al menos 8 caracteres.

Características de la aplicación de usuario

CAR_USR_01: Los usuarios deberán poder registrarse en el sistema.
CAR_USR_02: Los usuarios deberán poder autenticarse en el sistema mediante su nombre de usuario y su contraseña.
CAR_USR_03: Los usuarios deberán poder visualizar en el mapa las plazas libres próximas a su situación.
CAR_USR_04: Los usuarios podrán ver la dirección de la plaza libre o ser guiados por el navegador del dispositivo.
CAR_USR_05: Cada usuario deberá poder notificar que deja libre u ocupa una plaza de aparcamiento.

Características de la aplicación servidor.

CAR_SER_01: La aplicación servidor será la única que tendrá acceso a la base de datos.
CAR_SER_02: Determinará las plazas libres que se encuentran dentro del rango indicado.
CAR_SER_03: Deberá ser capaz de servir las plazas libres próximas a todos los usuarios conectados en ese momento.
CAR_SER_04: Deberá ser capaz de registrar usuarios.
CAR_SER_05: La aplicación servidor deberá validar las peticiones de autenticación de los usuarios.

4.2.3. Características del usuario.

La aplicación se dirige a todos aquellos usuarios que posean un dispositivo Android (Teléfono inteligente o Tablet generalmente) y que se encuentran en disposición de buscar una plaza de aparcamiento. Estos usuarios no tiene por qué tener conocimientos avanzados de informática, por lo que el uso de la aplicación deberá simplificarse todo lo posible, proporcionando una interfaz de usuario agradable y sencilla.

4.2.4. Restricciones.

De diseño.

RES_DIS_01: Se requiere el uso de diagramas UML.

De implementación.

RES_IMP_01: La aplicación cliente deberá estar escrita con Java utilizando el SDK de Android, concretamente la API nivel 14.
RES_IMP_02: La aplicación servidor deberá estar escrita en Java utilizándola versión 7.0 Revisión 25 del JDK.

RES_IMP_03: Se deberá usar PostgreSQL con el módulo PostGIS para la creación de la base de datos.
RES_IMP_04: Se deberá utilizar un dispositivo Android (Teléfono inteligente o Tablet) para la triangulación de la posición mediante el GPS.

Del sistema.

RES_SIS_01: Se requerirá Android 4.0 o superior en los terminales que ejecuten la aplicación.
RES_ SIS_02: La aplicación servidor deberá ejecutarse en una maquina accesible a través de internet para que los usuarios puedan conectarse.
RES_ SIS_03: La base de datos deberá estar alojada en un equipo al que tenga acceso la aplicación servidor.
RES_ SIS_04: Se utilizara la API de Google Maps, por lo que deberán cumplirse las limitaciones impuestas por la licencia de ésta.

De usabilidad.

RES_USA_01.- La interfaz debe ser clara, sencilla y nada sobrecargada, facilitando su uso a los conductores.
RES_USA_02.- La interfaz deberá estar adaptada para que se visualice correctamente en distintos dispositivos móviles con distintas resoluciones.

4.2.5. Supuestos y dependencias

Para el correcto funcionamiento de la aplicación, se da por supuesto que los componentes indicados en el apartado Interfaces Software están instalados y configurados correctamente. También se presupone que tanto el cliente como el servidor tendrán acceso a internet.

4.3.Especificación de requisitos

4.3.1. Requisitos funcionales de la Aplicación Cliente

Requisitos de interfaz.

RQ_AD_C_01: La aplicación se desarrolla para dispositivos con pantalla de 4,8 pulgadas, con una resolución de 720 x 1280 píxeles. En el caso de dispositivos con pantalla más grande o más pequeñas y una resolución menor, no se garantiza su correcta visualización.
RQ_AD_C_02: El icono que hace referencia a la aplicación en el menú del dispositivo Android será el siguiente:

Figura 16: *Icono de la aplicación*

RQ_AD_C_03: Todas las pantallas de la aplicación tendrán el mismo "Look and Feel", al igual que una estructura homogénea. En la medida de lo posible, el usuario deberá poder efectuar su operación en 3 o menos movimientos.
RQ_AD_C_04: La navegación por la aplicación será sencilla y eficiente.
RQ_AD_C_05: El usuario deberá saber en todo momento, en qué sección de la aplicación se encuentra.
RQ_AD_C_06: El usuario deberá poder visualizar, cuando lo desee, las plazas libres que se encuentran en sus proximidades.
RQ_AD_C_07: Pulsando sobre los marcadores de las plazas notificadas como libres, el usuario podrá elegir entre ver la dirección y ser guiado por el Navegador GPS del dispositivo.
RQ_AD_C_08: El usuario deberá poder notificar que abandona una plaza de aparcamiento.
RQ_AD_C_09: El usuario deberá poder acceder a un apartado de configuración donde podrá adaptar los parámetros de la aplicación a sus necesidades.
RQ_AD_C_10: Será necesario pedir confirmación cuando se desee salir de la aplicación.
RQ_AD_C_11: La aplicación notificará al usuario cuando no se tenga acceso a la red (WiFi o 3G), y cuando el GPS esté desactivado.
RQ_AD_C_12: En ningún momento habrá un desplazamiento en Scroll horizontal.

Requisitos de autenticación.

RQ_AD_C_13: Los usuarios deberán autenticarse en el sistema mediante su nombre de usuario o email y su contraseña.

Requisitos de comunicación.

RQ_AD_C_14: Existirá un servidor al que se conectarán los clientes para la obtención de los datos.
RQ_AD_C_15: Las aplicaciones cliente se conectarán al servidor cuando deseen obtener las plazas libres o notificar que una plaza se libera.
RQ_AD_C_16: Las aplicaciones cliente, cuando deseen obtener las plazas libres, deberán enviar ciertos parámetros de configuración al servidor, como por ejemplo la distancia de las plazas libres que desea conocer.

Requisitos de opciones.

RQ_AD_C_17: El usuario, al visualizar las plazas libres en el mapa, dispondrá la opción de ver la dirección o de ser guiado por el navegador GPS del dispositivo.
RQ_AD_C_18: El usuario dispondrá de la opción de editar su perfil de usuario.

4.3.2. Requisitos no funcionales de la Aplicación Cliente

Requisitos de rendimiento.

RQ_AD_C_19: El tiempo desde que se solicita la información hasta que se recibe no debe ser superior a 30 segundos.
RQ_AD_C_20: La ejecución de la aplicación debe hacerse de una manera fluida.

Requisitos de seguridad.

RQ_AD_C_21: Al contener información personal y contraseñas, la comunicación entre el cliente y el servidor deberá estar encriptada.
RQ_AD_C_22: Será necesario estar autenticado para poder editar el perfil de un usuario.

Requisitos de privacidad.

RQ_AD_C_23: Se respetará la ley de protección de datos de carácter personal (LOPD) vigente en este momento, Ley orgánica 15/1999, del 13 de diciembre.
RQ_AD_C_24: Cada persona tendrá acceso únicamente a la información a la que esté autorizada.

Requisitos de entrega.

RQ_AD_C_25: El plazo de entrega de la aplicación previsto es antes de finalizar el curso 2013-2014, en Septiembre de 2014.

Requisitos de usabilidad.

RQ_AD_C_26: La interfaz debe seguir los estándares de usabilidad.
RQ_AD_C_27: La aplicación del cliente debe ser clara y sencilla para el uso.

4.3.3. Requisitos funcionales de la Aplicación Servidor

Requisitos de generales.

RQ_AD_S_01: El servidor deberá estar disponible las 24 horas del día 365 días al año.
RQ_AD_S_02: El sistema deber poder manejar un número indeterminado de usuarios.
RQ_AD_S_03: Cada petición de una aplicación cliente se responderá lo antes posible, liberando los recursos utilizados una vez haya sido satisfecha.
RQ_AD_S_04: El servidor almacena la información (coordenadas, hora, y si está libre u ocupada) de las plazas de aparcamiento.
RQ_AD_S_05: Las aplicaciones cliente, obtienen acceso al intercambio de datos cuando hacen una petición al servidor.
RQ_AD_S_06: Para poder acceder al servicio, será necesario autenticarse, para ello se deberá tener la aplicación instalada y haberse registrado previamente.

Requisitos de la base de datos.

RQ_AD_S_07: La base de datos a la que accederá la aplicación podrá estar en la misma máquina en la que está se esté ejecutando o en otra diferente.
RQ_AD_S_08: Se usará PostgreSQL con el módulo PostGIS como sistema gestor de bases de datos geográficas.
RQ_AD_S_09: La base de datos estará construida siguiendo el esquema entidad-relación indicado en la documentación.
RQ_AD_S_10: La base de datos deberá almacenar información acerca de todas las plazas que notifican los usuarios como libres u ocupadas.

Requisitos de la comunicación con la aplicación cliente.

RQ_AD_S_11: La aplicación del servidor deberá atender las peticiones realizadas por las aplicaciones cliente.
RQ_AD_S_12: La comunicación entre la aplicación cliente y el servidor se realizara utilizando un protocolo a definir.
RQ_AD_S_13: Las aplicaciones cliente deben establecer la conexión con el servidor únicamente el tiempo que lo necesiten.

4.3.4. Requisitos no funcionales de la Aplicación Servidor

Requisitos de rendimiento.

RQ_AD_S_14: La aplicación permitirá gestionar un número indeterminado de peticiones.

RQ_AD_S_15: El tiempo de respuesta máximo aceptable del servidor, omitiendo las operaciones con otros módulos, frente a una petición del cliente será de 2 segundos.

Requisitos de seguridad.

RQ_AD_S_16: Siempre que sea posible, se encriptarán los datos a enviar.

RQ_AD_S_17: Podrán acceder a los servicios de búsqueda de aparcamiento todos los usuarios que tengan instalada la aplicación y se hayan autenticado correctamente (RQ_AD_C_13).

RQ_AD_S_18: Para las comunicaciones se utilizara un protocolo únicamente conocido por los diseñadores de la aplicación.

RQ_AD_S_19: El sistema mantendrá un log con los accesos a los servicios ofertados por la aplicación.

RQ_AD_S_20: El sistema deberá tener constancia de la siguiente información:

- Cuando se realiza una operación y quién a realiza.
- Qué dirección IP tiene el usuario que realiza cierta acción.
- Qué acción realiza un usuario.

Requisitos de privacidad.

RQ_AD_S_21: Se respetará la ley de protección de datos de carácter personal (LOPD) vigente en este momento, Ley orgánica 15/1999, del 13 de diciembre.

Requisitos de entrega.

RQ_AD_C_22: El plazo previsto de entrega de la aplicación es antes de finalizar el curso 2013-2014, en Septiembre de 2014.

Requisitos de usabilidad.

RQ_AD_S_23: El idioma de la aplicación será castellano, pero se facilitará la adaptación a cualquier otro idioma.

5. PLANIFICACIÓN Y CÁLCULO DE COSTES

En el presente apartado se va a desarrollar la planificación realizada antes de acometer el desarrollo del proyecto y el cálculo de los costes que ocasionaría dicho desarrollo.

5.1. Planificación

Para realizar la planificación del presente proyecto, se ha tenido en cuenta cada una de las fases necesarias para finalizar exitosamente cualquier desarrollo software de cierta envergadura. Dichas fases son la de análisis, diseño, implementación y pruebas.

5.1.1. Planificación inicial

El diagrama de Gantt que se presenta a continuación, muestra los plazos previstos inicialmente para la realización del proyecto:

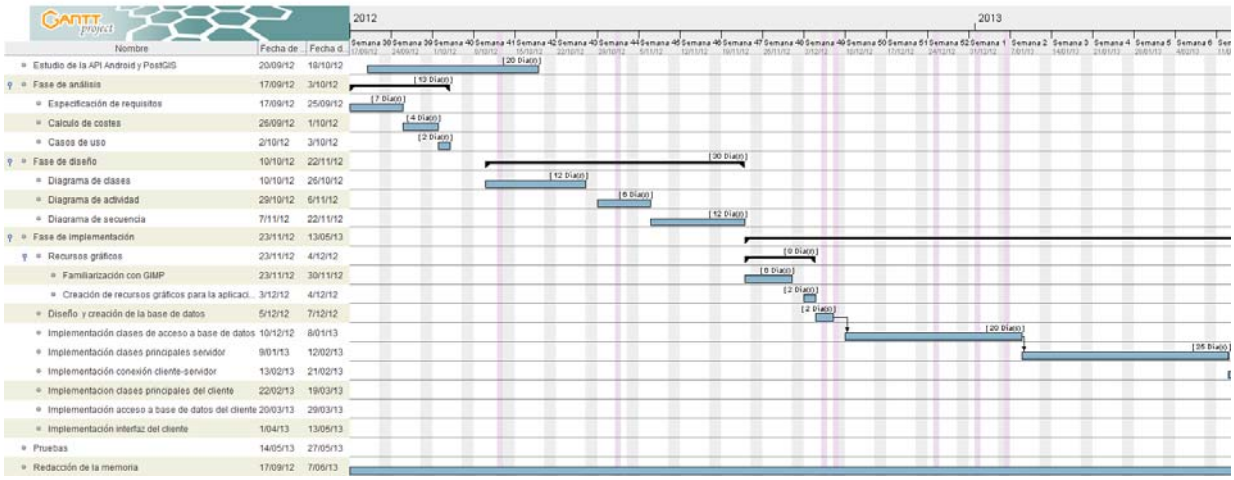


Figura 17: *Planificación inicial. Parte1*

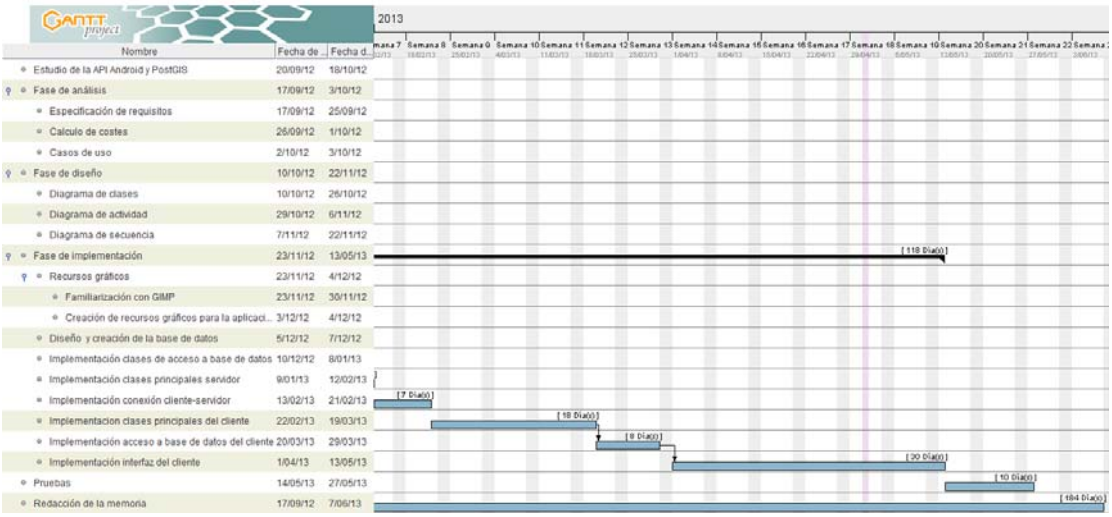


Figura 18: *Planificación inicial. Parte 2*

Como se puede ver en el diagrama, se plantean las siguientes tareas y su correspondiente duración:

- Estudio de la API de Android y PostGIS.
- Fase de análisis:
  - Especificación de requisitos.
  - Calculo de costes.
  - Casos de uso.
- Fase de diseño:
  - Diagrama de clases.
  - Diagramas de actividad.
  - Diagramas de secuencia.
- Recursos gráficos:
  - Familiarización con UIMP.
  - Creación de recursos gráficos para la aplicación.
- Fase de implementación:
  - Diseño y creación de la base de datos.
  - Implementación clases de acceso a base de datos.
  - Implementación clases principales del servidor.
  - Implementación conexión cliente-servidor.
  - Implementación clases principales del cliente.
  - Implementación acceso a base de datos del cliente.
  - Implementación interfaz cliente.
- Pruebas.
- Redacción de la memoria.

Aunque la mayoría de las tareas se realizan de forma secuencial, existen algunas que se solapan en el tiempo. La duración total estimada del proyecto es de 184 días trabajando de media 8 horas diarias.

Esta demora en el desarrollo del proyecto ha sido causada debido a las dificultades encontradas a la hora de desarrollar ciertas partes de la aplicación ya que, además, se tenía que lograr una compatibilidad entre los componentes Java de la aplicación servidor y, los componentes de la aplicación Android. Esto es debido a que el API de Android no incorpora todas las clases y métodos de Java además de que que en algunos casos, presenta incompatibilidades incluso entre distintos niveles de su propia API. Otra de las causas de la demora en cuanto a la fiscalización del proyecto ha sido mi incorporación al mercado laboral, que en un inicio fue únicamente los jueves y fines de semana pero desde Enero de 2014, gracias a haber obtenido una beca de la Diputación Provincial de Teruel, comencé a trabajar, de Lunes a Viernes, en el servicio de informática de dicha diputación sumándolo a mi anterior trabajo.

5.1.2. Planificación final

A continuación se muestra el diagrama de Gantt que muestra el calendario que realmente se ha seguido durante la realización del proyecto:

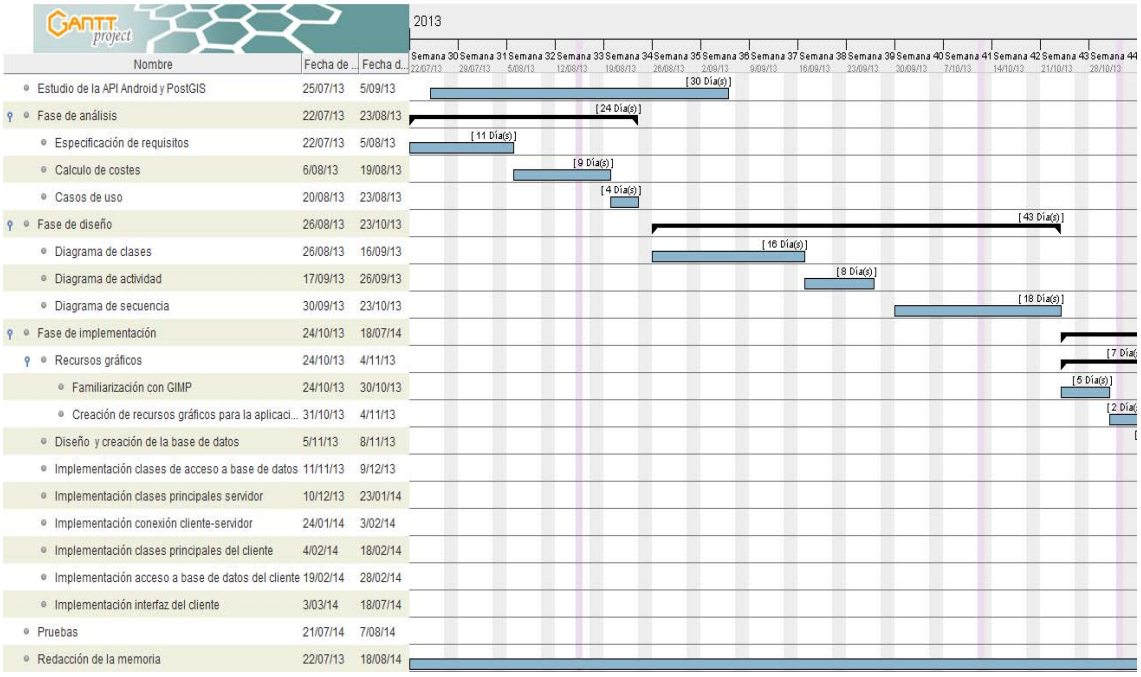


Figura 19: *Planificación final. Parte 1*

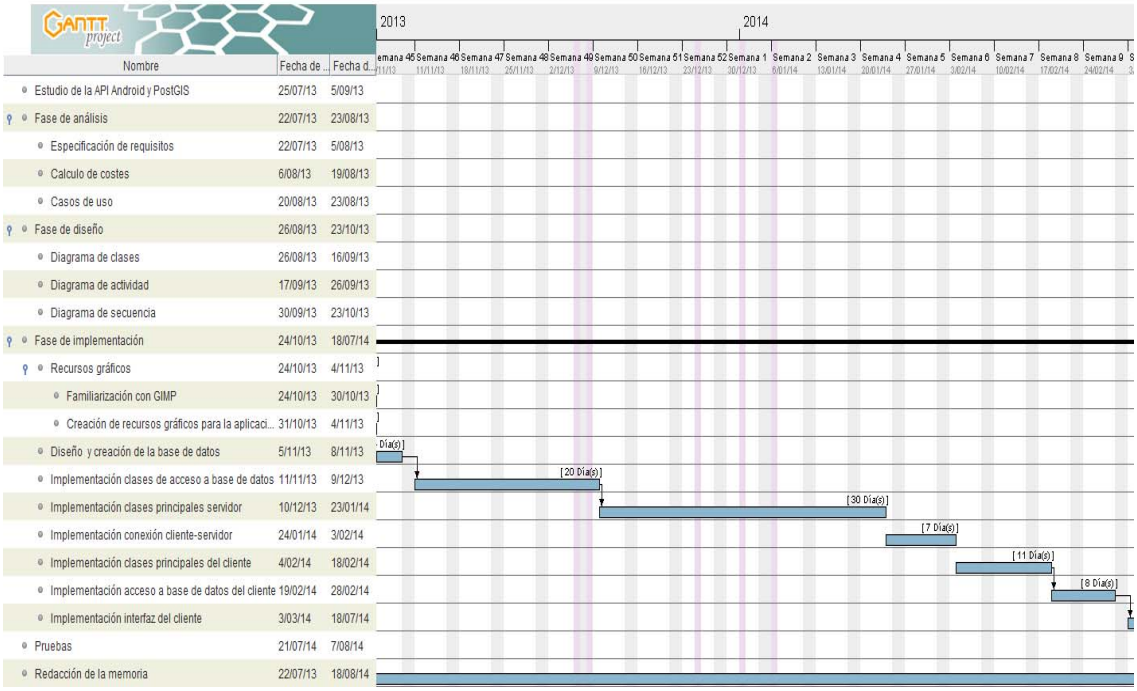


Figura 20: *Planificación final. Parte 2*

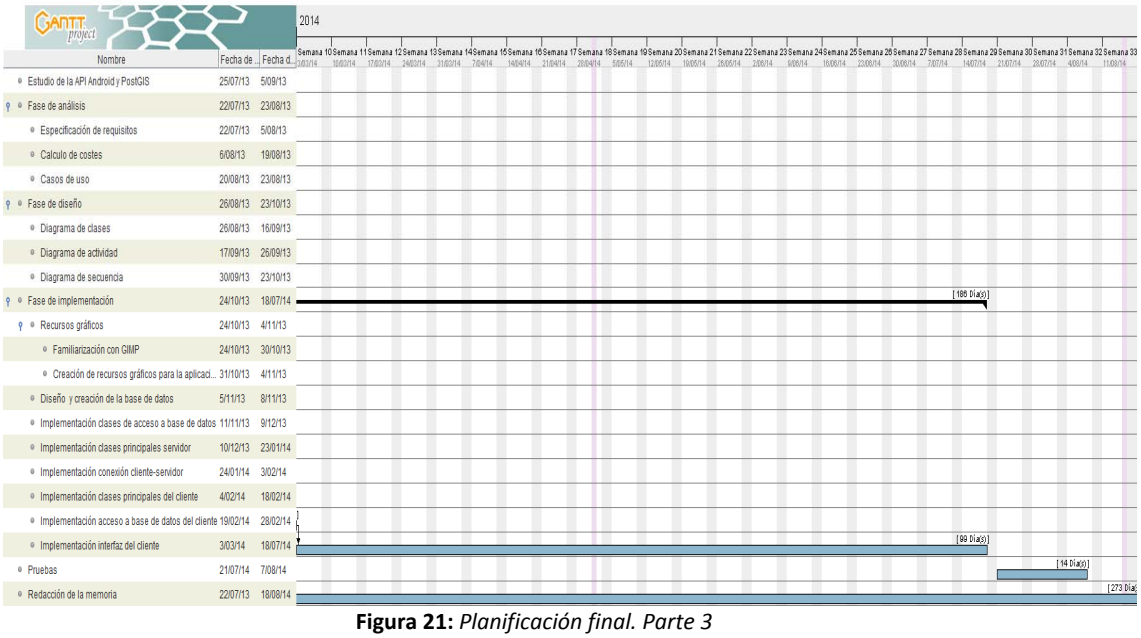


Figura 21: *Planificación final. Parte 3*

Como se puede observar, se han producido variaciones con respecto a la planificación realizada inicialmente, lo que indica que en numerosas ocasiones es difícil cumplir estrictamente los plazos marcados, ya sea por causas internas o externas al proyecto. Dichas causas internas pueden ser retrasos en la implementación de ciertas partes de la aplicación que se estimaban menos

costosas de lo que realmente eran y, las causas externas, pueden ser debidas a motivos personales o laborales, como es el caso, o cambios impuestos por el cliente en la aplicación.

La duración del proyecto ha sido de 273 días, 89 más de los planificados inicialmente habiendo trabajado 8 horas diarias de promedio.

La diferencia entre la planificación estimada y la real está causada por las dificultades encontradas a lo largo de la fase de implementación del proyecto. Parte de estas dificultades se han debido a incompatibilidades entre las librerías de Java y las de Android, las existentes entre los distintos niveles de API de Android correspondientes a cada una de sus versiones y la búsqueda de soluciones para dichas incompatibilidades. También se han producido retrasos en el desarrollo de ciertas partes de la aplicación que en un principio, se pensaba que iba a ser más sencilla su implementación y, una vez iniciada dicha implementación, se prolongó en el tiempo más de lo estimado.

La última causa del aumento de tiempo de ejecución del proyecto ha sido mi incorporación al mercado laboral, que en un inicio fue los Jueves y fines de semana y que actualmente, desde el mes de Enero gracias a la obtención de una Beca de la Diputación Provincial de Teruel, se ha ampliado junto con mi anterior trabajo, a toda la semana.

5.2. Cálculo de costes

En el desarrollo de aplicaciones informáticas, al igual que en otras actividades comerciales, es de vital importancia conocer los costes para calcular la rentabilidad de un producto. Por ello es necesario realizar una estimación de costes. El primer paso a realizar será utilizar la técnica de los puntos función [16] para medir el tamaño de la aplicación.

Este sistema de estimación de costes, consiste en asignar una cantidad de "puntos" a una aplicación informática según la complejidad de los datos que maneja y de los procesos que realiza sobre ellos, siempre tratando de considerarlo desde el punto de vista del usuario.

5.2.1. Tipo de recuento

El tipo de recuento que se llevará a cabo es para un proyecto de nuevo desarrollo, que consiste en una aplicación Android para la búsqueda de aparcamiento que interactúa con una aplicación servidor, también desarrollada.

5.2.2. Alcance del recuento y límites de la aplicación final

Dentro de los límites de la aplicación a efectos de recuento de puntos función se deben considerar los programas (o clases), las estructuras de datos y demás elementos del software. Se considerará que están fuera de los límites:

- El usuario final
- Ficheros de interface
- Informes o comunicados generados
- Mensajes a otra aplicación
- Páginas web enviadas
- Ficheros o procesos asignados a otras aplicaciones

5.2.3. Contar las funciones de datos

Para comenzar la estimación mediante puntos función, debemos obtener una serie de datos que nos facilitarán su cálculo. En primer lugar, obtendremos los datos funcionales. Estos datos están divididos en dos subconjuntos:

- *Internal Logic File (ILF):* Un ILF es un grupo de datos lógicamente relacionados, identificables por el usuario y mantenidos a través de procesos dentro de los límites de la aplicación.
- *External Interface File (EIF):* Un EIF es un grupo de datos lógicamente relacionados, identificables por el usuario y usados por la aplicación pero mantenidos dentro de los límites de otra aplicación.

A su vez, los dos tipos anteriores estan formados por:

- *Data Element Type (DET):* Los DETs corresponden a atributos o campos de los ficheros, incluyendo las claves ajenas.
- *Record Element Type (RET):* Los RETs corresponden a campos del fichero que referencian a otros ficheros.

La complejidad de los datos que maneja la aplicación, es calculada en base a la siguiente tabla, que asigna valores en función de los DET´s y RET´s que cada fichero maneja.

	1-19 DETs	20-50 DETs	>51 DETs
1 RET	Baja	Baja	Media
2-5 RETs	Baja	Media	Alta
>5 RETS	Media	Alta	Alta

Tabla 5: *Matriz de complejidad para ILFs y EIFS*

De este modo, tenemos los siguientes datos para nuestra aplicación:

Nombre del fichero	TIpo	DETs	RETs	Complejidad
Plaza compartida	ILF	4	3	Baja
Perfil	ILF	5	2	Baja
Historial	ILF	4	4	Baja
Acerca de	ILF	0	0	Baja
Vista puntuación	ILF	2	1	Baja
Posición	EIF	1	3	Baja
Plazas libres	EIF	4	4	Baja
Plaza ocupada	EIF	4	3	Baja

Tabla 6: *Estimación de la complejidad para ILFs y EIFs*

5.2.4. Contar las funciones transaccionales

El siguiente paso es identificar las distintas transacciones funcionales, divididas en:

- *External Input (EI): Es un proceso elemental que procesa datos o información de control que viene desde fuera de los límites de la aplicación. La intención principal de un EI es mantener uno o varios ILF's y/o cambiar la conducta del sistema.*

- *External Output (EO):* Es un proceso elemental que envía datos o información de control fuera de los límites de la aplicación. La intención principal de un EO es presentar información al usuario mediante lógica de proceso diferente a, y en adición a, la recuperación de datos o información de control. La lógica del proceso ha de contener al menos una fórmula matemática o cálculo o datos derivados. Un EO puede también mantener uno o más ILF's y/o cambiar la conducta del sistema.

- *External Inquiry (EQ):* Es un proceso elemental que envía datos o información de control fuera de los límites de la aplicación. La intención principal de un EQ es presentar información al usuario mediante la recuperación de datos o información de control desde un ILF o EIF. La lógica de proceso no contiene fórmulas matemáticas o cálculos y no crea datos derivados. Ningún ILF es mantenido mediante el proceso, ni la conducta del sistema es alterada.

A la hora de identificar las funciones transaccionales, debemos tener en cuenta el parámetro *File Types Referenced* (FTR), que es el número de ficheros a los que accede cada transacción.

A continuación se muestra una matriz a partir de la cual se calcula la complejidad:

Matriz de complejidad para EIs			
	1-4 DETs	5-15 DETs	>16 DETs
0 -1 FTR	Baja	Baja	Media
2 FTR	Baja	Media	Alta
>2 FTR	Media	Alta	Alta
Matriz de complejidad para EOs			
	1-5 DETs	6-19 DETs	>20 DETs
0-1 FTR	Baja	Baja	Media
2-3 FTRs	Baja	Media	Alta
>3 FTR	Media	Alta	Alta
Matriz de complejidad para EQs			
	1-5 DETs	6-19 DETs	>20 DETs
0-1 FTRs	Baja	Baja	Media
2-3 FTRs	Baja	Media	Alta
> 3 FTRs	Media	Alta	Alta

Tabla 7: *Matriz de complejidad para EIs, EOs y EQs*

De este modo, tenemos los siguientes datos para nuestra aplicación:

Nombre de la transacción	Tipo	FTRs	DETs	Complejidad
Iniciar aplicación	EI	0	3	Baja
Iniciar Sesión	EI	1	3	Baja
Registrar usuario	EI	1	7	Baja
Ocupar plaza	EI/EQ	4	4	Media
Compartir plaza	EI/EQ	4	4	Media
Ver perfil	EO	2	1	Baja
Editar perfil de usuario	EI/EQ	2	7	Media
Ver plazas libres	EO	2	6	Media
Ver plaza ocupada	EO	2	6	Media
Ver historial de plazas	EO	3	3	Media
Activar GPS	EI/EQ	1	3	Baja
Activar datos móviles	EI/EQ	1	3	Baja
Activar Wi-Fi	EI/EQ	1	3	Baja
Abrir navegador	EI/EQ	3	2	Media
Ver acerca de	EO	1	0	Baja
Ver información de la plaza	EI/EO	1	6	Baja
Ver información posición usuario	EI/EO	1	4	Baja

Tabla 8: *Estimación de la complejidad para EIs, EOs y EQs*

5.2.5. Cálculo del recuento bruto de puntos función

Una vez realizados los pasos anteriores, ya se puede proceder a calcular los puntos función. Para ello rellenaremos la siguiente tabla con los datos obtenidos en dichos pasos.

	Baja		Media		Alta		Total
EI	7	x3	4	X4	0	X6	37
EO	4	x4	3	X5	0	X7	31
EQ	3	x3	3	X4	0	X6	21
ILF	5	x7	0	X10	0	X15	35
EIF	3	x5	0	X7	0	X10	15
							139

Tabla 9: *Cálculo de los puntos función*

El resultado obtenido se conoce como *Unadjusted Function Points* (UFP).

5.2.6. Determinar el factor de ajuste

El VAF (*Value Adjustment Factor* / Valor del factor de ajuste) está basado en 14 GSCs (*General System Characteristics* / Características Generales del Sistema) que evalúan la funcionalidad de la aplicación. Cada característica tiene una descripción que ayuda a determinar el grado de influencia de dicha característica. Los grados de influencia van de 0 a 5 siendo:

- 0 : No influye
- 1 : Incidental
- 2 : Moderada
- 3 : Media
- 4 : Significativa
- 5 : Fuerte

Según esto, a la *aplicación para la gestión automática y eficiente del estacionamiento en las ciudades españolas*, le corresponden los siguientes GSCs:

Características	Grado de influencia
Comunicaciones de datos	5
Funciones distribuidas	1
Rendimiento	4
Configuraciones fuertemente utilizadas	1
Frecuencia transacciones	4
Entrada de datos on-line	5
Eficiencia del usuario final	3
Actualización on-line	3
Reutilización	3
Procesos complejos	3
Facilidad de instalación	3
Facilidad de operación	4
Instalación en distintos lugares	5
Facilidad de cambios	4
	48

Tabla 10: *Cálculo de GSCs*

La suma de la puntuación de cada característica da como resultado el grado total de influencia (TDI).

El VAF se calcula según la siguiente fórmula:

$$VAF = (TDI * 0,01) + 0,65$$

Por lo tanto, en nuestro caso resulta:

$$VAF = (48 * 0,01) + 0,65 = 1,13$$

5.2.7. Calcular el recuento ajustado

Por último, solo queda calcular los puntos función, pero esta vez teniendo en cuenta el VAF calculado en el apartado anterior. El resultado obtenido se conoce como *Adjusted Funtion Points* (AFP). Dicho cálculo queda del siguiente modo:

$$AFP = 139 * 1,13 = 157,07$$

5.2.8. Coste estimado de la aplicación

Suponiendo que la productividad es un punto función al día, obtenemos un resultado de 158 días para desarrollar el proyecto.

Asumiendo que el sueldo básico neto diario de un Ingeniero Técnico en la provincia de Teruel es de 48.02 [17], el coste estimado de la mano de obra es:

158 días * 48,02 €/d = **7.587,16 €**

A estos costes de mano de obra, hay que sumarles los costes de equipamiento utilizado para el desarrollo y las pruebas. Dicho equipamiento lo formar un ordenador portátil (800€) y un terminal Android, que en este caso es un Samsung Galaxy S3 (400€).

7.587,16 + 800 + 400 = **8787,16€**

5.2.9. Coste real de la aplicación

A continuación calcularemos el coste real de la aplicación para ver la diferencia existente con el coste estimado. Con los datos obtenidos del diagrama de Gantt (Planificación final) tenemos:

273 días * 48,02 €/día = 13.109,46€
13.109,46€ + 800 + 400 = **14.309,46€**

5.2.10. Conclusiones

Tras haber calculado los costes estimado y real, observamos que se ha producido una desviación de 38,59% debido al incremento del número de días empleados en el desarrollo del proyecto.

6. DISEÑO

Este apartado se centra en la fase de diseño de la aplicación. Dicho diseño debe de ser exhaustivo y claro, de manera que no se generen dudas o confusiones durante el proceso de implementación.

6.1.Base de datos

Para desarrollar la *Aplicación para la Gestión Automática y Eficiente del Estacionamiento en las ciudades españolas* es necesario diseñar e implementar una base de datos central, donde se almacene la información relativa a las plazas de aparcamiento y a los usuarios registrados. Además, este sistema dispondrá de una pequeña base de datos local en cada dispositivo donde se almacenarán datos relativos a los usuarios que se autentican en dicho dispositivo.

Realizar cualquier diseño de bases de datos, implica pasar por tres fases: diseño conceptual, diseño lógico y diseño físico. A continuación se van a desarrollar cada una de las fases, anteriormente citadas, en el contexto de nuestro proyecto. Primeramente se desarrollaran las tres fases anteriormente citadas para el caso de la aplicación servidor y, posteriormente, para la aplicación cliente:

6.1.1. Servidor

Diseño conceptual

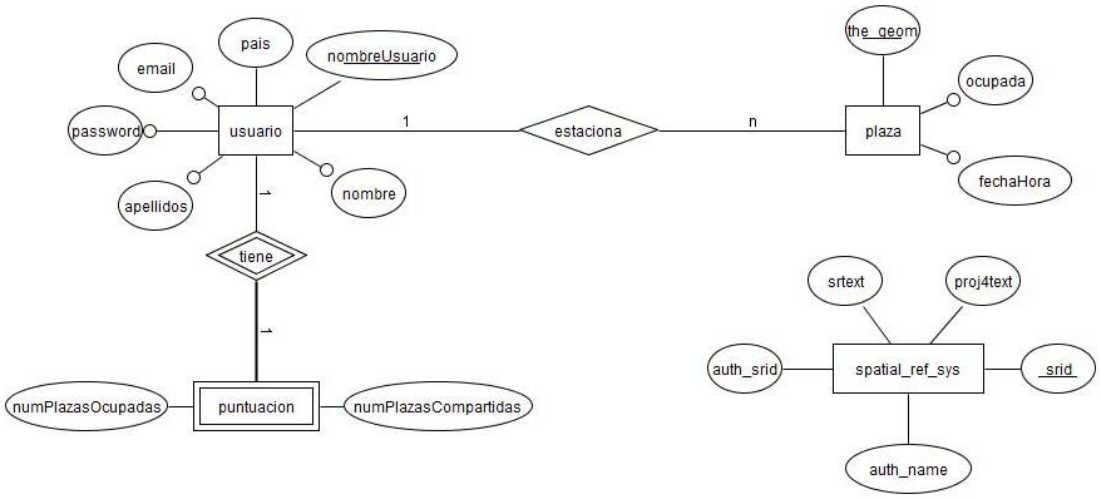


Figura 22: *Diagrama Entidad – Relación de la base de datos de la aplicación servidor*

A continuación, se explica más detalladamente el diagrama anterior:

- *plaza*: almacena la información relativa a una plaza que ha sido compartida u ocupada. Sus atributos son *the_geom* que almacena las coordenadas de la plaza, *fechaHora* que indica la fecha y hora de la última vez que ha cambiado el estado de la plaza (Ocupada/Libre) y *ocupada* que almacena dicho estado. Su clave primaria es *the_geom* y se relaciona con la entidad *usuario* mediante *Ocupa/Comparte*.

- *Usuario*: guarda toda la información referente los usuarios registrados en la aplicación. Tiene un identificador único (*nombreUsuario*). El resto de sus atributos son el *nombre*, *apellidos*, *password* (será la contraseña elegida para la cuenta de usuario) y *email*. Se relaciona con *puntuación* mediante *tiene.*

- *puntuación*: representa la puntuación asociada a un usuario concreto. Cada usuario sólo podrá tener una puntuación y está esta, tiene como atributos, *numPlazasCompartidas* y *numPlazasOcupadas* que son el número de plazas que ha compartido y ocupado el usuario.

- *spatial_ref_sys*: se trata de una entidad que habilita las funcionalidades GIS en nuestra base de datos. Incluye los sistemas de referencia espacial más comúnmente usados. Su identificador es el nombre del sistema espacial (*srid*), y ademas, tiene como atributos *srtext, pro4text, auth_srid* y *auth_name*.

Diseño lógico

A continuación se ha transformado el diagrama Entidad – Relación del apartado anterior en un esquema relacional. Dicho esquema relacional resultante es el siguiente:

```
plaza (the_geom integer, ocupada:integer,
fechaHora:integer)
      Clave Primaria { the_geom }
      Valor No Nulo { ocupada }
      Valor No Nulo { fechaHora }


usuario (nombreUsuario varchar, apellidos:varchar,
password:varchar, email:varchar, pais:varchar,
nombre:varchar )
      Clave Primaria { nombreUsuario }
      Valor No Nulo { apellidos }
      Valor No Nulo { password }
      Valor No Nulo { email }
      Valor No Nulo { nombre }
```

```
puntuacion (nombreUsuario varchar,
numPlazasCompartidas:integer, numPlazasOcupadas:integer )
      Clave Primaria { nombreUsuario }
       Clave Ajena { nombreUsuario } hace referencia a
       usuario


spatial_ref_sys (srid integer, auth_name:varchar,
auth_srid:integer, srtext:varchar, proj4text:varchar )
      Clave Primaria { srid }
```

Diseño físico

Para la implementación de la base de datos de ha utilizado PostgreSQL con el módulo PostGIS para bases de datos geográficas y pgAdmin III para facilitar la tarea de creación y administración de las tablas. Finalmente, la base de datos utilizada para la aplicación, consta de las siguientes 4 tablas:

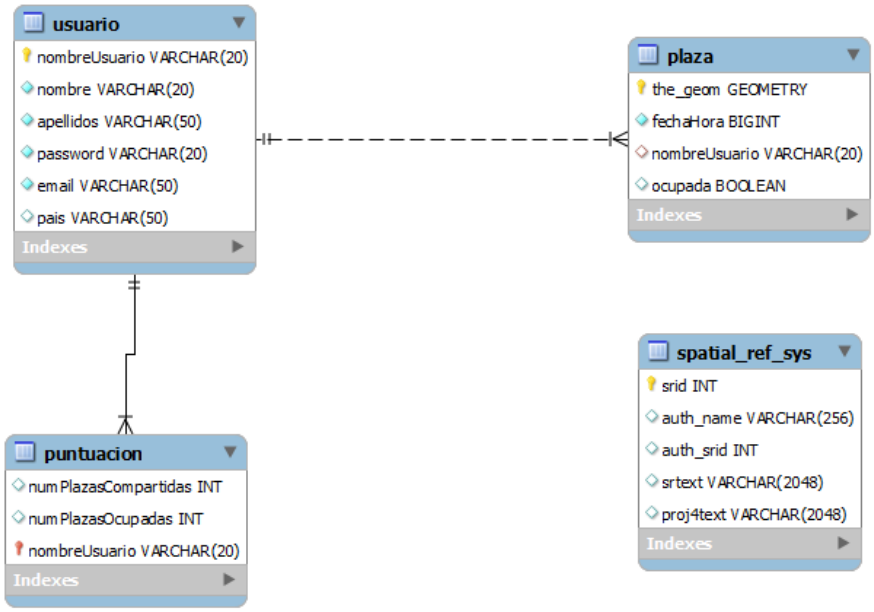


Figura 23: *Tablas de la base de datos de la aplicación servidor*

6.1.2. Cliente AparcaDroid

Diseño conceptual

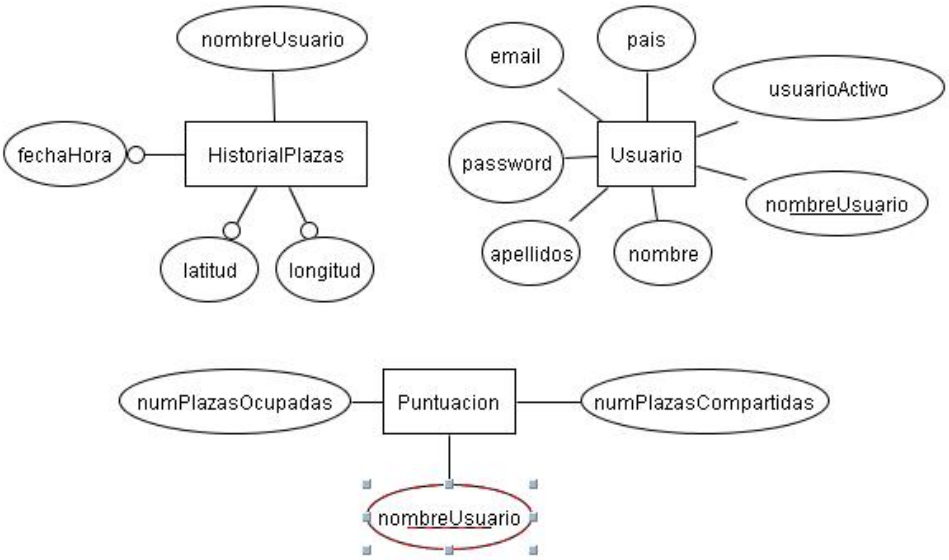


Figura 24: *Diagrama Entidad – Relación de la base de datos de la aplicación cliente*

A continuación, se comentarán los detalles y peculiaridades del diagrama anterior:

- *HistorialPlazas:* guarda la información necesaria de las plazas que el usuario ha ocupado y posteriormente liberado. Sus atributos son *latitud* y *longitud*, que almacenaran la localización del lugar, *fechaHora* almacenará la fecha y la hora en que la plaza fue liberada por última vez y un nombre de usuario que corresponderá con el usuario que se haya liberado la plaza.

- *Puntuación:* esta tabla almacenará la información de relativa al número de plazas que haya ocupado y compartido un usuario que se ha logueado en el terminal. Los atributos son *numPlazasOcupadas* y *numPlazasCompartidas* que corresponden al número de plazas que ha ocupado y compartido el usuario respectivamente y, *nombreUsuario* que almacenará al identificador del usuario relacionado con esa puntuación.

- *Usuario:* tabla que almacenará la información del perfil de usuario de todos los que se hayan autenticado en el terminal. Al igual que en la base de datos del servidor, tiene un identificador único (*nombreUsuario*). Además tiene los atributos *nombre*, *apellidos*, *password* (será la contraseña elegida para la cuenta de usuario) y *email*.

Diseño lógico

A continuación se ha transformado el diagrama Entidad – Relación del apartado anterior en el esquema relacional que se muestra a continuación:

```
HistorialPlazas (latitud:real, longitud:real,
fechaHora:bigint, nombreUsuario:text)
     Clave Primaria {  }
     Valor No Nulo { longitud }
     Valor No Nulo { fechaHora }
     Valor No Nulo { latitud }


Usuario (nombreUsuario text, nombre:text, apellidos:text,
password:text, email:varchar, pais:text,
usuarioActivo:numeric )
     Clave Primaria { nombreUsuario }

Puntuacion (nombreUsuario text, numPlazasCompartidas:integer,
numPlazasOcupadas:integer )
     Clave Primaria { nombreUsuario }
```

Diseño físico

Para la implementación de la base de datos de la aplicación cliente AparcaDroid, se ha utilizado SQLite. La creación y manejo de la base de datos se ha realizado mediante el código Java de la clase `DBHelper.java` y las clases de acceso creadas. A continuación se muestra las 3 tablas con las que cuenta esta base de datos:

Figura 25: *Tablas de la base de datos de la aplicación cliente*

6.2. Interfaz de usuario

La interfaz de usuario es un aspecto importante en la mayoría de aplicaciones informáticas. Además, en el caso de la aplicación que estamos desarrollando, adquiere especial importancia al necesitar una visualización clara y sencilla de las plazas, para facilitar y agilizar el estacionamiento. En este apartado, se desarrolla todos los aspectos relacionados con dicho interfaz.

6.2.1. Árbol de pantallas

En el diagrama de la siguiente figura, se muestra el árbol de pantallas de la presente aplicación. En él se muestra cada una de las diferentes pantallas, además de las diferentes transiciones y eventos que provocan cada una de dichas transiciones entre pantallas.

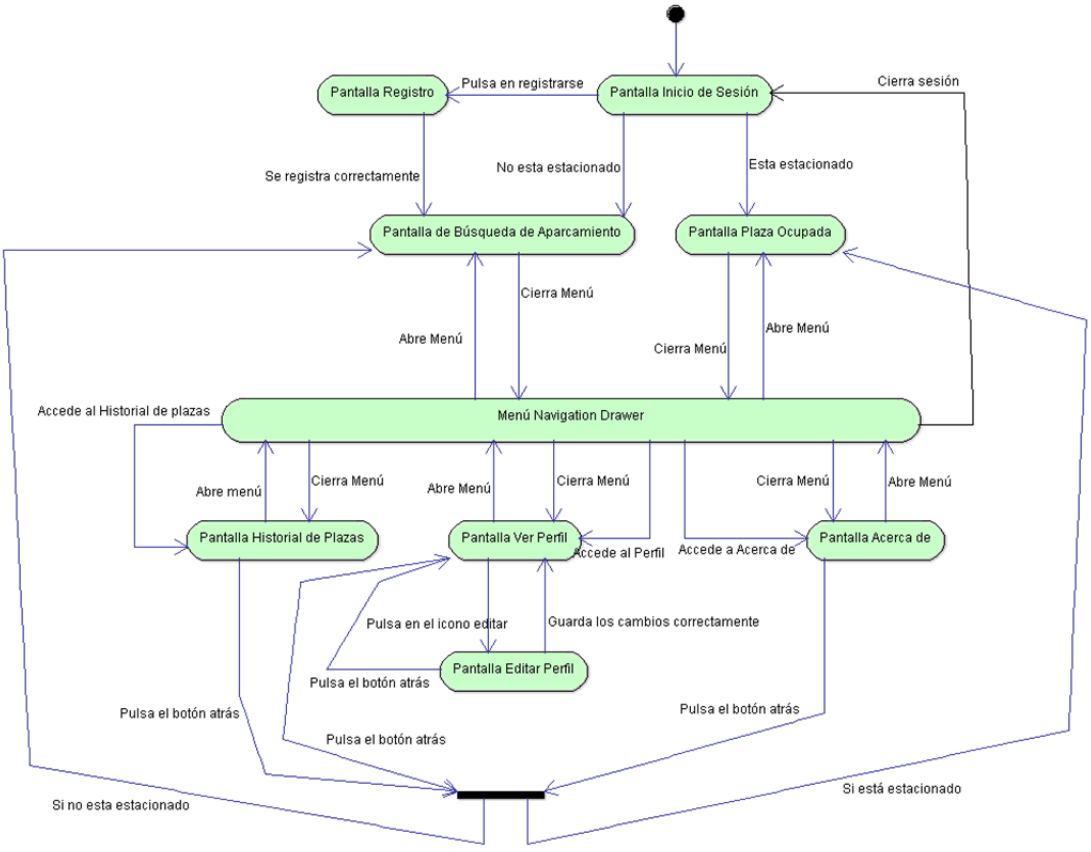


Figura 26: *Árbol de pantallas de la aplicación cliente*

Como se puede observar, cada ente en forma de burbuja, corresponde con una pantalla distinta de la aplicación cliente. La pantalla de entrada cuando abrimos la aplicación es la *Pantalla Inicio de Sesión*. Desde la *Pantalla Inicio de sesión*, también se puede acceder a la *Pantalla Registro* en la cual el usuario se puede registrar en el sistema. Tras haber iniciado sesión, dependiendo del si el usuario está estacionado o no, se mostrará la *Pantalla de Búsqueda de*

Aparcamiento o la *Pantalla de Plaza Ocupada*, respectivamente. Desde cualquiera de las dos anteriores se puede abrir el menú *Navigation Drawer* desde el cual podremos acceder tanto a las pantallas *Historial de Plazas*, *Ver Perfil* y *Acerca de*, como cerrar sesión. A su vez, desde la *Pantalla Ver Perfil* se puede acceder a la *Pantalla Editar Perfil* de modo que el usuario pueda editar su perfil. Debido a esta estructura, cuando pulsamos el botón atrás de nuestro terminal estando en las pantallas *Historial de Plazas*, *Ver Perfil*, *Editar Perfil*, *Acerca de*, dependiendo de si el usuario está estacionado o no volverá a una pantalla u otra. Si se encuentra estacionado mostrará la *Pantalla de Búsqueda de Aparcamiento*, y si no lo está, mostrará la *Pantalla de Plaza Ocupada*. Al lado de cada flecha se indica la acción o evento que provoca que la aplicación pase de una pantalla a otra. Estas acciones suelen ser la pulsación sobre un botón o elemento similar de la interfaz por parte del usuario.

6.2.2. Prototipo de interfaz

Previamente a la implementación de la interfaz gráfica de usuario (GUI, por sus siglas en inglés), se ha realizado los prototipos (*mockups*) de cada una de las pantallas que forman la aplicación. Esto se hace para tener una idea clara de cómo deben ser cada una de las pantallas así como sus funcionalidades. A continuación se muestran los *mockups* asociados con cada pantalla de la aplicación.

Pantalla de Inicio de Sesión

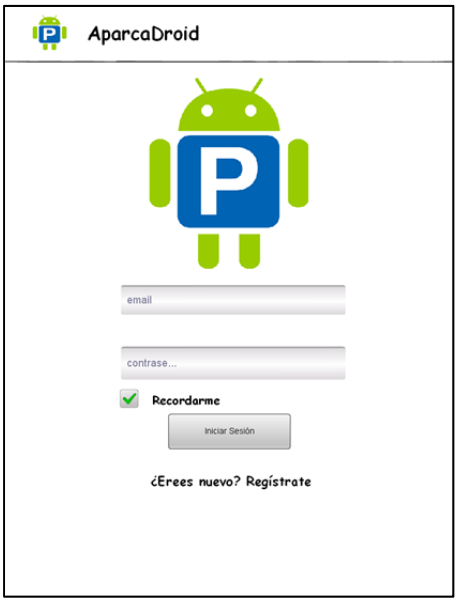


Figura 27: *Mockup pantalla Inicio de Sesión*

En este *mockup* se muestra el aspecto de la pantalla de inicio de sesión. Mostrará el nombre de la aplicación, así como un formulario para introducir

los datos necesarios para autenticarse en el sistema y un botón para acceder al registro de usuarios.

Pantalla de registro

Figura 28: *Mockup pantalla de registro*

En esta pantalla se posibilita el registro de nuevos usuarios en el sistema. Para ello existe un formulario con los campos necesarios para el registro; dichos campos, todos ellos necesarios son, nombre de usuario, nombre, apellidos, email y contraseña. Además, existe el botón *Terminar* para concluir dicho registro y, si los datos se validan correctamente, hacerlo efectivo.

Pantalla de Búsqueda de Aparcamiento

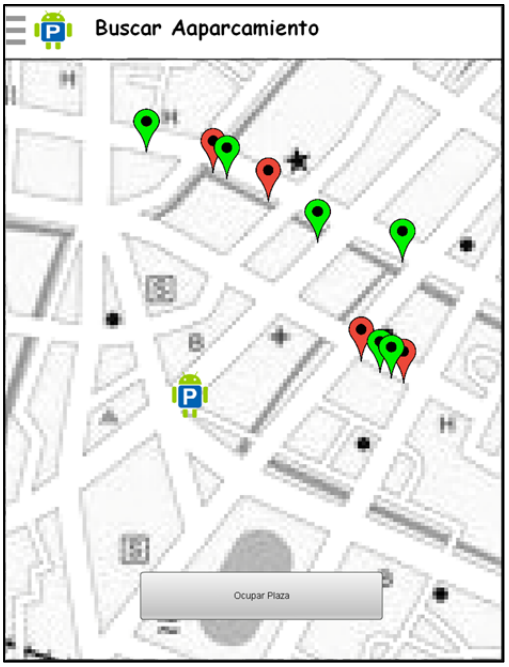


Figura 29: *Mockup pantalla búsqueda de aparcamiento*

Tras haberse autenticado en el sistema exitosamente, y en el caso de que el usuario no hay estacionado su vehículo previamente, se le mostrará esta pantalla. En ella podrá visualizar todas las plazas libres existentes a su alrededor, su posición, además de la información relativa a cada uno de los *markers* (plazas o posición de usuario) mostrados. Dicha información será, en el caso de las plazas, la dirección, distancia aproximada, usuario que la compartió y fecha en que lo hizo, asimismo, pulsando sobre la información, permite al usuario abrir el navegador para recibir indicaciones de cómo llegar. En el caso del *marker* que indica la posición de usuario mostrará el nombre de usuario, la dirección de donde se encuentra y la precisión del posicionamiento. Además existe un botón, que permite al usuario ocupar la plaza correspondiente a su posición.

Pantalla de Plaza Ocupada

Figura 30: *Mockup pantalla plaza ocupada*

La pantalla que muestra este *mockup* es similar a la del anterior apartado, con la diferencia que únicamente mostrará la posición actual del usuario y la de la plaza donde realizó su último estacionamiento. En cuanto a la información mostrada en el *marker* de la posición de usuario será la dirección, precisión, nombre de usuario y distancia aproximada; en el *marker* de la plaza, mostrará la dirección y, la fecha y hora en que se ocupó pudiendo, al igual que en el caso anterior, abrir el navegador tocando la información.

Pantalla Aparcamientos Recientes

Figura 31: *Mockup pantalla aparcamientos recientes*

En esta pantalla se muestra el *mockup* de la pantalla Aparcamientos recientes. Será similar a la *pantalla de búsqueda de aparcamientos*, pero en este caso las plazas se obtendrán de la base de datos local y serán las que haya ocupado y liberado recientemente el usuario.

Pantalla Ver perfil

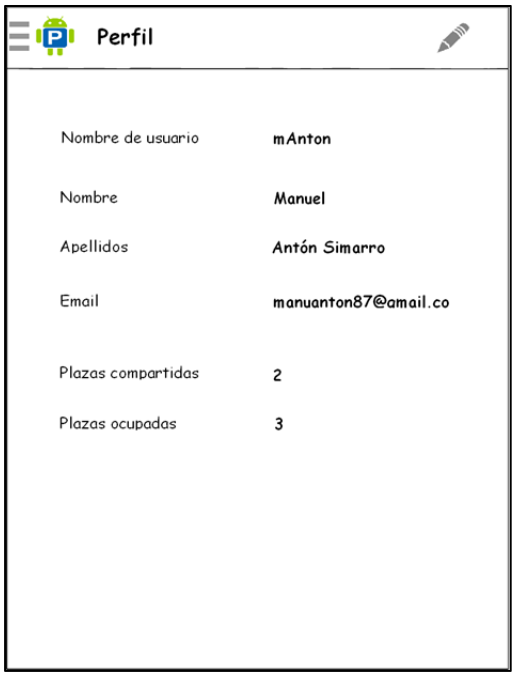


Figura 32: *Mockup pantalla ver perfil*

En esta pantalla se puede visualizar toda la información del usuario exceptuando la contraseña, que no se muestra por motivos de seguridad. Los datos que aparecen son nombre de usuario, nombre, apellidos, email y la puntuación del usuario, formada por el número de plazas compartidas y el número de plazas ocupadas. Además aparece un botón en *actionBar* que nos permite habilitar la edición del perfil.

Pantalla Editar Perfil

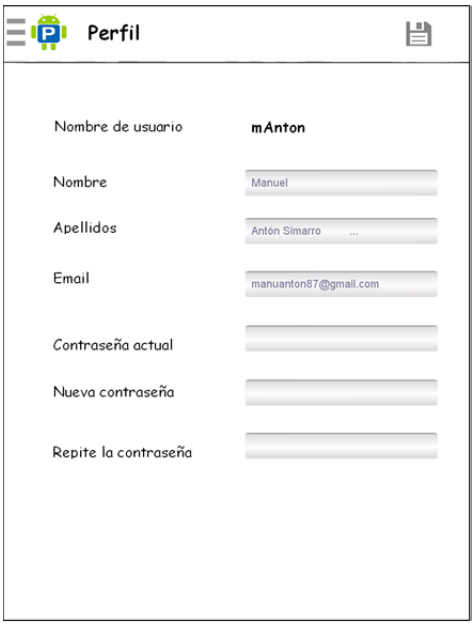


Figura 33: *Mockup pantalla editar perfil*

La actual pantalla es similar a la anterior, pero habilita los campos editables de texto para poder editar el perfil de usuario. Por ello se podrá editar el nombre, los apellidos, el email, e incluso, cambiar la contraseña. Para guardar los cambios existe un botón en el *actionBar* para guardar los cambios, tras comprobar la validez de los datos.

Pantalla Acerca De

Figura 34: *Mockup pantalla acerca de*

Como se puede apreciar en el mockup, esta pantalla tiene carácter meramente informativo. Solo mostrará el nombre de la aplicación, su logo, el autor y la versión en la que se encuentra.

Menú (Navigation Drawer)

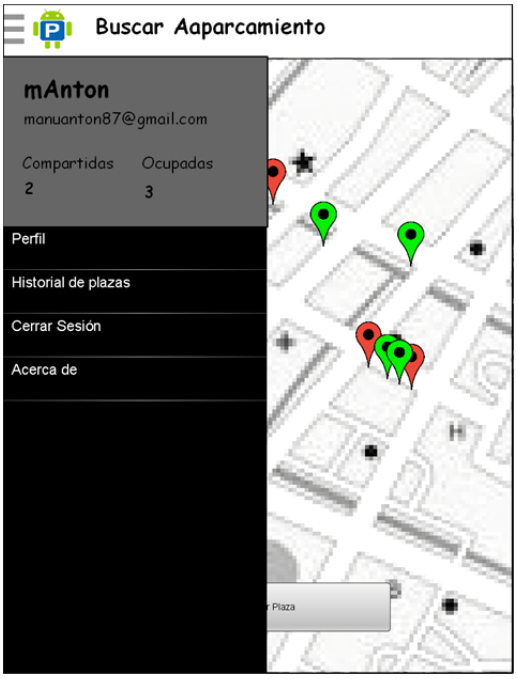


Figura 35: *Mockup menú de la aplicación*

Este elemento de interfaz es el menú principal de la aplicación. Está presente en todas las pantallas exceptuando la pantalla de inicio de sesión y la de registro, es decir, en todas tras haberse autenticado en el sistema. Se puede abrir y cerrar haciendo clic en la parte superior izquierda de la interfaz, más concretamente en el icono (*ic_drawer*) del *actionBar*, o sobre el nombre e icono de la aplicación de dicho *actionBar*. Este menú contiene una pequeño resumen del perfil de usuario y una lista con las secciones a las que puede acceder que son: Perfil, Historial de plazas y Acerca de, además de poder cerrar sesión.

6.3. Diagramas de clases

En este apartado se van a exponer los diagramas de clases de nuestro proyecto, tanto del servidor como del cliente. En primera instancia se expondrán las clases comunes al servidor y al cliente detallando aquellos atributos o métodos más destacables. Seguidamente, se expondrán las exclusivas del servidor y del cliente, iniciando ambos apartados con un diagrama general, que muestra todas las clases y sus relaciones. Tras estos diagramas generales, se comentarán con mayor profundidad las clases e interfaces más importantes, de forma individual, y centrándonos en aquellos métodos o atributos que tengan especial relevancia.

6.3.1. Comunes

Clase Coordenadas

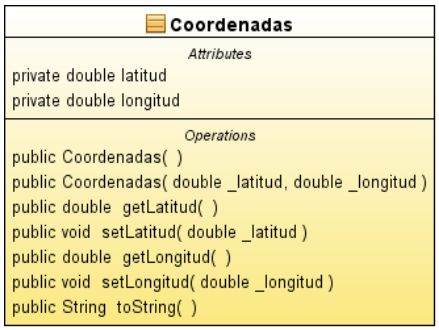


Figura 36: *Clase Coordenadas*

Esta clase permitirá manejar las coordenadas que representan la posición de los usuarios y de las plazas dentro de nuestra aplicación. Tiene como atributos:

- `private double longitud`: contendrá un valor de tipo `double` donde se indicará la longitud.
- `private double latitud`: contendrá un valor de tipo `double` donde se indicará la latitud.

Aparte de estos atributos y los constructores, la clase `Coordenadas` implementa todos los métodos `get` y `set` necesarios para modificar y obtener los valores de dichos atributos.

Clase Plaza

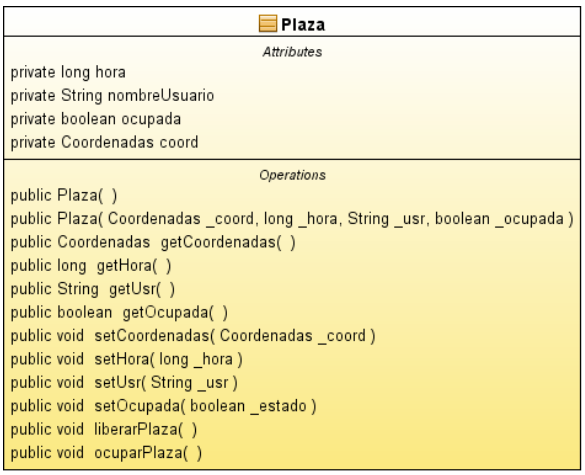


Figura 37: *Clase Plaza*

Se trata de una de las clases más importantes para el funcionamiento de nuestro sistema. Consta de los atributos y métodos necesarios para contener las características básicas de una plaza de aparcamiento así como el momento y la persona que la ocupó o liberó. Los atributos de esta clase son:

- `private long hora`: atributo que contiene la fecha y hora en que se ha ocupado o liberado la plaza. Dicha fecha se almacena en milisegundos transcurridos desde el 1 de Enero de 1970 utilizando el método `getTime` de la clase `Date`.

- `private String nombreUsuario`: esta cadena de texto contendrá el nombre de usuario del último usuario que haya compartido u ocupado la plaza.

- `private boolean ocupada`: atributo booleano que indica si la plaza está ocupada o no. Siendo true el valor que indica que está ocupada y false que está libre.

- `private Coordenadas coord`: atributo de tipo Coordenadas, explicado anteriormente, que determina la posición de la plaza.

A demás de los métodos `get` y `set` para obtener y modificar los atributos anteriormente citados, la clase `Plaza`, dispone de los siguientes métodos:

- `public void liberarPlaza()`: método que asigna al atributo `hora` la fecha y hora actual y establece el valor del atributo ocupada a false.

- `public void ocuparPlaza()`: método, que al igual que el anterior, asigna la fecha y hora actual al atributo `hora` y establece el atributo ocupada a true.

Clase Puntuación

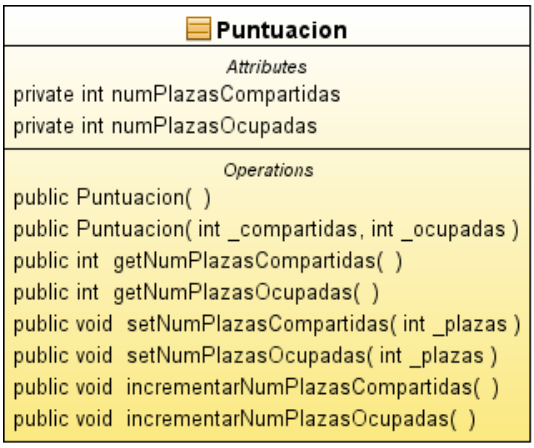


Figura 38: *Clase Puntuacion*

Se trata de una clase que nos permite almacenar la "puntuación" del usuario, es decir, el número de plazas que ha compartido y ocupado. Los atributos que forman la clase son:

- `private int numPlazasCompartidas`: atributo que almacena el número de plazas que ha compartido un usuario.

- `private int numPlazasOcupadas`: este atributo realiza una función similar al anterior, con la diferencia de que almacena el número de plazas ocupadas por el usuario.

Aparte de los métodos `get` y `set` para modificar los atributos anteriormente citados, esta clase dispone de los siguientes métodos destacables:

Clase Usuario

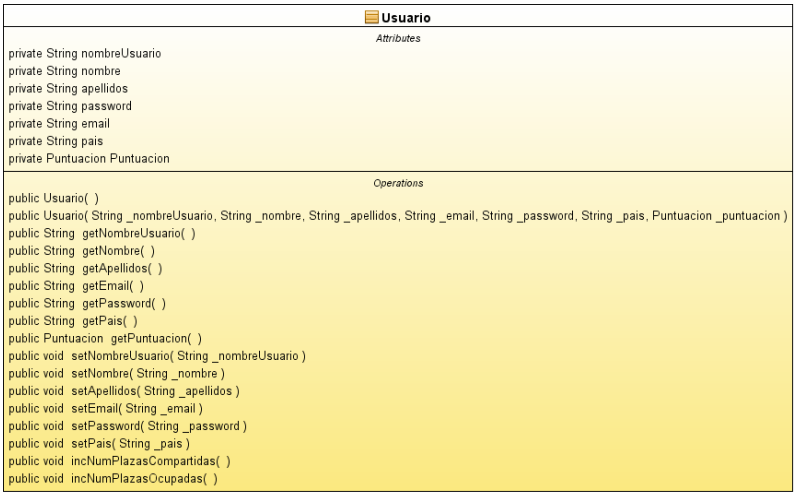


Figura 39: *Clase Usuario*

Esta clase permitirá manejar los usuarios de nuestra aplicación y tiene como atributos:

- `private String nombreUsuario`: este atributo, contendrá el nombre de usuario que será único en el sistema.
- `private String nombre`: contendrá el nombre de pila del usuario.
- `private String apellidos`: en este atributo almacenaremos los apellidos del usuario.
- `private String password`: este atributo será el encargado de contener la contraseña secreta de acceso al sistema que poseerá el usuario.
- `private String email`: en este atributo se almacena el email de contacto del usuario.
- `private String país`: se almacenará el país del usuario.
- `private Puntuación puntuación`: este atributo de tipo `Puntuacion` contiene la puntuación asociada al usuario. Esta clase ha sido explicada anteriormente.

Aparte de estos atributos y los constructores, la clase Usuario implementa todos los métodos `get` y `set` necesarios para modificar y obtener los valores de dichos atributos:

- `public void incNumPlazasCompartidas`: método que incrementa en 1 el número de plazas que el usuario ha compartido.

- `public void incNumPlazasOcupadas`: método que incrementa en 1 el número de plazas que el usuario ha ocupado.

Clase Protocolo

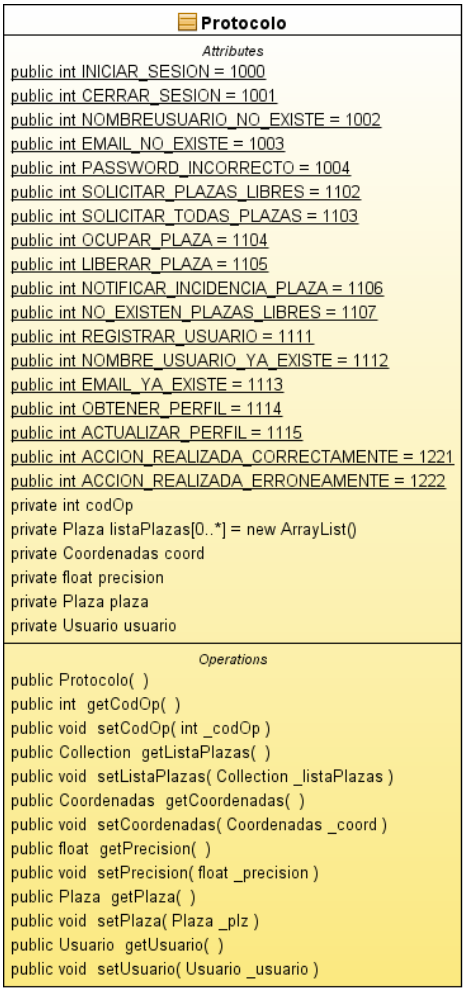


Figura 40: *Clase Protocolo*

Clase que representa el protocolo de comunicación diseñado para la Aplicación para la *Gestión Automática y Eficiente del Estacionamiento en las Ciudades españolas*. Contiene una serie de constantes de tipo entero que representan los códigos de las operaciones que se pueden realizar. Esta clase, además del constructor correspondiente, solamente dispone de métodos `get/set` para el acceso a los diferentes atributos de la clase, que son los siguientes:

- `private int codOp`: almacena el código de operación a realizar, representado por alguna de las constantes comentadas con anterioridad.

- `private Collection<Plaza> listaPlazas`: lista de plazas que se utiliza cuando la acción a realizar es obtener las plazas libres cerca del usuario.

- `private Coordenadas coord`: indica la posición del usuario cuando va a ocupar o liberar una plaza.

- `private float precisión`: precisión del sistema de posicionamiento a la hora de realizar alguna operación.

- `private Plaza plaza`: contiene la plaza que ha ocupado o libera el usuario cuando va a realizar alguna de estas dos acciones (ocupar o liberar una plaza).

:

- `private Usuario usuario`: este atributo sirve para indicar el usuario que realiza cierta petición.

Clase Crypter

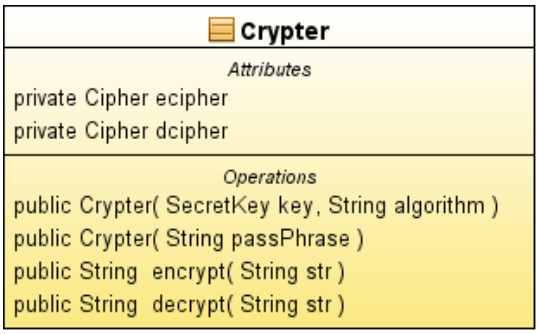


Figura 41: *Clase Crypter*

Se trata de una clase importante para la comunicación de datos, ya que, es la encargada de encriptar y desencriptar los datos que se van a enviar. Esta clase tiene los siguientes atributos:

- `private Cipher ecipher:` atributo que contendrá la clave o código de encriptado.

- `Private Cipher dcipher:` similar al anterior, este atributo contendrá la clave o código de desencriptado.

Además de los constructores de clase, existe uno en el que puedes indicar la clave de cifrado y el algoritmo y otro que solo indicas la clave, existen los siguientes métodos destacables:

- `Public String encrypt(String str):` método que recibe como parámetro una cadena de texto con que contendrá los datos a encriptar (en nuestro caso será un JSON). Devuelve encriptada la cadena de datos recibida.

- `Public String decrypt(String str):` Este método recibe como parámetro una cadena de texto encriptada y la devuelve desencriptada.

Interfaz InterfaceInterprete

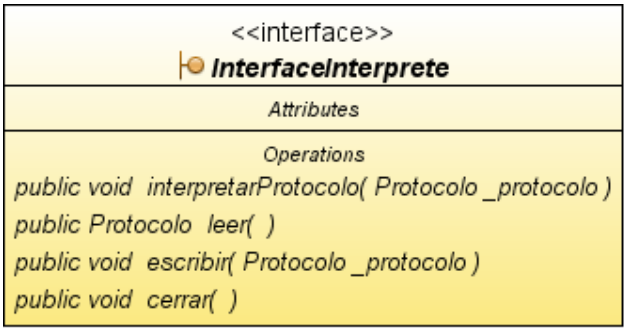


Figura 42: *Interface InterfaceInterprete*

Interfaz que define los métodos que deberá implementar la clase que se encargue de interpretar el protocolo de comunicación diseñado para el sistema de la *Aplicación para la Gestión Automática y Eficiente del Estacionamiento en las Ciudades españolas*. Los métodos que define esta interfaz son los siguientes:

- `public void interpretarProtocolo(Protocolo _protocolo):` éste método se encarga de tratar cada uno de los posibles códigos de operación que pueden aparecer en la instancia de la clase Protocolo que recibe como único argumento. Dependiendo del código de operación recibido llamará a un método `handleCase` u otro. (ver las clases `InterpreteServer` y `Protocolo` para más información).

- `public Protocolo leer():` este método se encarga de "leer" (*deserializar*) una instancia de la clase `Protocolo` del stream de entrada creado a partir del socket para comunicarse con la aplicación AparcaDroid.

- `public void escribir(Protocolo _protocolo):` "escribe" (serializa) una nueva instancia de la clase protocolo en el stream de salida creado a partir del socket para comunicarse con la aplicación AparcaDroid.

- `public void cerrar():` cierra en primer lugar los streams utilizados para leer y escribir en el socket, y a continuación el propio socket.

-

6.3.2. Servidor

A continuación se muestra el diagrama de clases completo de la aplicación servidor:

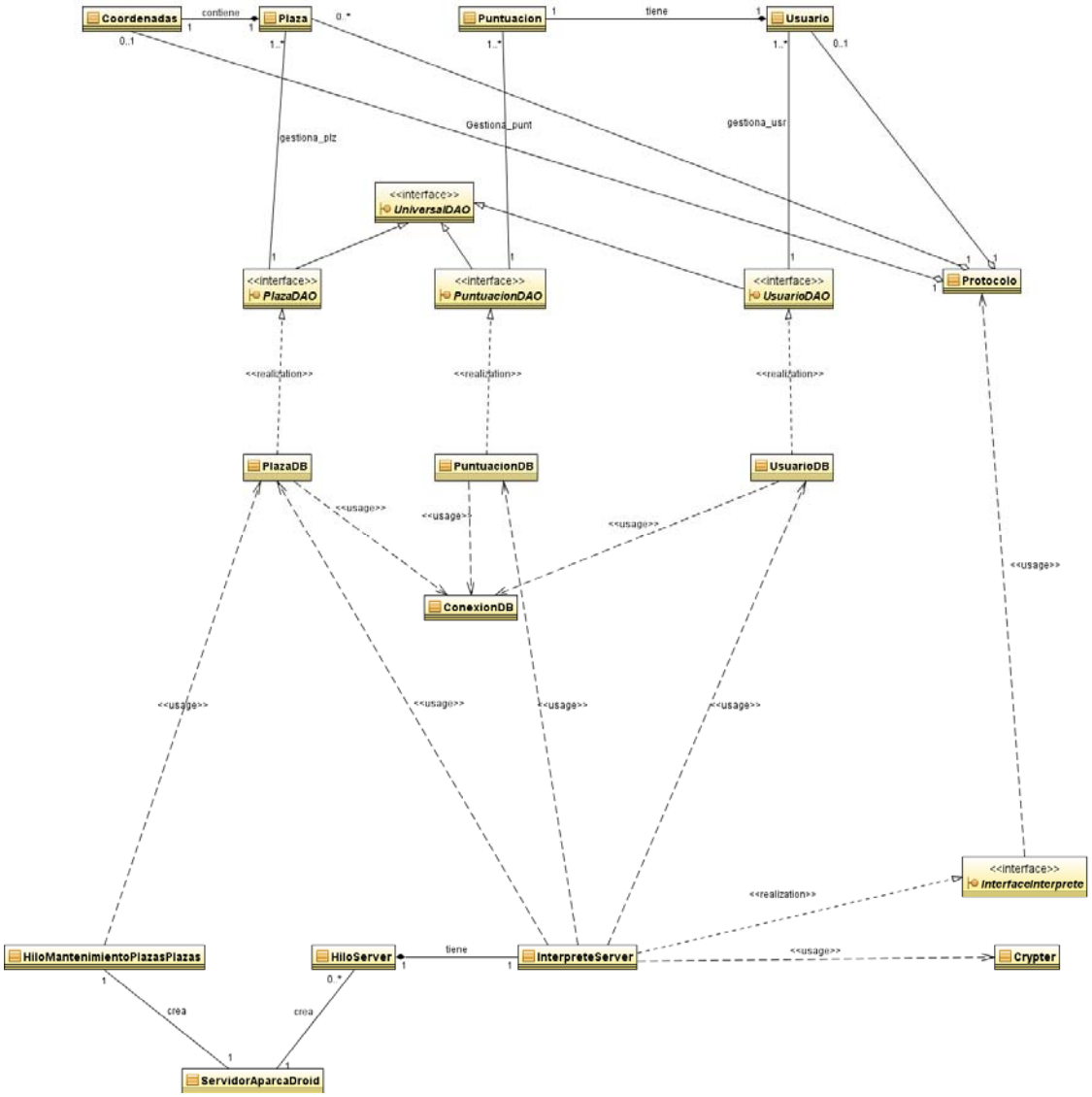


Figura 43: *Diagrama de clases completo aplicación servidor*

Interfaz UniversalDAO

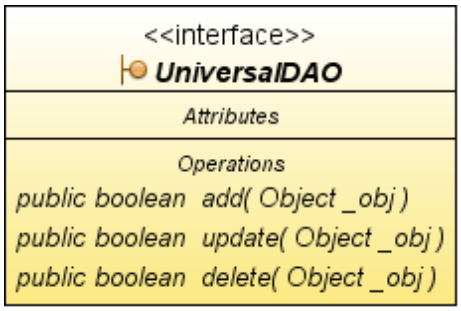


Figura 44: *Interface UniversalDAO*

`UniversalDAO` es la interfaz general a partir de la cual heredan todas las interfaces que controlan los métodos que permiten acceder al elemento encargado de la persistencia. Define los siguientes métodos:

- `public boolean añadir ( Object _object )`: Este método será a través del cual se insertarán elementos para su almacenaje.
- `public boolean borrar( Object _object )`: El método borrar como su nombre indica nos permitirá eliminar cualquier elemento de memoria secundaria.
- `public boolean actualizar( Object _object )`: La función de este método es la de modificar los elementos almacenados en memoria secundaria.

Interfaz PlazaDAO

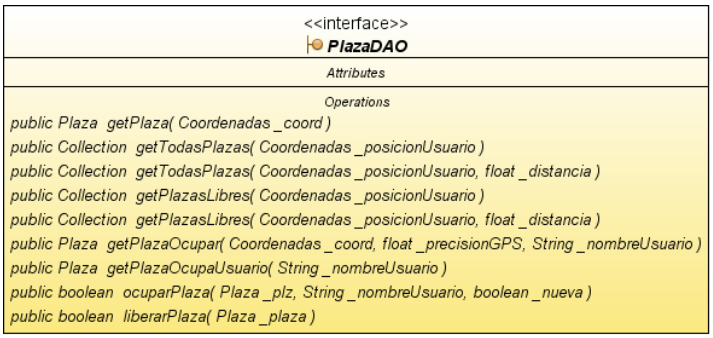


Figura 45: *Interfaz PlazaDAO*

`PlazaDAO` es la interfaz que define los métodos que deberán implementar las clases que se encarguen del manejo de las plazas que estarán almacenadas en la base de datos. Además de los métodos heredados de `UniversalDAO`, define los siguientes:

- `public Plaza getPlaza (Coordenadas _coord)`: Método que recibe como parámetro las coordenadas de una localización y devuelve la plaza correspondiente a dicha localización con toda su información.

- `public Collection getTodasPlazas (Coordenadas _posicionUsuario)`: Método que obtiene todas las plazas existentes (libres y ocupadas) a una distancia predefinida alrededor de la posición indicada como parámetro.

- `public Collection getTodasPlazas (Coordenadas _posicionUsuario, float _distancia)`: Al igual que el anterior, obtiene todas las plazas alrededor de la posición indicada, pero además, se le debe indicar la distancia máxima de las plazas que deseamos obtener.

- `public Collection getPlazasLibres (Coordenadas _posicionUsuario)`: Método que obtiene las plazas libres existentes a una distancia máxima predefinida.

- `public Collection getPlazasLibres (Coordenadas _posicionUsuario, float _distancia)`: Al igual que el anterior, obtiene las plazas libres alrededor de una posición indicada como parámetro, pero además, se le indica la distancia máxima que queremos obtener dichas plazas.

- `public Plaza getPlazaOcupar (Coordenadas _coord, float _precisionGPS, String _nombreUsuario)`: método que obtiene, a partir de las coordenadas indicadas, la plaza que se va a ocupar. Además del parámetro `_coord` de tipo Coordenadas también se utiliza la precisión del GPS (correspondiente al parámetro `float _precisionGPS`) para determinar qué plaza se ocupa.

- `public Plaza getPlazaOcupaUsuario (String _nombreUsuario)`: Método que obtiene la plaza que ocupa un usuario a partir del nombre de usuario.

- `public boolean ocuparPlaza (Plaza _plz, String _nombreUsuario, boolean _nueva)`: método que ocupa una plaza. Los parámetros necesarios son la propia plaza, el nombre de usuario y un booleano que indica si la plaza es nueva en la base de datos o no.

- `public boolean liberarPlaza (Plaza _plaza)`: este método libera la plaza que se le pasa como parámetro.

InterfazPuntuaciónDAO

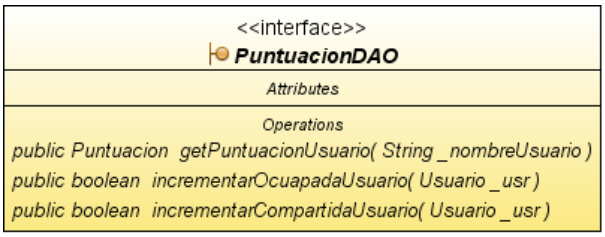


Figura 46: *Interface PuntuacionDAO*

Esta interfaz contiene la especificación de los métodos que deberá implementar la clase `PuntuacionDB` que será la encargada de manejar las puntuaciones de usuario en la base de datos. Dichos métodos son los siguientes:

- `public Puntuacion getPuntuacionUsuario (String _nombreUsuario)`: Método que obtiene la puntuación de un usuario determinado a partir de su nombre de usuario pasado como parámetro.
- `public boolean incrementarOcuapadaUsuario (Usuario _usr)`: método que, apartir del nombre de usuario, incrementa el número de plazas que ha ocupado éste usuario.
- `public boolean incrementarCompartidaUsuario (Usuario _usr)`: similar al método anterior, incrementa el número de plazas compartidas por un usuario indicando su nombre de usuario como parámetro.

Interfaz UsuarioDAO

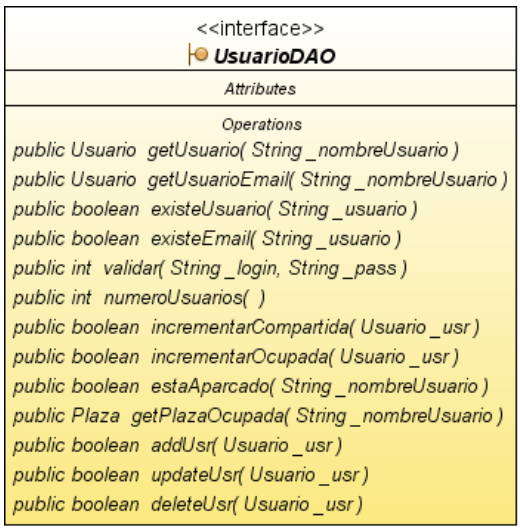


Figura 47: *Interface UsuarioDAO*

`UsuarioDAO` es la interfaz que define los métodos que deberá implementar la clase que se encargue del manejo de usuarios (en nuestro caso `UsuarioDB`). Dichos métodos son los siguientes:

- `public Usuario getUsuario (String _nombreUsuario)`: método que permite obtener un usuario y toda su información asociada a partir del nombre de usuario pasado como parámetro.

- `public Usuario getUsuarioEmail(String _nombreUsuario)`: similar al método anterior, permite obtener un usuario y toda su información.

- `public boolean existeUsuario (String _usuario)`: método que devuelve true si existe el nombre de usuario indicado como parámetro y false si no existe

- `public boolean existeEmail (String _usuario)`: similar al anterior, devuelve true si existe en la base de datos el email indicado como parámetro, y en caso contrario, devuelve false.

- `public int validar (String _login, String _pass)`: método que comprueba la existencia del nombre de usuario indicado como parámetro (`String _login`) y que la contraseña concuerda con la almacenada para ese usuario concuerda con la que se ha pasado como parámetro (`String _pass`). Del mismo modo devolvera un código dependiendo del resultado de la operación.

- `public int numeroUsuarios()`: método que devuelve el número de usuarios total del sistema.

- `public boolean incrementarCompartida (Usuario _usr)`: método que realiza las acciones necesarias para incrementar el número de plazas compartidas de un usuario indicado como parámetro.

- `public boolean incrementarOcupada(Usuario _usr)`: similar al anterior, en este caso, realiza las operaciones necesarias para incrementar el número de plazas que ha ocupado un usuario que se pasa como parámetro.

- `public boolean estaAparcado (String _nombreUsuario)`: método que devuelve true si el usuario del cual se pasa como parámetro su nombre de usuario, esta aparcado y false si no lo está.

- `public Plaza getPlazaOcupada (String _nombreUsuario)`: método que devuelve la plaza que ocupa un usuario indicado como parámetro, en caso de que este estacionado.

- `public boolean addUsr(Usuario _usr)`: método que realiza las llamadas necesarias para añadir un usuario a la base de datos y su puntuación asociada.

- `public boolean updateUsr(Usuario _usr)`: método que realiza las acciones necesarias para actualizar la información de un usuario, tanto datos personales como puntuación. Dicho usuario se pasará como parámetro.

- `public boolean deleteUsr (Usuario _usr)`: método que elimina toda la información referente a un usuario indicado como parámetro. Elimina tanto la información personal como la puntuación asociada.

Clase ConexionDB

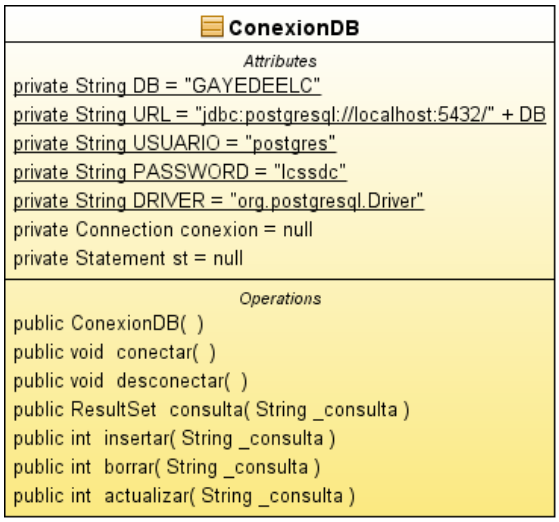


Figura 48: *Clase ConexionDB*

La clase `ConexiónDB` contiene los parámetros y métodos necesarios que permiten el acceso y gestión de la información de la base de datos. Los atributos que forman parte de esta clase son:

- `private static final String DB`: Contiene el nombre de la base de datos.

- `private static final String USUARIO`: Contiene la información del usuario que nos permite acceder a la base de datos.

- `private static final String PASSWORD`: Contiene la información de la contraseña usada para acceder a la base de datos.

- `private static final String URL`: Contiene la URL de la base de datos de nuestra aplicación.

- `private static final String DRIVER`: Contiene el nombre del driver que nos permite conectar con la base de datos.

Los métodos que implementa la clase son los siguientes:

- `public ConexionDB ()`: Constructor de la clase.
- `public void conectar ()`: Método que conecta con la base de datos.
- `public void desconectar ()`: Método que desconecta de la base de datos.
- `public ResultSet consulta (String _consulta)`: Método a través del cual se realizarán consultas a la base de datos.
- `public int actualizar (String _consulta)`: Método que permitirá añadir, borrar y modificar elementos de la base de datos, devuelve el número de campos modificados.

Clase PlazaDB

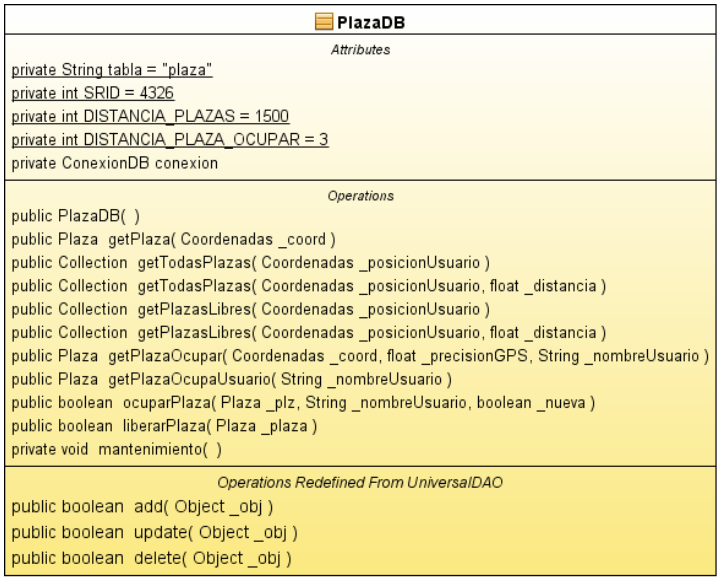


Figura 49: *Clase PlazaDB*

La clase `PlazaDB` implementa los metodos de la clase `PlazaDAO` y de la interfaz `UniversalDAO`, explicados con anterioridad en este documento. Además implementa el método `private void mantenimiento()` que modifica el estado de las plazas que lleven libres más de un tiempo determinado, pasado a estar ocupadas.

Clase PuntuacionDB

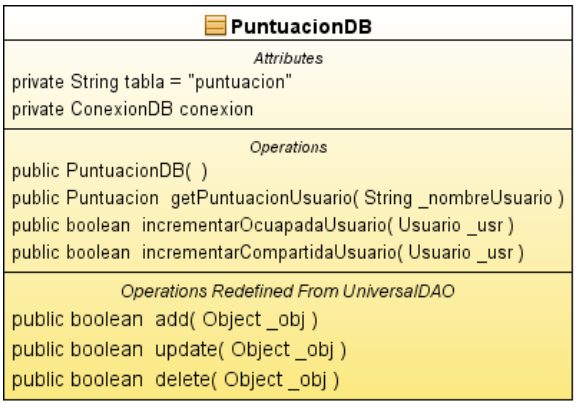


Figura 50: *Clase PuntuacionDB*

La clase `PuntuacionDB` implementa los métodos de la interfaz `PuntuacionDAO` y de la interfaz `UniversalDAO`, que ya han sido explicados anteriormente en este documento. Aparte de estos métodos, no implementa ninguno propio adicional.

Clase UsuarioDB

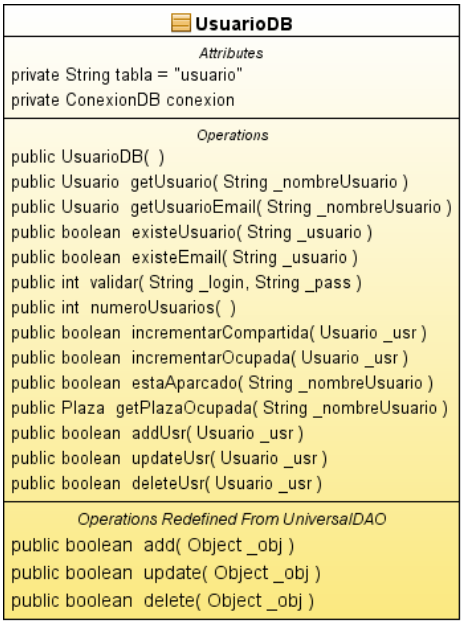


Figura 51: *Clase UsuarioDB*

La clase `UsuarioDB` implementa los métodos definidos en las interfaces `UsuarioDAO` y `UniversalDAO`, que ya han sido explicados anteriormente. Ademas, no implementa métodos adicionales propios.

Clase ServidorAparcaDroid

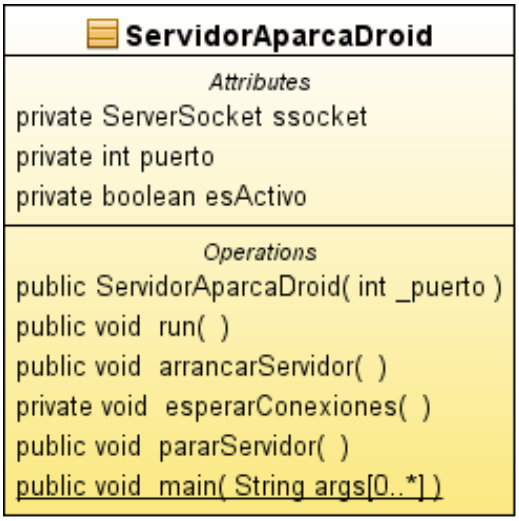


Figura 52: *Clase ServidorAparcaDroid*

Clase que representa el servidor del sistema, se encarga de aceptar y tratar las peticiones realizadas por los clientes. Tiene como atributos el número de puerto en el que está escuchando peticiones y un booleano para indicar si el servidor está activo, además del socket. Sus métodos son los siguientes:

- `public ServidorAparcaDroid(int _puerto)`: constructor de la clase. Recibe un entero como parámetro, que será el puerto en el que el servidor estará escuchando las peticiones de los clientes.
- `public void run()`: método que se usa en el caso de que queramos lanzar el servidor en nuevo hilo de ejecución.
- `public void arrancarServidor()`: método que inicializa el servidor.
- `public void esperarConexiones()`: este método de la clase `ServidorAparcaDroid` es el encargado de aceptar las peticiones de los clientes. Se ejecuta constantemente mientras el servidor este activo.
- `public void pararServidor()`: método que se encarga de detener el servidor, es decir, cierra el socket y deja de atender peticiones de clientes.
- `public void main(String[] args)`: este método es el punto de entrada de la aplicación servidor.

Clase HiloServer

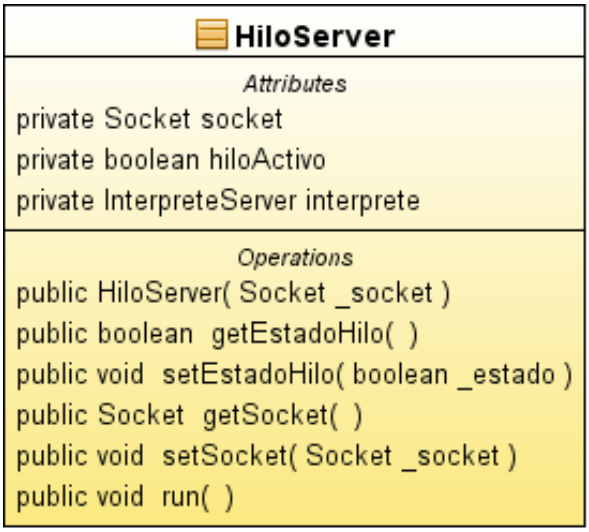


Figura 53: *Clase HiloServer*

Se creará una instancia de la clase `HiloServer` para cada una de las peticiones de conexión aceptadas por la clase `ServidorAparcaDroid`. Esta clase, que se ejecuta en su propio *thread*, es la que realmente se encargará de atender cada una de dichas peticiones. Los métodos que cabe destacar en esta clase son los siguientes:

- `public HiloServer(Socket _socket)`: constructor de la clase. Recibe una instancia del socket como parámetro.
- `public boolean getEstadoHilo()`: devuelve cierto en el caso en el que el hilo este activo, falso en caso contrario.
- `public void setEstadoHilo(boolean _estado)`: permite modificar el estado del atributo `hiloActivo`.
- `public void run()`: método que se ejecuta en un *thread* diferente y que interpretará el contenido del protocolo.

Clase HiloMantenimientoPlazas

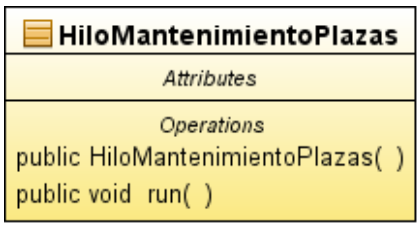


Figura 54: *Clase HiloMantenimientoPlazas*

Esta clase es la encargada de realizar un mantenimiento periódico de las plazas registradas en el sistema. Se ejecuta en su propio *thread*, creándose la instancia al arrancar el servidor, es decir, al ejecutar la clase `ServidorAparcaDroid`. Dispone del constructor de clase y del método `run` únicamente.

Clase InterpreteServer

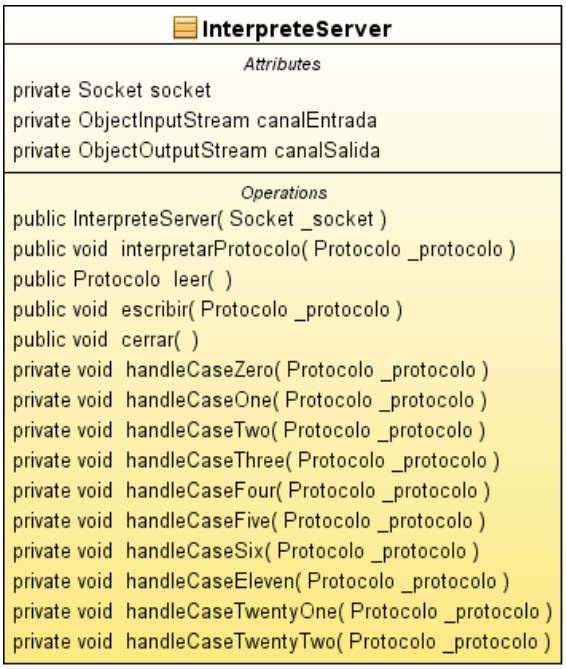


Figura 55: *Clase InterpreteServer*

Clase que implementa la interfaz `InterfazInterprete`. Tiene como atributos el socket correspondiente y un `ObjectInputStream` y un `ObjectOutputStream` que se usan para leer y escribir del *socket* respectivamente. Como método dispone del constructor de las que recibe como parámetro el socket del cliente del cual se van a atender peticiones. Además del constructor, anteriormente citado, existe un método `handleCase` para cada uno de los códigos de operación que aparecen en el protocolo. En cada uno de estos métodos se llevarán a cabo las acciones pertinentes correspondientes a la operación a realizar.

6.3.3. Cliente

El siguiente diagrama es el diagrama de clases completo de la aplicación cliente:

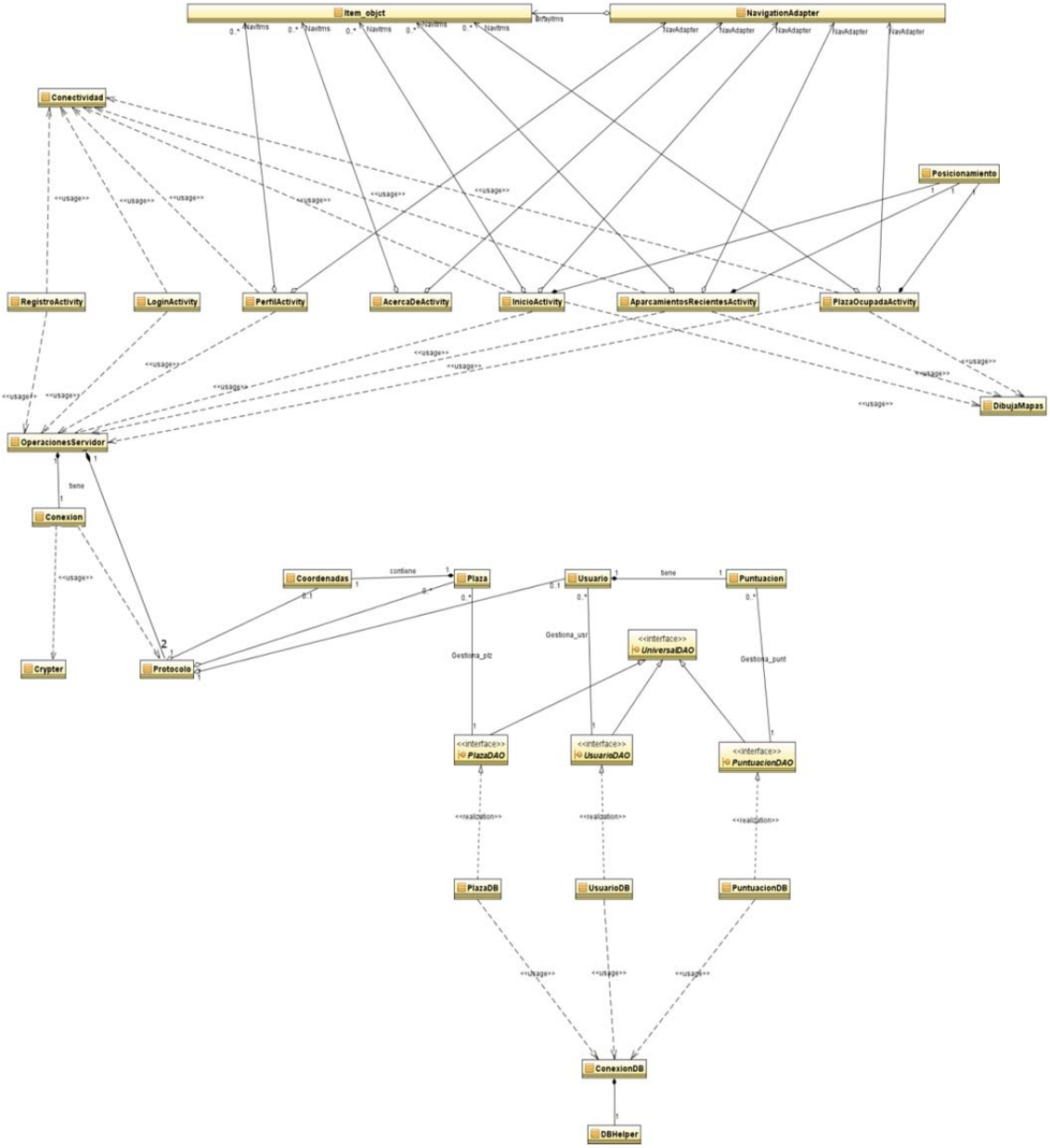


Figura 56: *Diagrama de clases completo aplicación servidor*

UniversalDAO

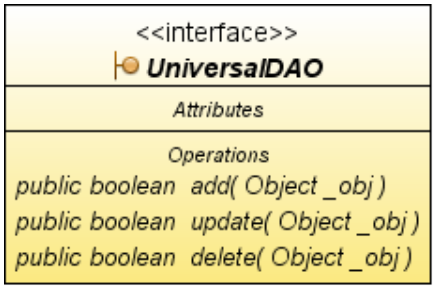


Figura 57: *Interface UniversalDAO*

Es la interfaz que heredan todas las interfaces que controlan los métodos que permiten tener acceso al elemento encargado de la persistencia. Define los siguientes métodos:

- `public boolean anyadir ( Object _object ):` Este método será a través del cual se insertarán elementos para su almacenaje.
- `public boolean borrar( Object _object ):` nos permitirá eliminar cualquier elemento de memoria secundaria.
- `public boolean actualizar( Object _object ):` La función de este método es la de modificar los elementos almacenados en memoria secundaria.

PlazaDAO

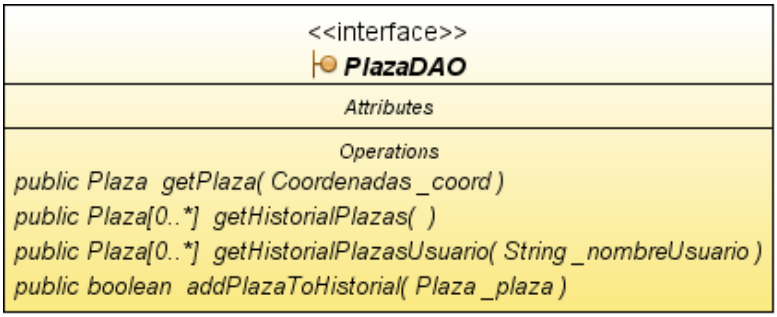


Figura 58: *Interface PlazaDAO*

`PlazaDAO` es el interfaz que define los métodos que deberán definir las clases encargadas del manejo de plazas en la base de datos. Los métodos que define `PlazaDAO` son los siguientes:

- `public Plaza getPlaza (Coordenadas _coord):` método que a partir de unas coordenadas que se le pasan como parametro, obtiene la plaaza correspondiente a esa localización.

- `public List<Plaza> getHistorialPlazas()`: método que retorna una lista con todas las plazas registradas en el terminal.

- `public List<Plaza> getHistorialPlazasUsuario(String _nombreUsuario)`: método que, al igual que el anterior, devuelve una lista con las plazas pero, en este caso, que ha registrado un determinado usuario.

- `public boolean addPlazaToHistorial (Plaza _plaza)`: método al que se le pasa una plaza como atrigbuto y la añade al historial del terminal.

PuntuacionDAO

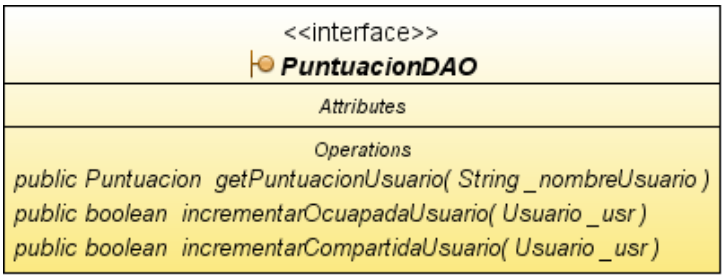


Figura 59: *Interface PuntuacionDAO*

`PuntuacionDAO` es el interfaz encargado de definir los métodos necesarios para el manejo de las puntuaciones de usuario en la base de datos del cliente. Dichos métodos son:

- `public Puntuacion getPuntuacionUsuario (String _nombreUsuario)`: método que a partir del nombre de usuario pasado como parametro, obtiene la puntuación del usuario indicado.

- `public boolean incrementarOcuapadaUsuario (Usuario _usr)`: método que incrementa el número de plazas ocupadas del usuario indicado como parámetro.

- `public boolean incrementarCompartidaUsuario (Usuario _usr)`: este método incrementa en uno el número de plazas compartidas por el usuario que se pasa como parámetro.

UsuarioDAO

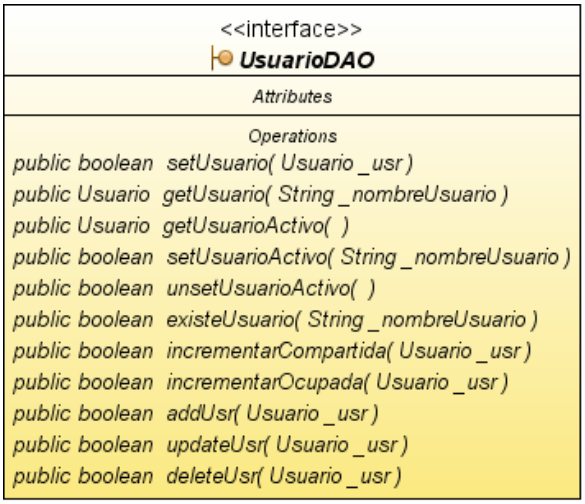


Figura 60: *Interface UsuarioDAO*

El interfaz UsuarioDAO define los métodos que deberán implementar las clases que se encarguen de la gestión de los usuarios en la base de datos de la aplicación cliente. Dichos métodos del interfaz son:

- `public Usuario getUsuario(String _nombreUsuario)`: método que a partir del nombre de usuario pasado como parámetro obtiene el usuario con todos los datos almacenados en la base de datos almacenados en el terminal.

- `public Usuario getUsuarioActivo()`: método que obtiene todos los datos del usuario que en ese momento se encuentra logueado en el sistema.

- `public boolean setUsuarioActivo(String _nombreUsuario)`: método que establece, al usuario indicado como parámetro, como usuario activo.

- `public boolean unsetUsuarioActivo()`: método que establece como inactivo al usuario que en ese momento este activo.

- `public boolean existeUsuario(String _nombreUsuario)`: método que indica si un usuario indicado como parámetro existe ya en la base de datos o no.

- `public boolean incrementarCompartida (Usuario _usr)`: método que incrementa el número de plazas compartidas de un usuario que se pasa como parámetro.

- `public boolean incrementarOcupada(Usuario _usr)`:método similar al anterior, pero en este caso, incremeneta el número de

plazas que ocupado el usuario. Dicho usuario se le pasa como parámetro.

- `public boolean addUsr(Usuario _usr)`: método que añade un usuario, que se pasa como parámetro, a la base de datos de la aplicación cliente.

- `public boolean updateUsr(Usuario _usr)`: este método actualiza los datos almacenados en la basee de datos de la aplicación cliente, del usuario indicado como parámetro.

- `public boolean deleteUsr (Usuario _usr)`: elimina de la base de datos de la aplicación cliente al usuario indicado como parámetro.

Clase PlazaDB

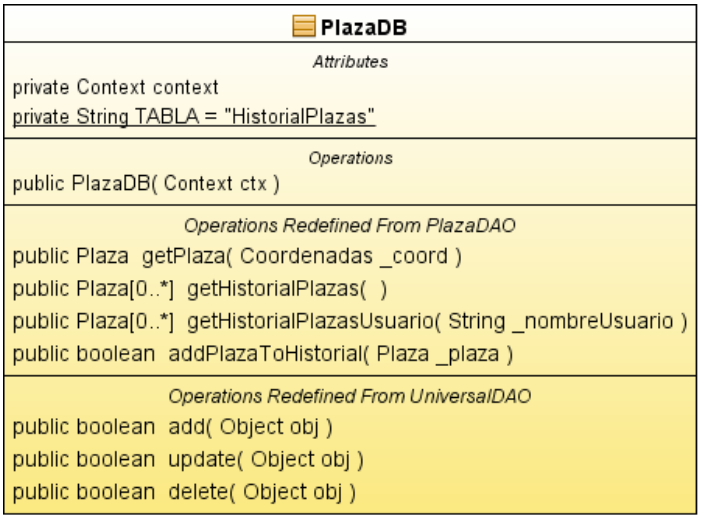


Figura 61: *Clase PlazaDB*

La clase `PlazaDB` implementa los métodos definidos en los interfaces `PlazaDAO` y `UniversalDAO`, explicados con anterioridad en el presente documento.

Clase PuntuacionDB

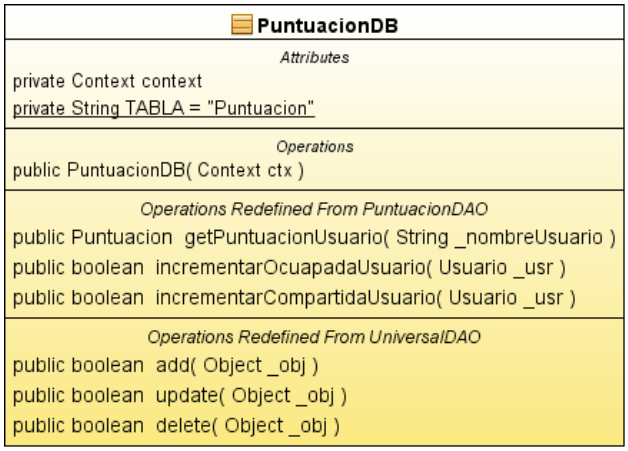


Figura 62:: *Clase PuntuacionDB*

La clase `PuntuacionDB` implementa los métodos de la interfaz `PuntuacionDAO` y de la interfaz `UniversalDAO`, explicados anteriormente. Aparte de estos métodos, no implementa ninguno propio adicional.

Clase UsuarioDB

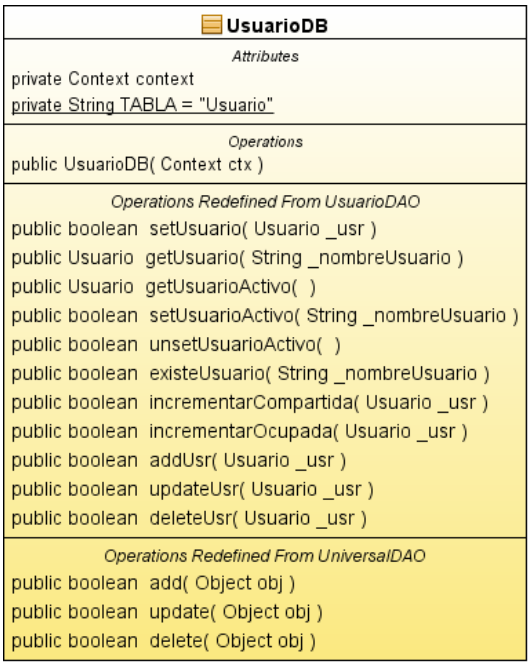


Figura 63: *Clase UsuarioDB*

La clase `UsuarioDB` implementa los métodos definidos en las interfaces `UsuarioDAO` y `UniversalDAO`, ya expuestos anteriormente. Además, no implementa métodos adicionales propios.

Clase ConexionDB

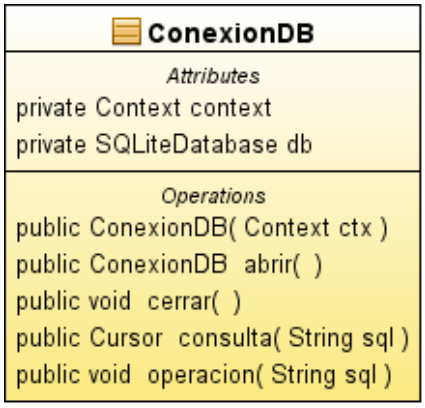


Figura 64: *Clase ConexionDB*

Esta clase tiene los parámetros y métodos necesarios que permitirán el acceso y gestión de la información almacenada en la base de datos de la aplicación AparcaDroid cliente. Los atributos que forman la clase son:

- `private final Context context`: constante que contendrá el contexto de la aplicación necesario para instanciar el DBHelper.
- `private DBHelper dbHelper`: contendrá la instancia de DBHelper necesaria para interactuar con la base de datos.
- `private SQLiteDatabase db`: variable que contendrá la instancia de la base de datos que permite el acceso a ella.

Además de los atributos anteriores, dispone de los métodos de los que dispone son los siguientes:

- `public ConexionDB (Context ctx)`: Constructor de la clase.
- `public ConexionDB abrir()`:Método que conecta con la base de datos.
- `public void cerrar ()`: Método que cierra la conexión con la base de datos.
- `public Cursor consulta( String sql)`: Método a través del cual se realizarán consultas a la base de datos.
- `public void operacion( String sql)`: Método que permitirá añadir, borrar y modificar elementos de la base de datos.

Clase DBHelper

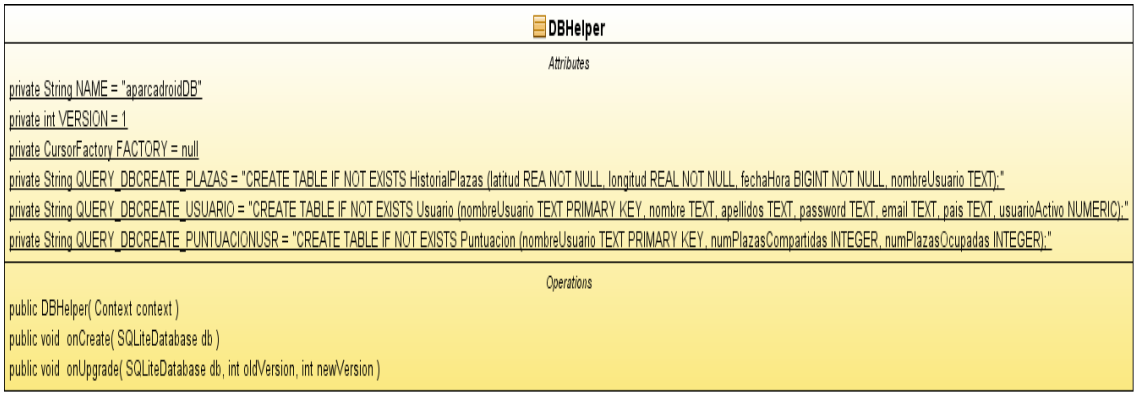


Figura 65: *Clase DBHelper*

La clase `DBHelper` extiende de la clase `SQLiteOpenHelper` y se utiliza para el manejo y creación de la base de datos, además de realizar el control de versiones de dicha base de datos. Aparte del constructor de clase, dispone de los siguientes métodos que se sobrescriben de la clase que extienden (`SQLiteOpenHelper`) y son:

- `public void onCreate(SQLiteDatabase db)`: método que crea la base de datos en el caso de que no exista ya. Recibe como parámetro un objeto de la clase `SQLiteDatabase`, que es sobre el cual se creará la base de datos.

- `public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`: es llamado cuando la base de datos necesita ser actualizada, concretamente eliminara la base de datos existente, y creara una nueva desde cero. El control de las versiones de la base de datos se lleva a cabo con los parámetros de entrada.

Esta clase, además de los métodos anteriormente citados dispone de las siguientes variables constantes, que sirven como parámetros de entrada para la ceración de la base de datos y son:

- `private static final String NAME = "aparcadroidDB"`: atributo que indica el nombre de la base de datos de la aplicación cliente.

- `private static final int VERSION = 1`: atributo que indica el número de versión de la base de datos.

- `private static final String QUERY_DBCREATE_PLAZAS = "CREATE TABLE IF NOT EXISTS HistorialPlazas (latitud REA NOT NULL, longitud REAL NOT NULL, fechaHora BIGINT NOT NULL, nombreUsuario TEXT);"`: constante string que sirve como

parametro de entrada y que contiene la consulta para crear la tabla contenedora del historial de plazas.

- `private static final String QUERY_DBCREATE_USUARIO = "CREATE TABLE IF NOT EXISTS Usuario (nombreUsuario TEXT PRIMARY KEY, nombre TEXT, apellidos TEXT, password TEXT, email TEXT, pais TEXT, usuarioActivo NUMERIC);":` este atributo es una constante de tipo string que contiene la consulta que servirá para crear la tabla Usuario.

- `private static final String QUERY_DBCREATE_PUNTUACIONUSR = "CREATE TABLE IF NOT EXISTS Puntuacion (nombreUsuario TEXT PRIMARY KEY, numPlazasCompartidas INTEGER, numPlazasOcupadas INTEGER);":` es una constante String que almacena la consulta que creará la tabla Puntuacion encargada de almacenar las puntuaciones de los usuarios que se han logueado en el sistema.

Clase Conectividad

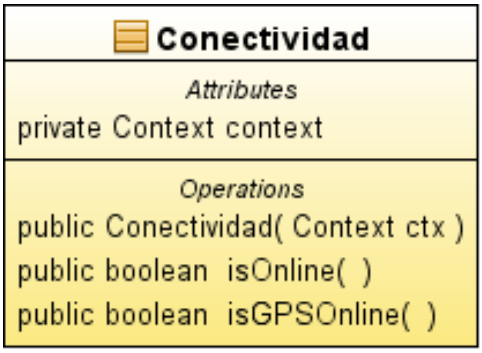


Figura 66: *Clase Conectividad*

Clase que sirve de apoyo para las clases java de los Activities que explicaremos más adelante. Es la encargada de comprobar si existe conexión de datos o si el GPS está activado. Los métodos de los que consta son los siguientes:

- `public Conectividad(Context ctx):` constructor de clase, al cual se le pasa como parámetro el contexto de la aplicación, necesario para conocer los recursos de conectividad (GPS y conexión de datos). Dicho contexto se almacenará en un atributo de clase.

- `Public boolean isOnline():` método que se encarga de comprobar si existe conexión de datos. En el caso de que existiera devolvería true y, en caso contrario, devolvería false.

- `Public boolean isGPSOnline():` método que comprueba si el GPS del dispositivo está activado, devolviendo true en el caso de que lo este y false si no lo está.

Clase Posicionamiento

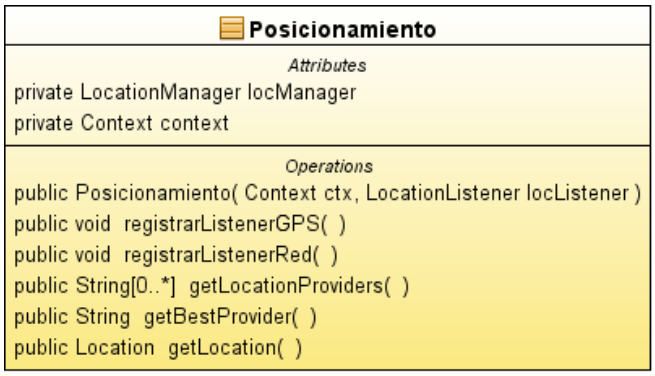


Figura 67: *Clase Posicionamiento*

Es una clase que, al igual que la anterior, sirve de apoyo para las clases de los Activity. En este caso, se encarga de realizar las tareas necesarias para obtener la localización del usuario tanto mediante GPS como mediante la triangulación de la posición a través de las antenas de telefonía móvil. Dispone de dos variables de clase:

- `private LocationManager locManager:` instancia de la clase que permite el acceso a los servicios de localización del sistema.

- `Private Context context:` almacenará el contexto de la aplicación.

Además de los atributos, dispone de una serie de métodos de los cuales destacan los siguientes:

- `public Posicionamiento(Context ctx, LocationListener locListener):` constructor de la clase. Se le pasa como parametro el locationListener deseado y ontexto de la aplicación.
- `public void registrarListenerGPS():` método encargado de registrar y establecer los parametros de refresco del posicionamiento a través del GPS.

- `public void registrarListenerRed():` método encargado de registrar y, como en el anterior, establecer los parámetros de refresco del posicionamiento, en este caso, a través de la red de datos (antenas de telefonía).

- `public List<String> getLocationProviders ():` método que devuelve un listado con todos los proveedores de localización del dispositivo.

- `public String getBestProvider():` método que devuelve el nombre del proveedor de localización activado más preciso.

- `public Location getLocation():` método que obtiene la localización del usuario.

Clase DibujaMapas

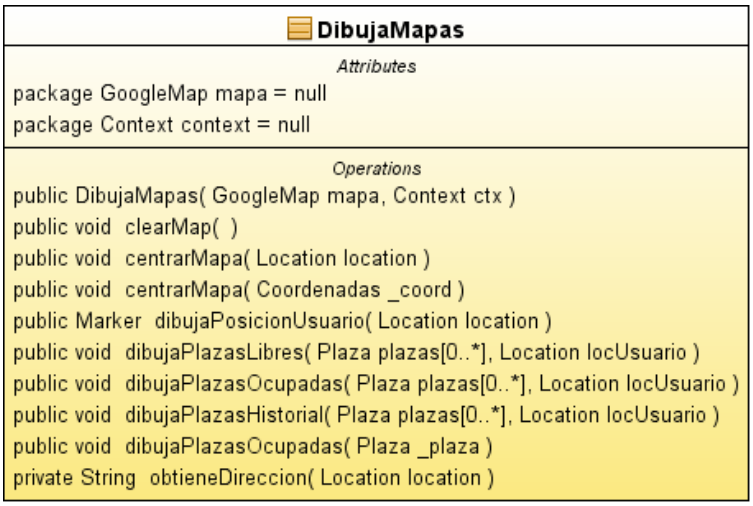


Figura 68: *Clase DibujaMapas*

La clase `DibujaMapas` es la encargada de realizar el manejo de los mapas que se muestran en la aplicación, es decir, dibujar las plazas libres, la posición del usuario, obtener la información asociada a cada *marker* (dirección y distancia por ejemplo), etc. Para ello esta clase dispone de una serie de atributos y métodos que posibilitan el manejo de los mapas, dichos atributos y métodos son:

- `GoogleMap mapa:` variable que contendrá la instancia del mapa que se va a tratar.

- `Context context:` variable que contendrá la instancia del contexto de la aplicación.

- `public DibujaMapas (GoogleMap mapa, Context ctx):` constructor de la clase. Recibe como parámetros la instancia del mapa que se va a manejar y el contexto de la aplicación.

- `public void clearMap ():` método que borra todos los elementos dibujados en el mapa.

- `public void centrarMapa (Location location)`: centra el mapa en la localización indicada como parámetro.

- `public void centrarMapa (Coordenadas _coord)`: este método realiza la misma función que el anterior, pero recibe como parametro una instancia de la clase Coordenadas.

- `public Marker dibujaPosicionUsuario (Location location)`: método que dibuja la posición del usuario. Dicha posición se recibe en una instancia de `Location` pasada como parámetro. Además devuelve la instancia del marker dibujado.

- `ublic void dibujaPlazasLibres (ArrayList<Plaza> plazas, Location locUsuario)`: método que dibuja las plazas libres recibidas como parámetro en un ArrayList. Además también recibe como parámetro una instancia de `Location` correspondiente a la posición de usuario para calcular la distancia existente entre éste y cada plaza.

- `public void dibujaPlazasOcupadas (ArrayList<Plaza> plazas, Location locUsuario)`: método similar al anterior, pero en este caso dibujará las plazas ocupadas.

- `public void dibujaPlazasHistorial (ArrayList<Plaza> plazas, Location locUsuario)`: método similar a los dos anteriores con la diferencia que, en este caso, servirá para dibujar las plazas almacenadas en el historial y su información asociada.

- `public void dibujaPlazasOcupadas (Plaza _plaza)`: método que dibuja en el mapa la plaza que ocupa el usuario, la cual se recibe como parámetro.

- `private String obtieneDireccion (Location location)`: método que devuelve la dirección de una localización del mapa que ha sido pasada como parámetro.

Clase Conexión

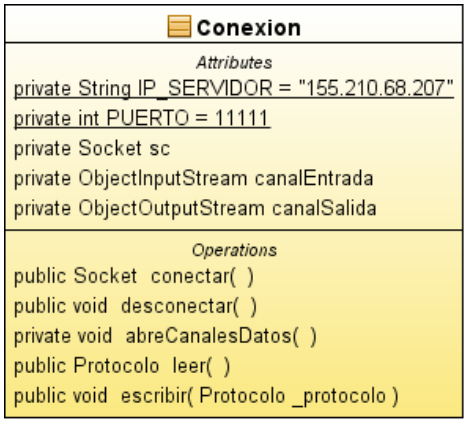


Figura 69: *Clase Conexion*

Esta clase es la encargada de establecer la conexión con el servidor y realizar la lectura y escritura de datos en los canales correspondientes. Para ello dispone de los siguientes atributos:

- `private String IP_SERVIDOR:` constante que almacena la IP de la máquina que ejecuta la aplicación servidor.
- `Private int PUERTO:` constante que almacena el puerto a través del cual se conectará.
- `Private Socket sc:` variable que contendrá la instancia del socket que se abrirá para conectar con el servidor.
- `Private ObjectInputStream canalEntrada:` variable que contendra la instancia del canal de entrada.
- `Private ObjectOutputStream canalSalida:` variable que contendrá la instancia del canal de salida creado a partir del socket.

Aparte de estos atributos, la clase dispone de los siguientes métodos:

- `public Socket conectar()` : método que envía la petición de conexión al servidor y que devuelve el socket creado tras realizar dicha conexión.
- `public void desconectar()` : cierra los canales de entrada y salida, además del socket, finalizando así la conexión con el servidor.

- `private void abreCanalesDatos()` : este método abre los canales de entrada y salida utilizando la instancia de la clase Socket obtenida al conectar.

- `public Protocolo leer()`: método que lee la información contenida en el canal de entrada y devuelve la instancia de la clase Protocolo con los datos leídos.

- `public void escribir(Protocolo _protocolo)`: método el cual escribe en el canal de salida la instancia de la clase Protocolo recibida como parámetro.

Clase OperacionesServidor

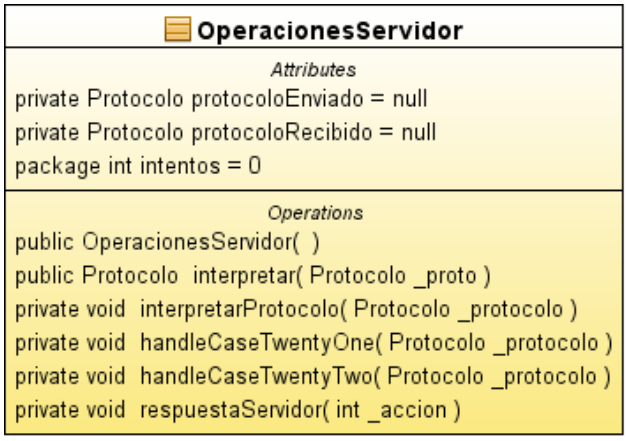


Figura 70: *Clase OperacionesServidor*

Clase que se encarga de interpretar los datos recibidos del servidor y enviar la respuesta pertinente, utilizando los métodos proporcionados por la clase `Conexion`. Para ello dispone de los siguientes atributos y métodos:

- `private Protocolo protocoloEnviado`: variable que contendrá la instancia del protocolo que se ha enviado.

- `private Protocolo protocoloRecibido`: variable que contendrá la instancia del protocolo que se ha recibido.

- `private Conexion con`: instancia de la clase Conexión que permitira realizar la lectura y escritura.

- `int intentos` : variable que contiene la cuenta del número de intentos a la hora de realizar una operación con el servidor.

- `public OperacionesServidor()`: constructor de clase.

- `public Protocolo interpretar(Protocolo _proto)` : método que se encarga de permanecer leyendo del canal mientras el socket este abierto y devolver la instancia de Protocolo leída del canal.

- `private void interpretarProtocolo(Protocolo _protocolo)` : método que se encarga de interpretar el protocolo recibido y realizar una acción u otra dependiendo del código de operación que contenga.

- `private void handleCaseTwentyOne(Protocolo _protocolo)`: método que envía al servidor la petición de desconexión y desconecta de éste.

- `private void handleCaseTwentyTwo(Protocolo _protocolo)`: método, que en el caso de que no se haya realizado la acción requerida correctamente, reintenta el número de veces que este estipulado.

- `private void respuestaServidor(int _accion)`: método que se encarga de enviar al servidor la respuesta con el resultado de una acción.

Clase LoginTask, RegisterTask y LanzaConexión

Son tres clases que extienden de la clase `AsyncTask` que se utiliza para ejecutar pequeñas tareas en segundo plano. Estas clases las se utilizaran en las clases `RegistroActivity`, `LoginActivity`, `InicIoActivity`, `PlazaOcupadaActivity`, `PerfilActivity` y `AparcamientosRecientesActivity`, por ello el diagrama se muestra en los correspondientes apartados. En nuestro caso lo hacemos para no ejecutar la conexión con el servidor y las operaciones de lectura/escritura del canal en el hilo principal de la aplicación. Para ello, aparte de tener el constructor de clase, sobrescribe los siguientes métodos de la clase `AsyncTask`:

- `protected void onPreExecute()` : método que ejecuta las acciones previas deseadas antes de ejecutar las acciones en nsegundo plano.

- `protected Protocolo doInBackground(Protocolo... arg0)`: método que ejecuta las acciones deseadas en segundo plano.

- `protected void onPostExecute(Protocolo _protoResultado`: método que se ejecuta después del método anterior y que realiza el procesado de los datos obtenidos tras la ejecución en segundo plano.

Clase Item_object

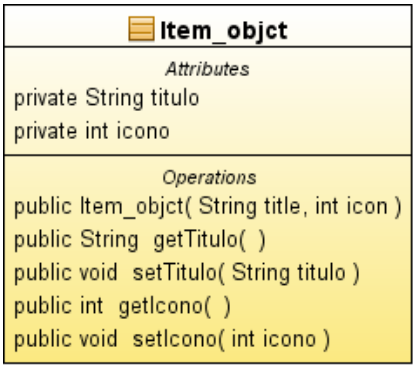


Figura 71: *Clase Item_objct*

Esta clase es la encargada de almacenar cada uno de los ítems que aparecerán en el menú lateral de nuestra aplicación. Es decir, cada uno de dichos ítems se corresponderá con una instancia de esta clase. Los atributos que forman la clase son los siguientes:

- `private String titulo:` cadena de texto que contendrá el título de la entrada o ítem del menú.
- `Private int icono:` variable entera que almacenará la referencia al icono correspondiente.

Además de estos atributos, la clase dispone de un constructor que recibe como parámetros el título y la referencia al icono, y los métodos get y set necesarios para modificar y obtener estos dos atributos.

Clase NavigationAdapter

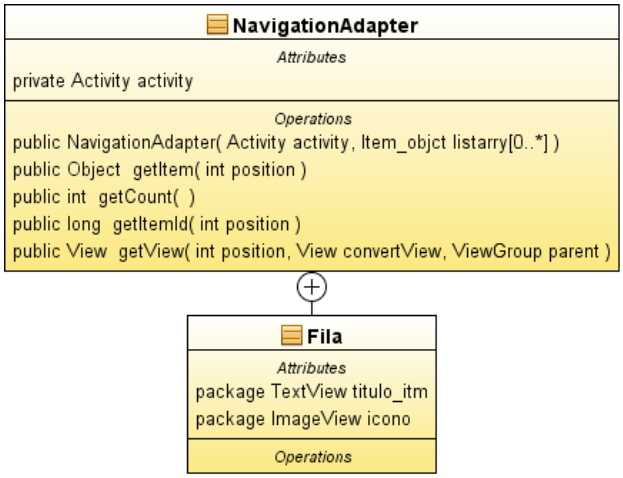


Figura 72: *Clase NavigationAdapter*

La clase `NavigationAdapter` extiende de la clase `BaseAdapter` y se encarga de dibujar y formatear los ítems del *NavigationDrawer* o menú lateral. Dispone de un constructor de clase que recibe como parámetro el Activity en el que se mostrará y la lista de entradas de las que dispondrá el menú. Además de los métodos necesarios para manejar el formateo de las entradas dispone de una subclase, llamada Fila. Cada instancia de esta subclase, servirá para representar gráficamente cada una de las instancias de la clase `Item_object`.

Interfaz OnclickListener

Interfaz que implementan las actividades que desean capturar los eventos de clic o pulsación sobre la pantalla. Para ello deberán implementar el siguiente método:

- `public void onClick(View view)`

Clase Acticvity

Clase del API de Android, de la cual heredan todas las clases java que implementan la lógica de las actividades o pantallas. Por ello sobrescriben los siguientes métodos:

- `protected void onCreate(Bundle savedInstanceState)`: método sobrescrito de la clase Activity que se ejecuta en el momento en el que se crea la pantalla.
- `protected void onRestart()`: método sobrescrito de la clase Activity que se ejecuta después de que el activity se haya detenido antes de volver a ser iniciado.

- `Protected void onResume()`: método sobrescrito de la clase Activity que se ejecuta cuando el activity comienza a interactuar con el usuario.

Algunos de los Activity de nuestra aplicación no extienden directamente de la clase Activity sino que lo hacen de `FragmentActivity` que a su vez extiende a la clase Activity. Estas clases que que heredan de `FragmentActivity` lo hacen así porque necesitan soportar *Fragments*, que en nuestro caso serán, para contener el mapa. Dichas clases que hereden de `FragmentActivity` también sobrescribirán los métodos nombrados unas líneas más arriba.

Clase RegistroActivity

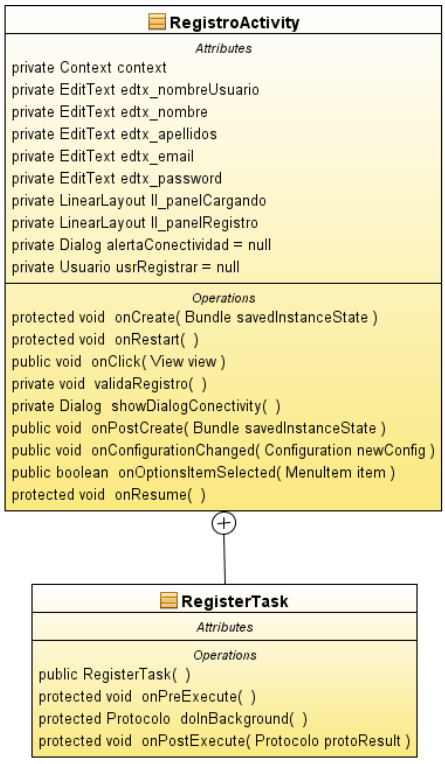


Figura 73: *Clase RegisterActivity*

La clase `RegistroActivity` implementa las funcionalidades de la pantalla o *Activity* de registro de usuarios. Además extiende a la clase Activity por lo que sobrescribe varios de sus métodos (`onCreate()`, `onRestart()` y `onResume()`) y, además, implementa el método definido en el interfaz `OnclickListener`. Para realizar el registro dispone de 5 variables de tipo `EditText` que se corresponden a cada campo del formulario y contendrán los datos introducidos además de dos variables `LinearLayout` que contendrán las instancias del panel con el formulario de registro y otro con la barra de progreso, cosa que nos permitirá ocultarlos y mostrarlos. Además tiene una

variable de tipo `Dialog` que contendrá la instancia del dialogo que se muestre cuando se produzca algún error y una variable de la clase Usuario que contendrá la instancia del usuario a registrar. Por otro lado los métodos que implementa son:

- `public void validaRegistro()` : método que realiza la validación y comprobaciones necesarias de los datos introducidos antes de enviar los datos al servidor para realizar el registro del usuario. En caso de error, mostrará un mensaje de error en el EditText correspondiente.

- `private Dialog showDialogConectivity ():` método que muestra un dialogo de alerta cuando el dispositivo no dispone de conexión a internet.

Aparte de estos métodos, la clase `RegistroActivity` contiene la subclase `RegisterTask` que extiende a la clase `AsyncTask`. Esta subclase ya ha sido expuesta anteriormente.

Clase LoginActivity

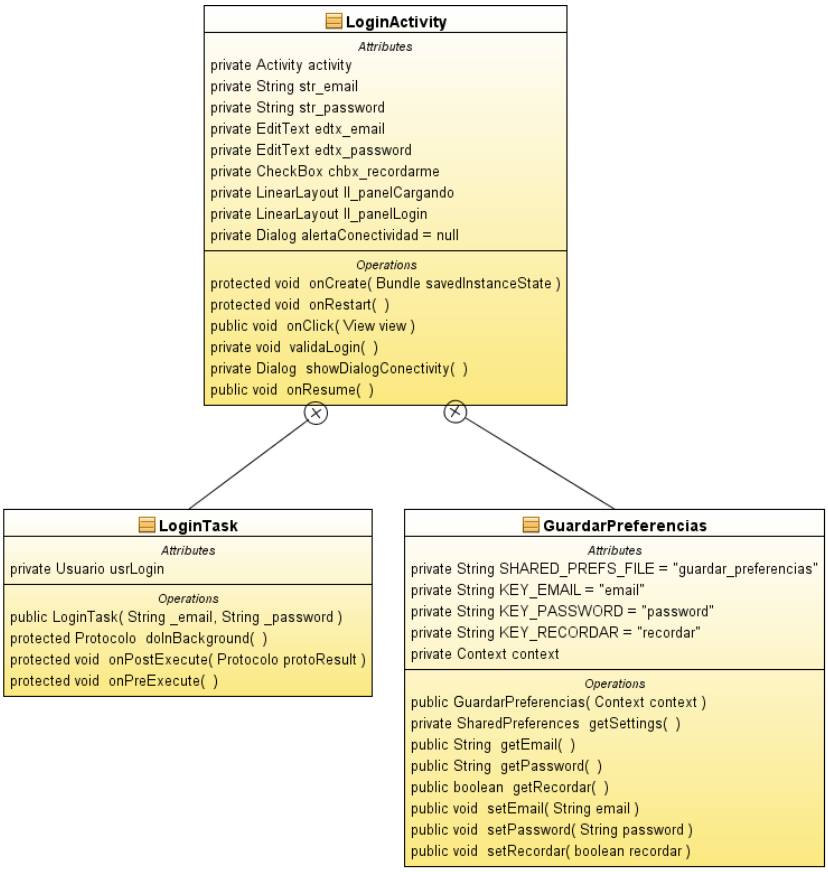


Figura 74: *Clase LoginActivity*

Es la clase que implementa las funcionalidades de la pantalla de inicio de sesión. La clase `LoginActivity` extiende de Activity, y por lo tanto, sobrescribirá los métodos anteriormente citados (`onCreate()`, `onRestart()` y `onResume()`). Además implementa el método `onClick()` del interfaz `OnClickListener` también detallado anteriormente. Aparte de esto dispone de una serie de variables que almacenarán los elementos de interfaz como son `LinearLayout`, `EditText`, `CheckBox` y `Dialog`. También dispone de dos atributos String que serán los encargados de almacenar el usuario y la contraseña obtenidos de los `EditText`. Sumándose a los métodos ya citados y explicados, esta clase dispone de los siguientes atributos y métodos:

- `private LoginTask userLoginTask`: variable que contendrá la instancia del método que extiende AsyncTask anteriormente explicado.

- `private GuardarPreferencias guardarPreferencias`: variable que contendrá la instancia de la clase encargada de guardar las preferencias para recordar o no las credenciales de acceso.

- `private String str_email`: variable que contendrá el email introducido.

- `private String str_password`: variable que contendrá la contraseña introducida.

- `private EditText edtx_email`: esta variable contendrá la instancia del EditText encargado de recoger el *login* de acceso.

- `private EditText edtx_password`: variable que contendrá la instancia de `EditText` encargada de recoger la contraseña de acceso.
- `private CheckBox chbx_recordarme`: ésta variable de tipoCheckBox contendrá la instancia del elemento checkbox que se mostrará en el interfaz.

- `private LinearLayout ll_panelCargando`: variable que contendrá la instancia del panel que mostrará la barra de progreso.

- `private LinearLayout ll_panelLogin`: variable que contendrá la instancia del panel que contendrá el formulario y los EditText de los títulos de este.

- `private Dialog alertaConectividad`: es una variable que contendrá la instancia del dialogo de alerta, en el caso de que se muestre.

- `public void validaLogin()`: método encargado de comprobar que se hayan introducido datos en los EditText y que estos sean

correctos. Además mostrará un mensaje de error en el `EditText` correspondiente.

- `Private Dialog showDialogConectivity()`: método que mostrará el dialogo de alerta cuando no exista conexión a Internet.

Esta clase, también contiene la subclase `LoginTask` explicada anteriormente y la subclase `GuardarPreferencias` que será la encargada de guardar los datos de inicio de sesión en un archivo de configuración, en caso de que se haya seleccionado el *checkbox* Recordarme.

NavigationDrawer

No se trata de ninguna clase, pero es un elemento en común que forma parte de las clases que se exponen de aquí en adelante. Se trata del nombre de la herramienta utilizada para crear el menú lateral de la aplicación. Para su creación, manejo y visualización, las clases deberán tener los siguientes atributos y métodos:

- `private String[] titulos`
- `private DrawerLayout NavDrawerLayout`
- `private ListView NavList`
- `private ArrayList<Item_objct> NavItms`
- `private TypedArray NavIcons`
- `private ActionBarDrawerToggle mDrawerToggle`
- `private CharSequence mDrawerTitle`
- `private CharSequence mTitle`
- `NavigationAdapter NavAdapter`
- `protected void onPostCreate(Bundle savedInstanceState)`
- `public void onConfigurationChanged(Configuration newConfig)`
- `public boolean onOptionsItemSelected(MenuItem item)`

Clase InicioActivity

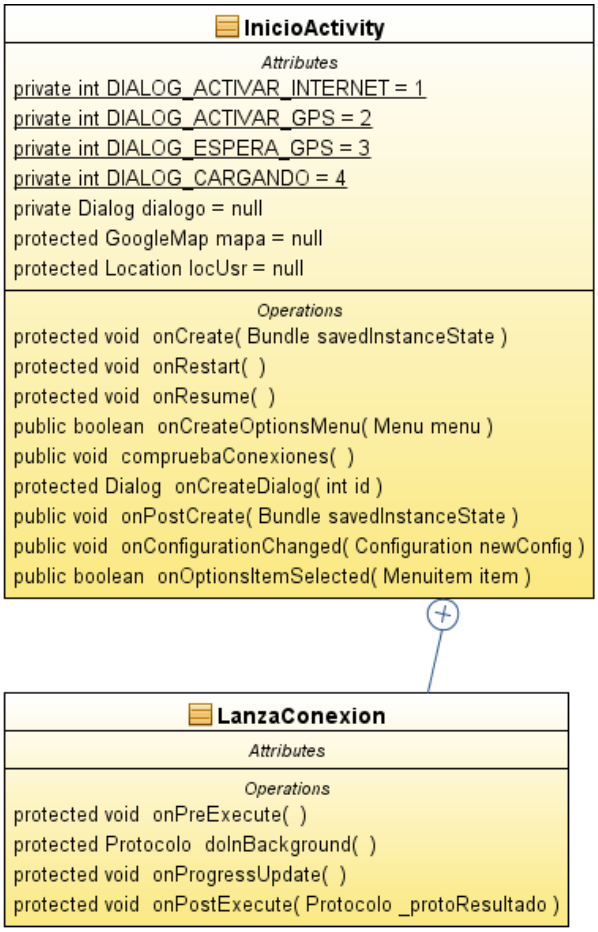


Figura 75: *Clase InicioActivity*

Clase que implementa la lógica de la pantalla o actividad encargada de mostrar las plazas libres al usuario. Para ello extiende de la clase `FragmentActivity` y por lo tanto sobrescribirá los métodos que se han citado más arriba. Además también implementará el interfaz `OnClickListener`. Esta clase contiene una serie de constantes int que sirven para indicar el código del dialogo de alerta que se mostrará. Además, dispone de las siguientes variables:

- `private Dialog dialogo`: contendrá la instancia del dialogo de alerta, en caso de que sea mostrado.
- `protected GoogleMap mapa`: contendrá la instancia del mapa que se muestra en la actividad.
- `protected Posicionamiento pos`: esta variable contiene la instancia de la clase Posicionamiento que nos permite obtener la localización del usuario.

- `protected Location locUsr:` variable que contendrá la instancia de Location con los datos de ubicación del usuario.

Además de estas variables, métodos sobrescritos e implementados, la clase `InicioActivity` tiene los siguientes métodos:

- `protected Dialog onCreateDialog(int id):` método que recibiendo el id del dialogo a mostrar, lo crea y muestra en la pantalla.

- `public void compruebaConexiones():` método que se encarga de realizar las llamadas a los métodos de la clase Conectividad para comprobar que se dispone de los servicios de conectividad y posicionamiento necearios.

Por otro lado, y como ya ocurre con las dos de las anteriores, esta clase contiene una subclase que extiende de `AsyncTask` llamada `LanzaConexion`, cuyos métodos han sido detallados en un apartado anterior.

Clase PlazaOcupadaActivity

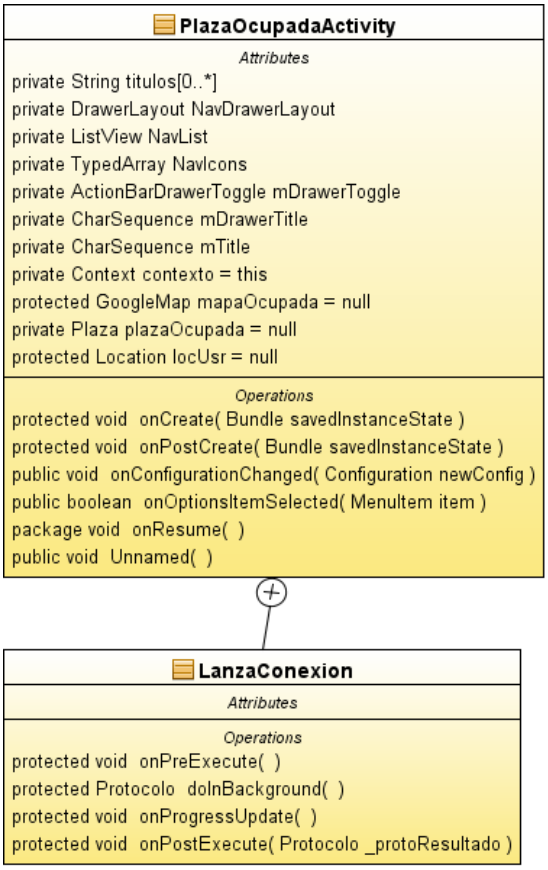


Figura 76: *Clase PlazaOcupadaActivity*

La clase `PlazaOcupadaActivity` es la que implementa la lógica correspondiente a la pantalla que muestra la posición y demás datos de la plaza de aparcamiento que ocupa un usuario. Extiende de la clase `FragmentActivity` y por lo tanto sobrescribirá los métodos que se han citado anteriormente, además, implementa el interfaz `OnclickListener`. Para proporcionar las funcionalidades necesarias, la clase `PlazaOcupadaActivity` dispone de las siguientes variables de clase:

- `private Context context:` variable que contendrá el contexto de la aplicación.
- `protected GoogleMap mapaOcupada:` variable que contendrá una instancia del mapa que se muestra en pantalla.
- `private Plaza plazaOcupada:` variable que contendrá la instancia de la plaza que se ha ocupado.
- `protected Location locUsr:` esta variable de tipo Location contendrá una instancia con los datos de localización del usuario.

Además de los atributos que se acaban de detallar, sobrescribe los métodos de la clase `FragmentActivity` e implementa el de `OnclickListener` como los cuales se han detallado anteriormente. La clase `PlazaOcupadaActivity` también dispone de una clase `LanzaConexión` que extiende de `AsyncTask` y cuyos métodos han sido explicados anteriormente.

Clase PerfilAcitivity

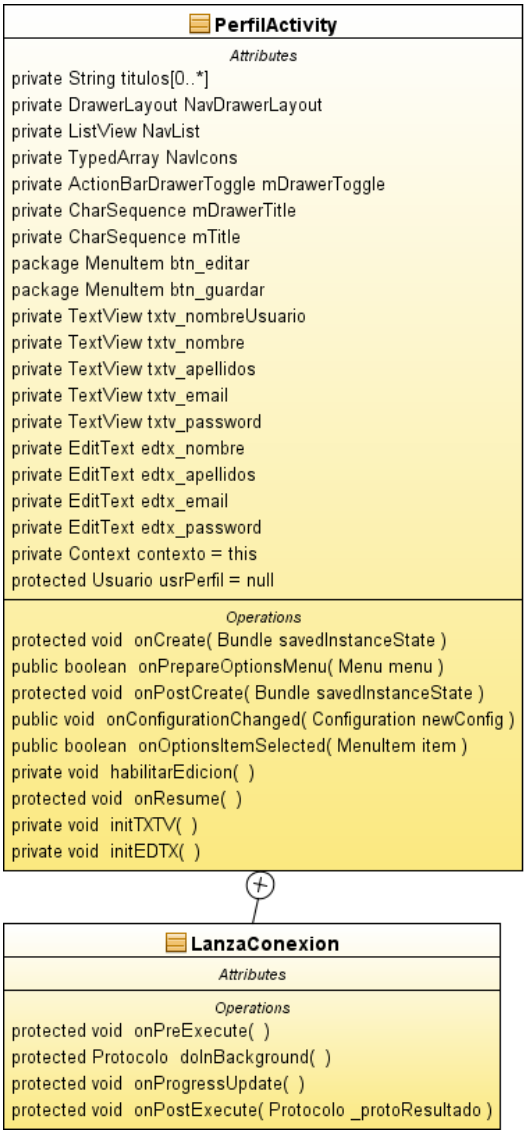


Figura 77: *Clase PerfilActiviy*

Esta clase es la encargada de implementar la lógica de la pantalla de perfil de usuario. Para ello extiende de `Activity` y, por lo tanto, implementará los métodos citados anteriormente. Para permitir las funcionalidades requeridas en esta actividad, dispone de los siguientes variables y métodos:

- `MenuItem btn_editar`: contendrá la instancia del botón que permitirá la edición del perfil.
- `MenuItem btn_guardar`: contendrá la instancia del botón que permitira guardar los cambios.
- Las siguientes variables de tipo TextView contendrán las instancias de cada vista que, además, mostrará los datos del usuario:

  - `private TextView txtv_nombreUsuario`
  - `private TextView txtv_nombre;`
  - `private TextView txtv_apellidos;`
  - `private TextView txtv_email;`
  - `private TextView txtv_password;`

- Las siguientes variables de tipo `EditText` contendrán las instancias de los campos de este tipo que conformaran el formulario de edición del perfil:

  - `private EditText edtx_nombre;`
  - `private EditText edtx_apellidos;`
  - `private EditText edtx_email;`
  - `private EditText edtx_password;`

- `private Context contexto:` variable que almacenará el contexto de la aplicación.

- `protected Usuario usrPerfil:` variable de tipo usuario que contendrá la instancia del usuario que se muestra en el perfil.

- `public boolean onPrepareOptionsMenu(Menu menu):` método que crea los botones de editar y guardar contenidos en la ActionBar.

- `private void habilitarEdicion():` método que hace visibles los EditText y oculta los TextView para habilitar la edición del perfil.

- `private void deshabilitarEdicion():` método similar al anterior, que deshabilita la edición del perfil ocultando los EditText y mostrando los TextView.

- `private void initTXTV()`: método encargado de inicializar los TextVIew que contendrán la información del perfil de usuario.

- `private void initEDTX()`: método encargado de inicializar el texto de los EditText que forman el formulario de edición del perfil de usuario.

Además de todo lo citado, esta clase dispone también de una subclase que extiende a la clase `AsyncTask`, en este caso será también `LanzaConexión` cuyos métodos han sido explicados anteriormente.

Clase AparcamientosRecientesActivity

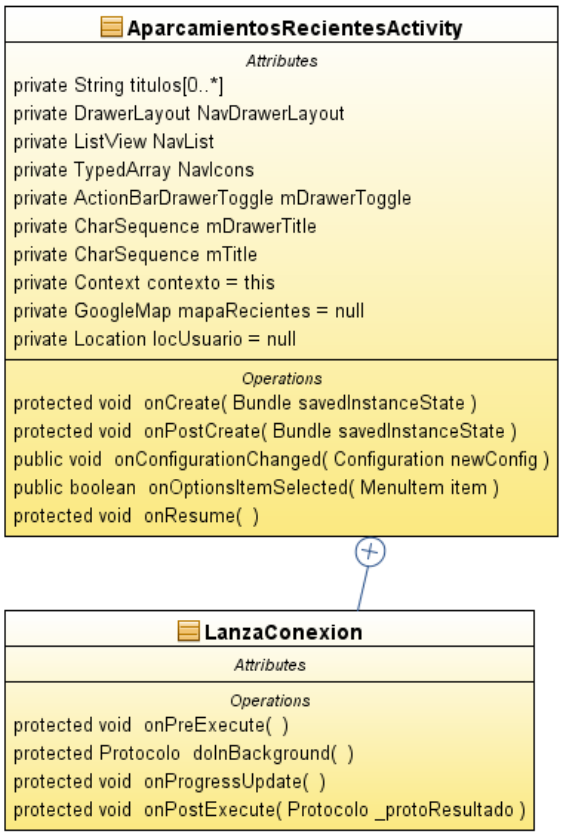


Figura 78: *Clase AparcamientosRecientesActivity*

La clase `AparcamientosRecientesActivity` implementa la lógica correspondiente a la actividad que muestra el historial de plazas ocupadas recientemente. Como el resto de pantallas que muestran mapas, extiende de la clase `FragmentActivity` y, por lo tanto, como ya se ha explicado antes sobrescribirá alguno de sus métodos. Además también implementa el método definido en la interfaz `OnClickListener`. Para la realización de este cometido, dispone de las siguientes variables de clase:

- `private Context contexto = this`: almacenará el contexto de la aplicación.

- `private GoogleMap mapaRecientes`: contendrá una instancia del mapa en el cual se van a mostrar las plazas donde se ha estacionado recientemente.

- `private Location locUsuario`: contiene una instancia de la clase `Location` con toda la información relativa a la posición del usuario.

Además de estas variables y los métodos sobrescritos e implementados, la clase `AparcamientosRecientesActivity` contiene una

subclase `LanzaConexion`, que como se ha citado repetidas veces extiende de `AsyncTask`.

Clase AcercaDeActivity

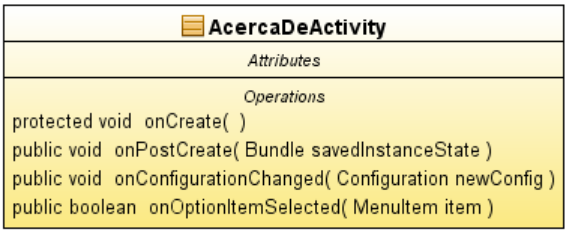


Figura 79: *Clase AcercaDeActivity*

Esta clase, a pesar de ser la más sencilla de todas, también extiende a la clase `Activity` y, por lo tanto, deberá sobrescribir al menos, alguno de sus métodos. En este caso solo sobrescribe el método `onCreate()`.

6.4.Diagramas de Actividad

En este aparatado, se comentarán los diferentes diagramas de actividad, diferenciando los diagramas de la aplicación servidor y los de la aplicación AparcaDroid cliente.

6.4.1. Servidor

Registrar usuario

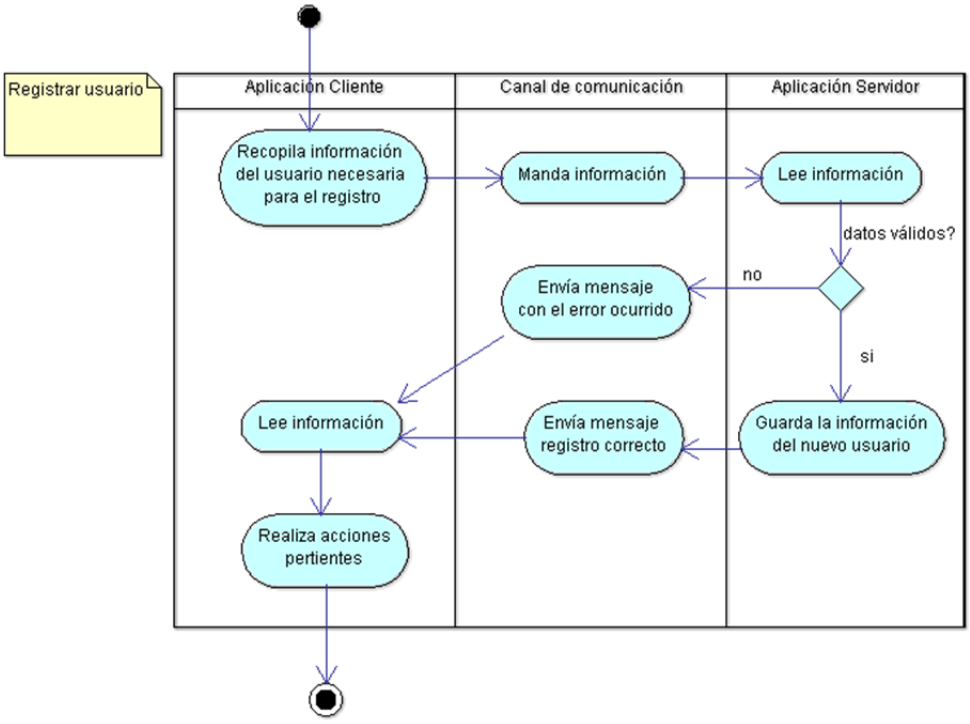


Figura 80: *Diagrama de actividad registrar usuario*

Como paso previo para poder interactuar con la aplicación, cada usuario deberá registrarse en el sistema. Para ello el servidor recibirá una petición de registro y la información relativa al usuario (datos personales, email y contraseña). Tras haber leído la información y comprobado su validez, registrará al usuario enviando, tras esto, la respuesta con el resultado (registro correcto o incorrecto) de las operaciones realizadas.

Autenticarse en el sistema

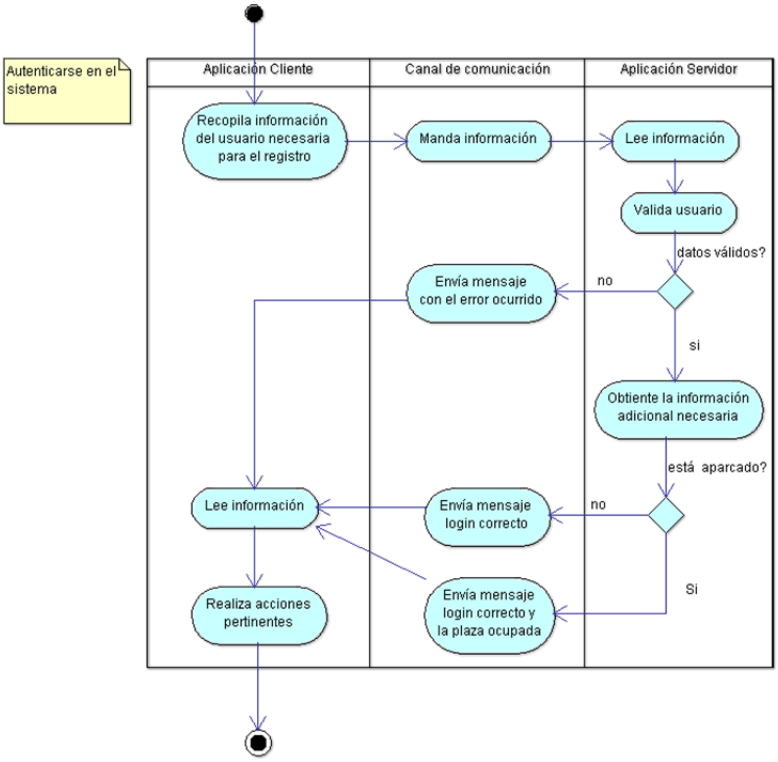


Figura 81: *Diagrama de actividad autenticarse en el sistema*

Tras haberse registrado y para poder empezar a interactuar con la aplicación, un usuario lanzará una petición de login que será recibida por el servidor. Tras leer los datos y comprobar que son válidos (existe el nombre de usuario y la contraseña es la correcta), el servidor obtendrá los datos adicionales necesarios, es decir, si el usuario está estacionado, caso en el que además del mensaje de confirmación de login correcto, enviará la plaza que ocupa. En caso de que el usuario o la contraseña sean incorrectos, el servidor detendrá el proceso de login y enviará un mensaje con el error ocurrido a la aplicación cliente.

Ver plazas libres

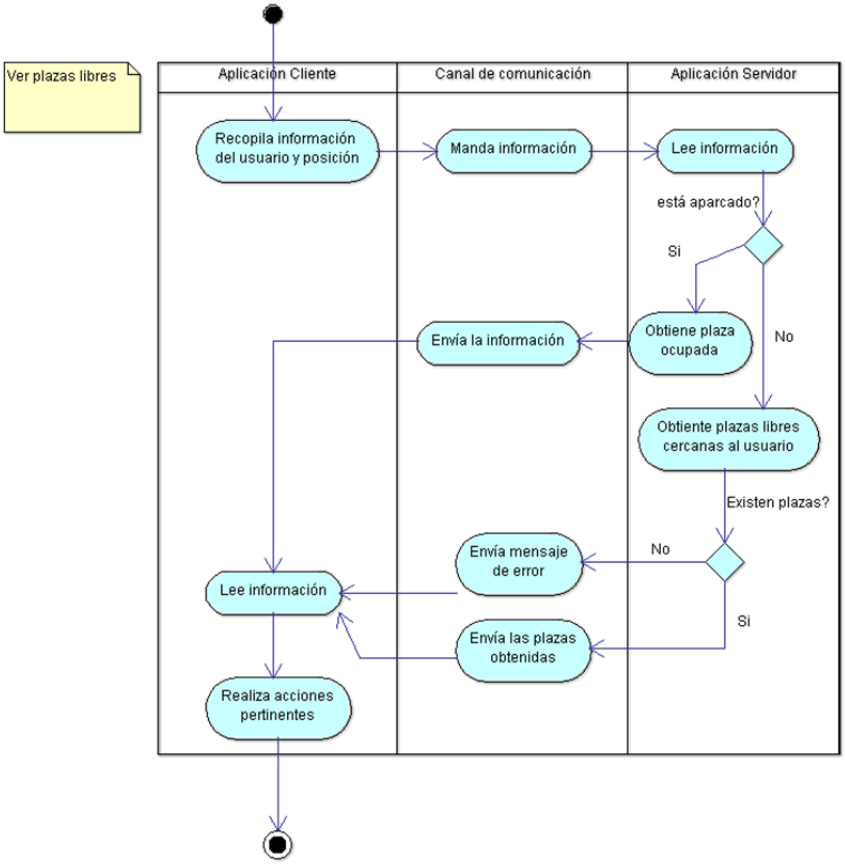


Figura 82: *Diagrama de actividad ver plazas libres*

Cuando inicie sesión el usuario o libere la plaza que ocupa, enviará una petición de obtención de las plazas libres que será recibida por el servidor. Tras recibir la información, realiza las comprobaciones pertinentes y, si el usuario no ocupa ninguna plaza, obtiene las plazas libres a su alrededor utilizando las Coordenadas recibidas.

Ocupar plaza

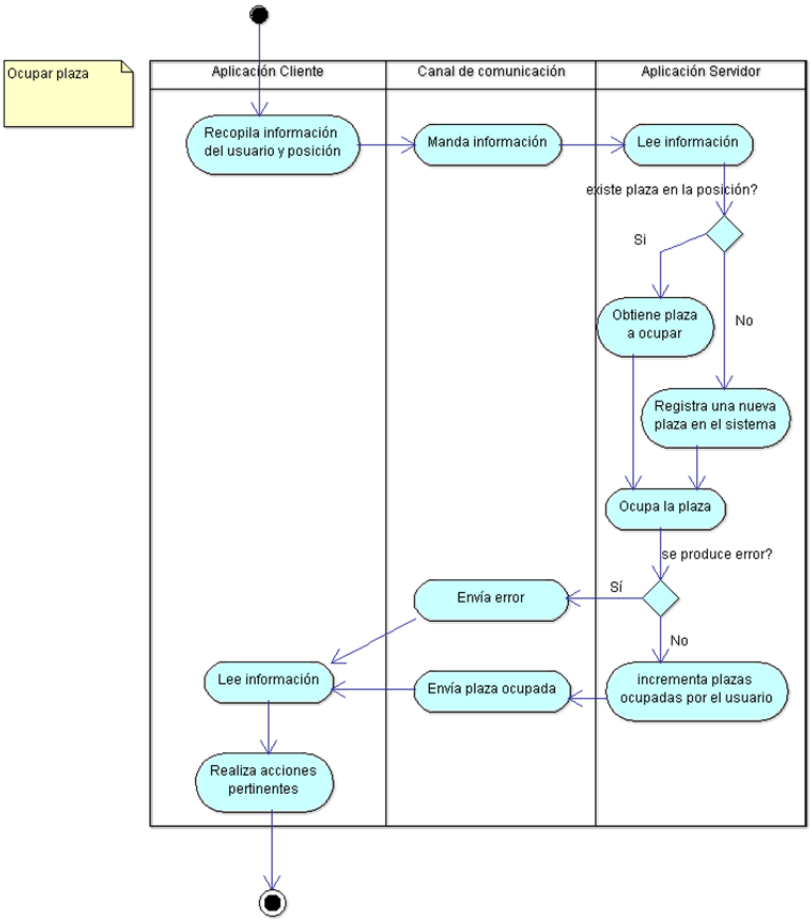


Figura 83: *Diagrama de actividad ocupar plaza*

Cuando un usuario ocupa una plaza, el servidor recibe la petición. Tras ello, lee la información, concretamente las coordenadas y la precisión del GPS del dispositivo que realiza la petición. Una vez hecho esto, comprueba si existe una plaza registrada con esas características en la localización indicada por las coordenadas, creándola nueva en el caso de que no existiera. Teniendo la plaza seleccionada, la ocupa y, si se ha realizado la acción correctamente, modifica la puntuación del usuario que ha hecho la petición (se incrementa en uno el número de plazas ocupadas) y se envía el mensaje de confirmación junto con la plaza que se ha ocupado. En caso que se produzca algún error ocupando la plaza seleccionada, se enviaría un mensaje de error y se detendría el proceso de ocupar la plaza en ese punto.

Ver plaza Ocupada

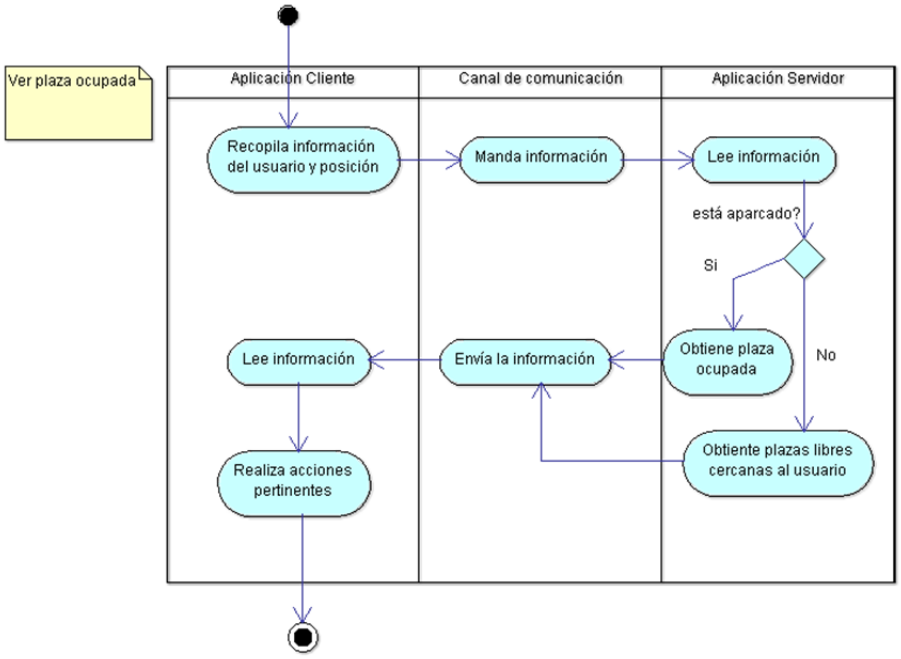


Figura 84: *Diagrama de actividad ver plaza ocupada*

Cuando un usuario inicia sesión, si se encuentra estacionado, el servidor le contestará adjuntando además la plaza que ocupa. Para ello cuando recibe la información de petición de login comprueba en la base de datos si el usuario está estacionado. Una vez comprobado, en el caso de que lo esté, el servidor contestará al cliente con un mensaje de operación realizada correctamente y adjuntando la plaza que ocupa.

Liberar plaza

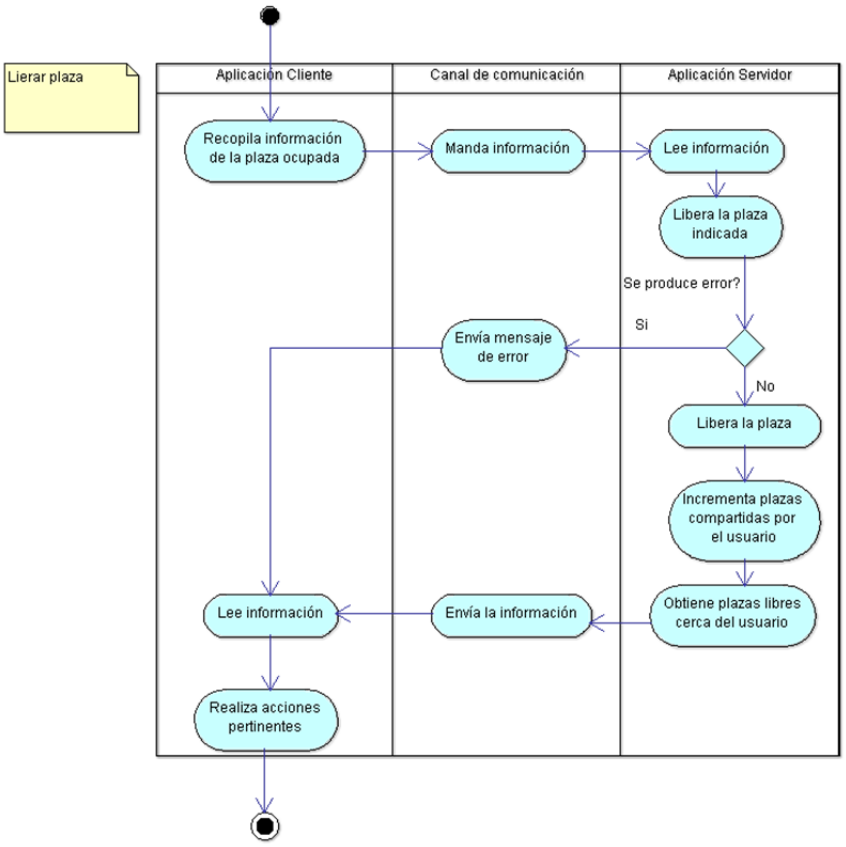


Figura 85: *Diagrama de actividad liberar plaza*

Cuando el usuario desea modificar su perfil, tras editarlo, manda una petición al servidor con la nueva información. El servidor tras recibir la petición valida los datos y, si son correctos, procede con la actualización de los datos del usuario en la base de datos. Durante el desarrollo de las operaciones para realizar este trámite, si se produce un error, enviará el mensaje con el error ocurrido y no seguirá con la ejecución de los siguientes pasos. Por el contrario si se realiza correctamente, enviará un mensaje de confirmación.

Modificar perfil

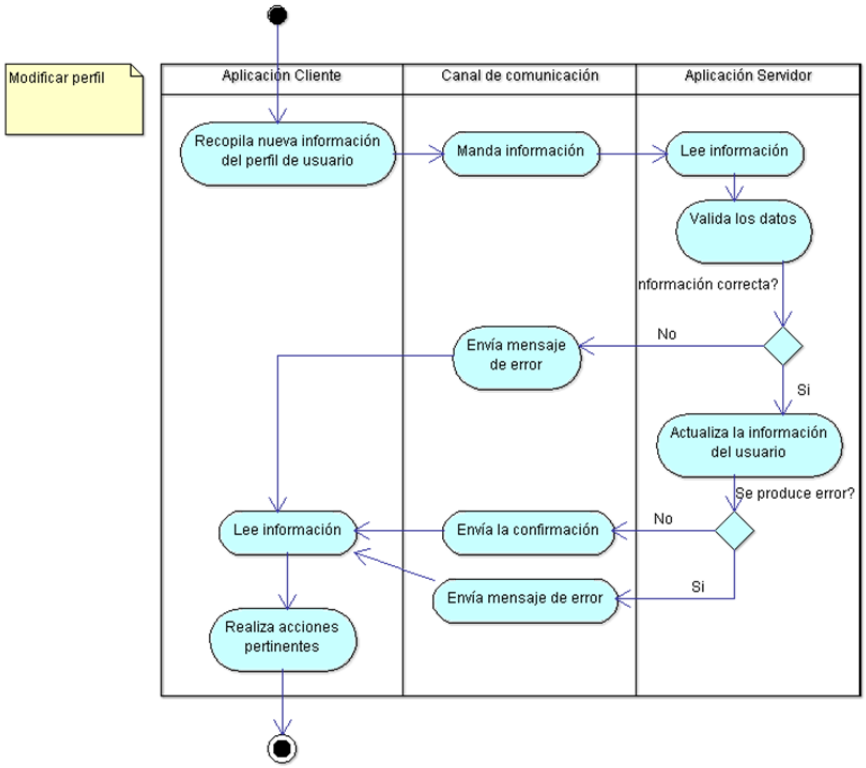


Figura 86: *Diagrama de actividad modificar perfil*

Cuando el usuario desea modificar su perfil, tras editarlo, manda una petición al servidor con la nueva información. El servidor tras recibir la petición valida los datos y, si son correctos, procede con la actualización de los datos del usuario en la base de datos. Durante el desarrollo de las operaciones para realizar este trámite, si se produce un error, enviará el mensaje con el error ocurrido y no seguirá con la ejecución de los siguientes pasos. Por el contrario si se realiza correctamente, enviará un mensaje de confirmación.

Conectar

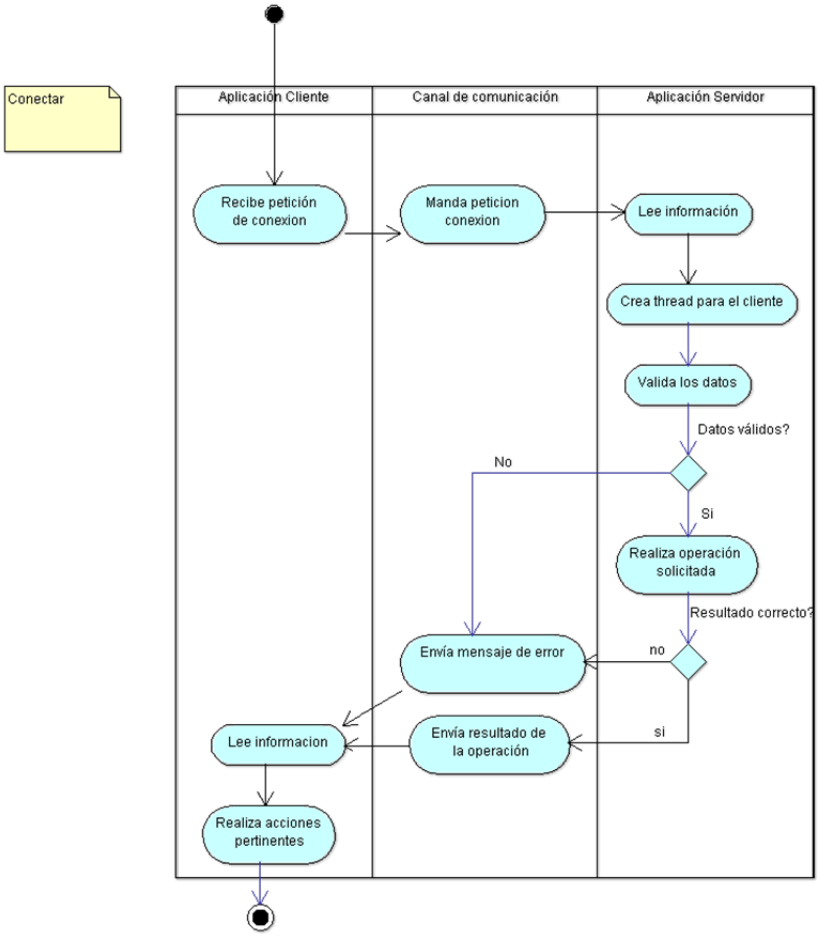


Figura 87: *Diagrama de actividad conectar*

Para interactuar con el servidor y realizar cualquier operación contra este, la aplicación cliente debe conectarse primero. Por esto, cuando el servidor recibe la petición de conexión y es aceptada, crea un nuevo hilo de ejecución para ese cliente, validando los datos y, si son correctos, procede a realizar la operación solicitada. Tras esto, en caso de producirse un error envía al usuario un mensaje con el código del error y en caso de realizar la acción correctamente mandará un mensaje de confirmación a la aplicación cliente.

Desconectar

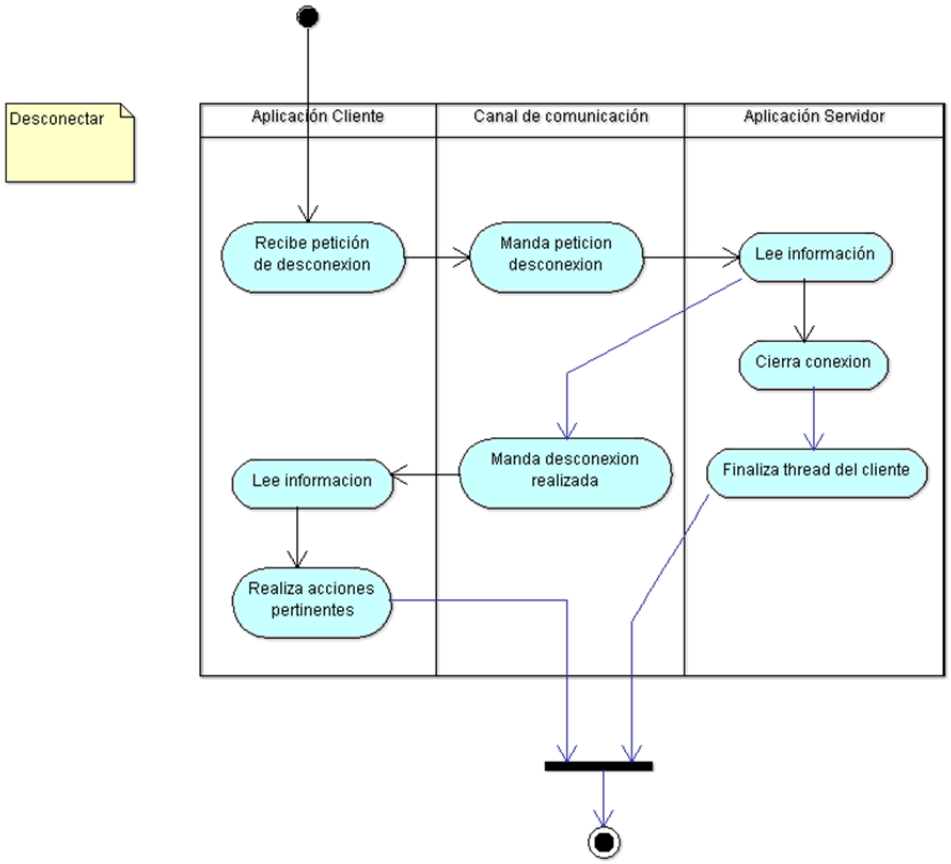


Figura 88: *Diagrama de actividad desconectar*

Una vez realizada la operación requerida por el usuario, la aplicación cliente solicita la desconexión del servidor. Cuando dicho servidor recibe la petición, manda un mensaje de desconexión realizada al cliente y procede a cerrar los canales de datos (`ObjectOutputStream` y `ObjectInputStream`), posteriormente cierra el socket con el cliente y finaliza el hilo creado para atender las peticiones de este.

6.4.2. Cliente

Registrar usuario

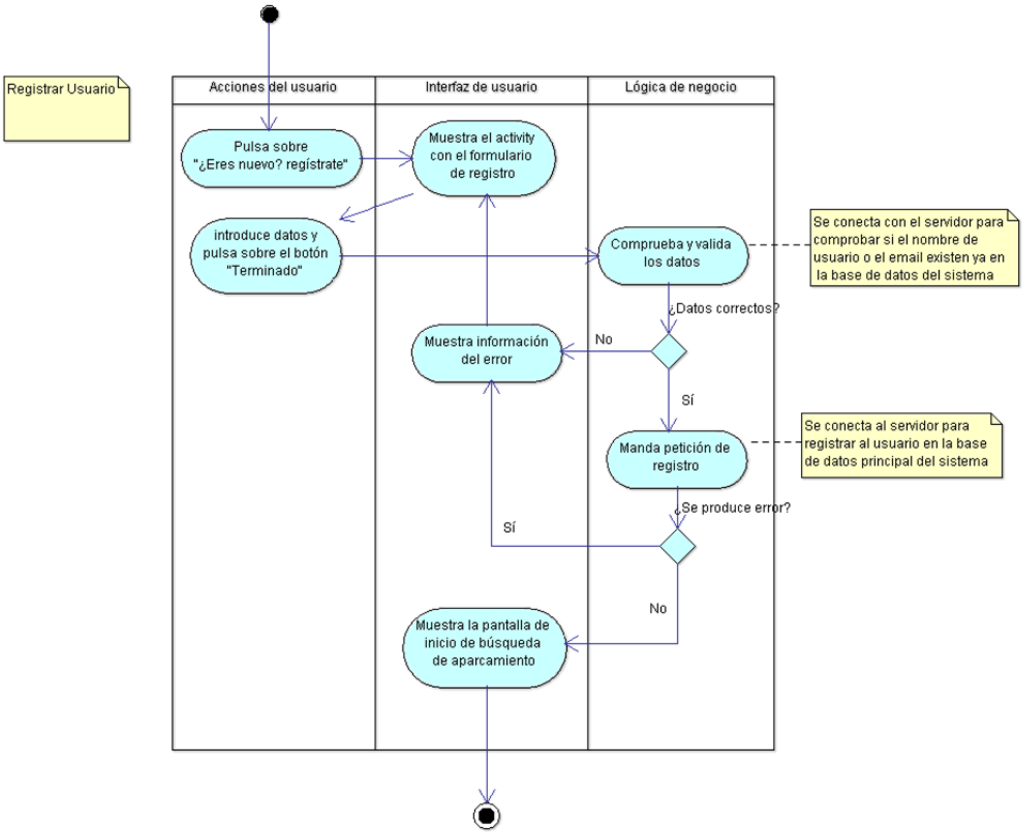


Figura 89: *Diagrama de actividad registrar usuario*

Como paso previo para poder interactuar con la aplicación, cada usuario deberá registrarse en el sistema. Para ello deberá acceder a la sección de registro de usuarios y rellenar el formulario que aparece. Tras pulsar terminado la aplicación cliente comprobará que todos los datos están rellenados adecuadamente y que no existe en la base de datos el nombre de usuario ni el email. Una vez hecha esta comprobación, en el caso de que se produzca un error, indicará cual en el campo afectado y, en caso de que no se produzca tal error, procederá a enviar la petición de registro al servidor, mostrando la pantalla de búsqueda de aparcamiento si se realiza correctamente y un mensaje con el error ocurrido en caso de que no.

Autenticarse en el sistema

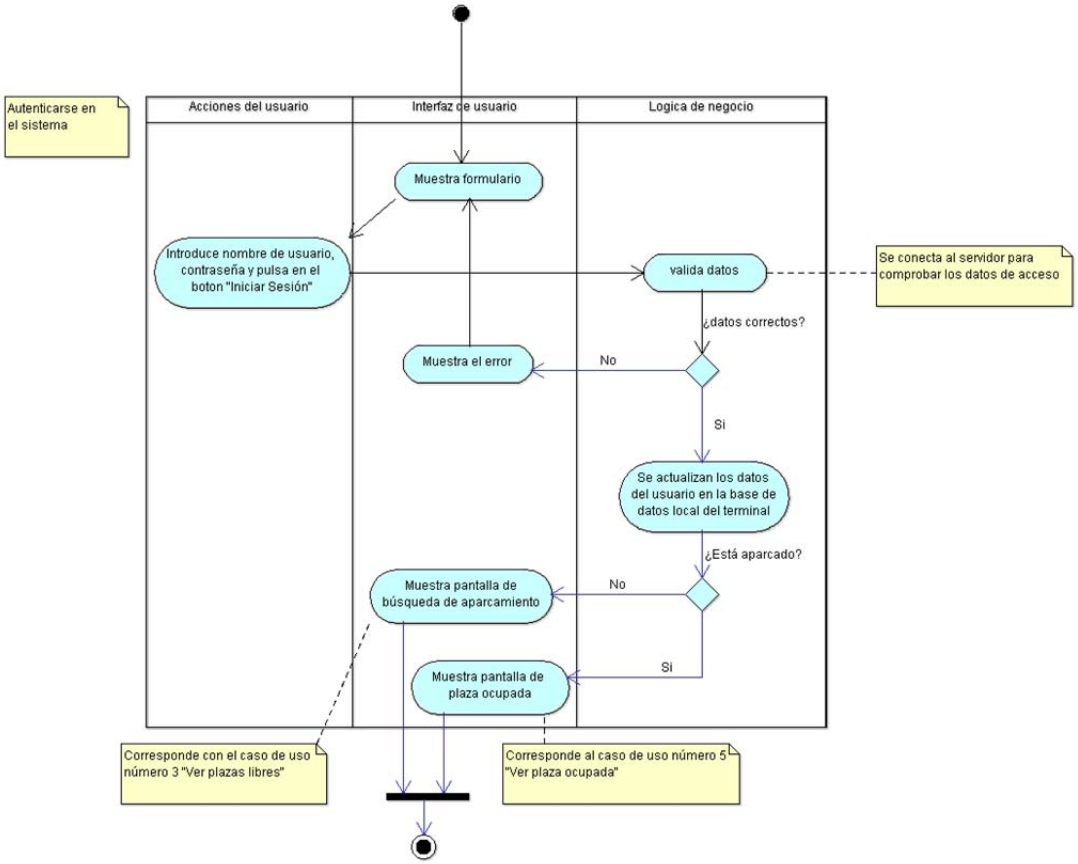


Figura 90: *Diagrama de actividad autenticarse en el sistema*

Una vez registrado el usuario, puede acceder a la aplicación tantas veces como desee utilizando su email y contraseña. Para ello los deberá introducir en la pantalla de inicio de sesión. Después de que el usuario haya pulsado en el botón Iniciar Sesión, la aplicación se conectará con el servidor para comprobar que las credenciales de acceso son correctas. En el caso de que lo sean, actualizara los datos del usuario almacenados en la base de datos local y, dependiendo de si esta aparcado previamente o no, mostrará la pantalla de plaza ocupada o de búsqueda de aparcamiento, respectivamente.

Ocupar plaza

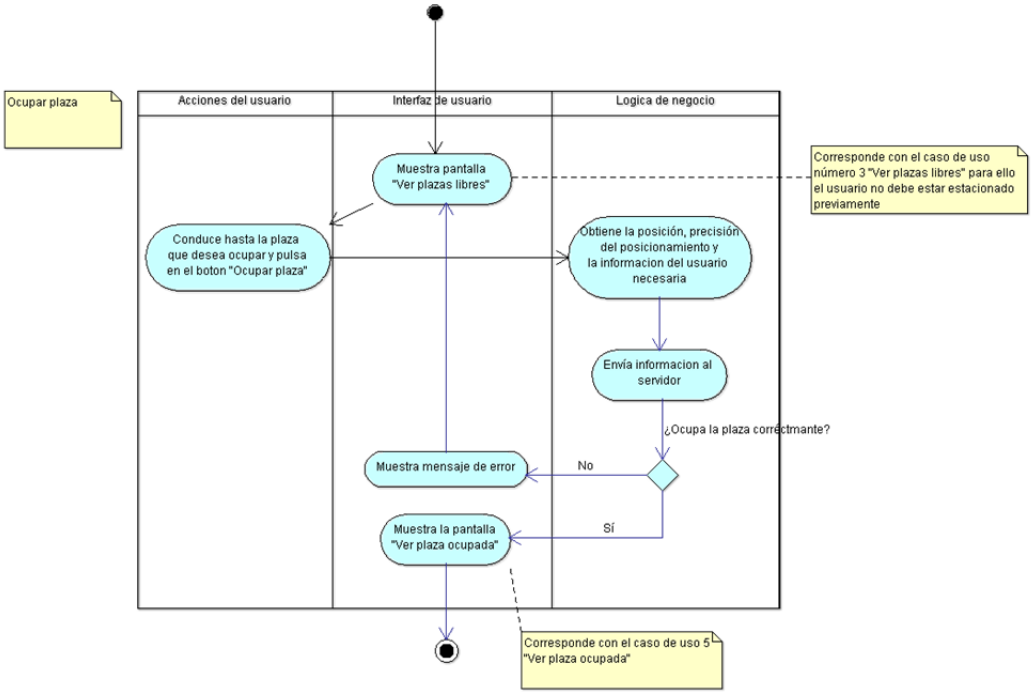


Figura 91: *Diagrama de actividad ocupar plaza*

Una de las principales funcionalidades de esta aplicación es la búsqueda de aparcamiento y ocupar la plaza seleccionada. Para realizar esta última tarea deberemos estar en la pantalla de búsqueda de plazas. Tras esto y haber seleccionado donde quiere estacionar su vehículo, el usuario, conducirá hasta el lugar donde se encuentra la plaza. Una vez estacionado, el usuario pulsará sobre el botón Ocupar plaza, momento en que la aplicación cliente obtendrá los datos necesarios de localización, precisión del GPS y la información necesaria del usuario enviando toda la información recopilada y, de este modo, registrar la plaza que ocupa de una manera más precisa. Si se realiza correctamente la ocupación, se mostrara la pantalla de plaza ocupada y en caso contrario se mostrará un mensaje de error.

Liberar plaza

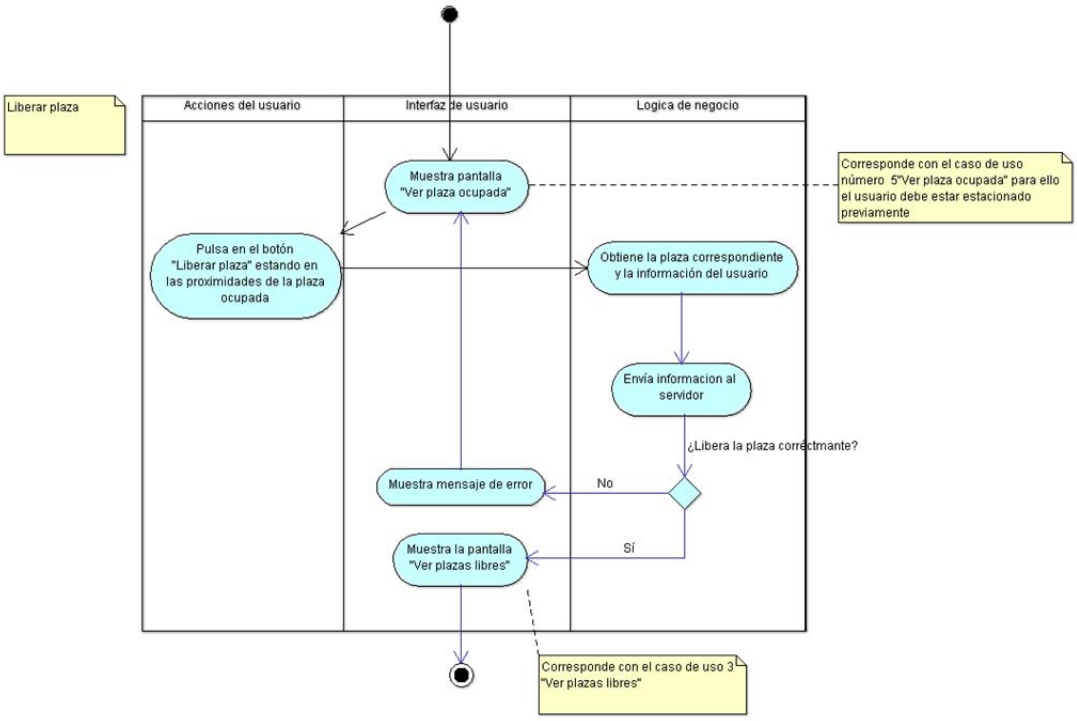


Figura 92: *Diagrama de actividad liberar plaza*

Otra de las funcionalidades principales de nuestra aplicación y que es necesaria para que la aplicación funcione. Para liberar una plaza, el usuario debe haberla ocupado previamente. Por esto, encontrándose en la pantalla Ver plaza ocupada, pulsará sobre el botón Liberar plaza. Una vez pulsado el botón, la aplicación recopilará la información necesaria acerca de la plaza y del usuario enviando la petición correspondiente al servidor. Una vez recibida la respuesta, si la operación se ha realizado correctamente mostrará la pantalla de búsqueda de aparcamiento y en caso de error, no realizará ninguna acción mostrando el error ocurrido.

Abrir/Cerrar menú

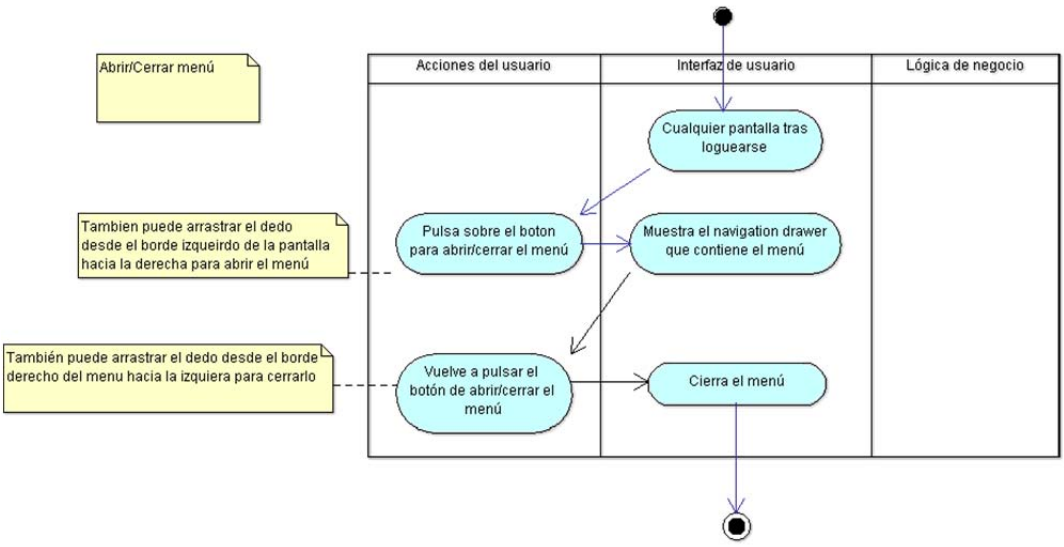


Figura 93: *Diagrama de actividad abrir/cerrar menú*

Para poder acceder a alguna de las funcionalidades como puede ser el perfil se debe abrir el menú de usuario. Por ello, tras haber iniciado sesión correctamente, en cualquiera de las pantallas a las que se puede acceder, se hará una pulsación sobre la parte superior izquierda de la pantalla (zona de `ic_drawer`, icono y título de la aplicación).Del mismo modo, para cerrar el menú, se hará otra pulsación en el mismo lugar. También se puede abrir el menú deslizando el dedo desde el lado izquierdo de la pantalla hacia la derecha y cerrarlo, deslizando el dedo desde el borde derecho del menú, hacia el lado izquierdo.

Ver perfil

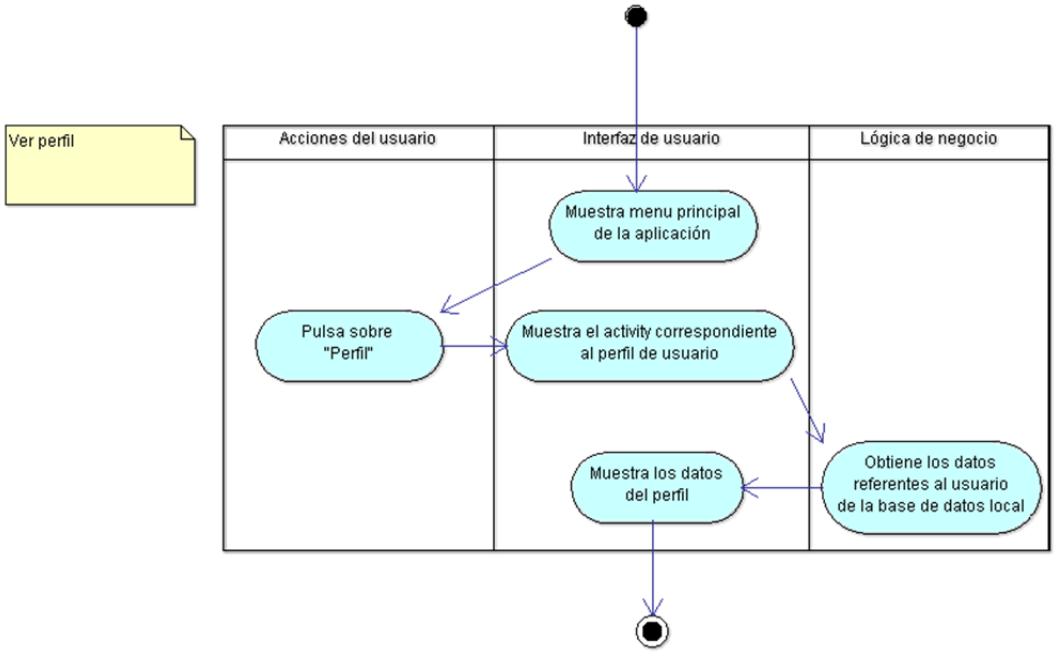


Figura 94: *Diagrama de actividad ver perfil*

El perfil de usuario es la parte en la que podremos consultar todos los datos referentes al usuario activo. Para ello, estando el menú principal de la aplicación abierto, el usuario deberá pulsar sobre el botón Perfil. En este momento se abrirá la actividad correspondiente al perfil y, la aplicación obtendrá los datos del usuario de la base de datos local del terminal, mostrándolos acto seguido en los componentes de la interfaz correspondientes.

Modificar perfil

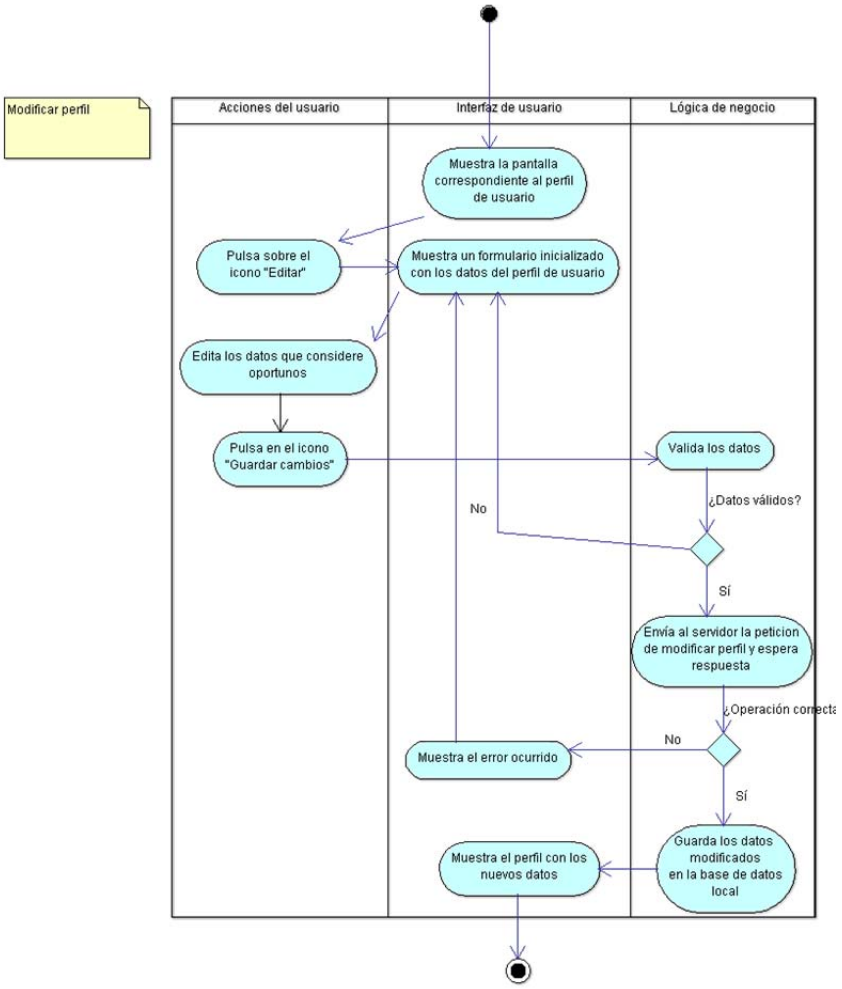


Figura 95: *Diagrama de actividad modificar perfil*

Partiendo de la visualización del perfil de usuario, también se puede editar. Para el usuario deberá pulsar sobre el icono de editar que se encuentra en el *ActionBar* en la parte superior derecha. Una vez pulsado el botón, la aplicación muestra el formulario de modificación con los campos inicializados según los datos actuales del perfil. Tras editar los campos deseados, el usuario pulsara sobre el botón de guardar, también contenido en el *ActionBar*, momento en el que la aplicación validará los datos y en caso de que sea todo correcto, los envía al servidor para actualizar el perfil en la base de datos central del sistema. Acto seguido espera la respuesta del servidor, y si esta es que se ha realizado la operación correctamente, actualizará la información del usuario en la base de datos local y mostrará el perfil con los datos actualizados, en caso contrario, mostrara el mensaje del error ocurrido y volverá a la pantalla de edición del perfil.

Ver aparcamientos recientes

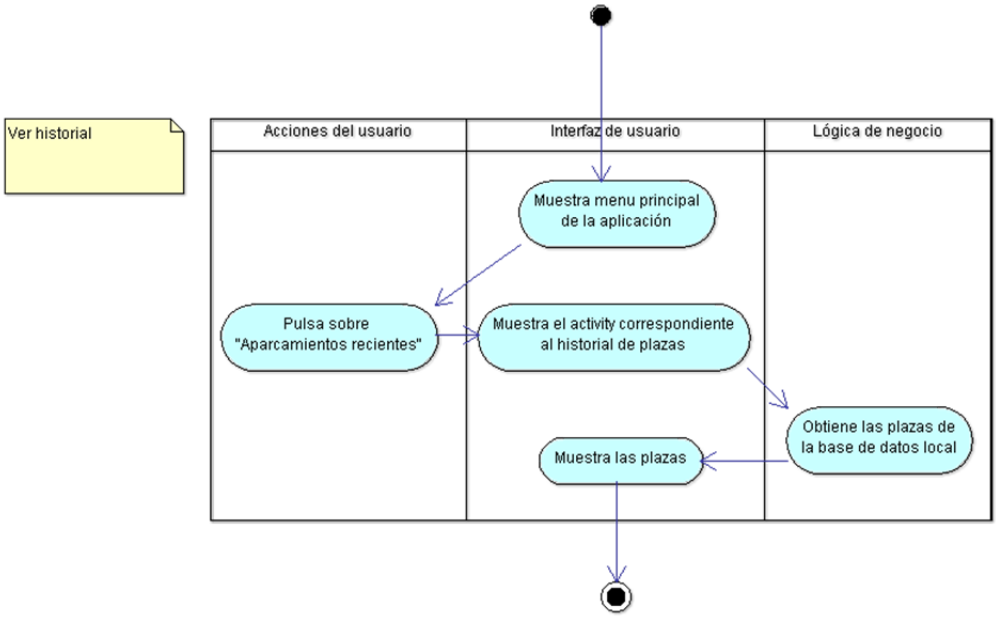


Figura 96: *Diagrama de actividad ver aparcamientos recientes*

Al igual que en los dos casos anteriores, deberá estar el menú principal abierto para acceder al historial de plazas. Para ello el usuario pulsará en la entrada del menú llamada Aparcamientos recientes. Momento en el que se abrirá el *activity* que muestra dicho historial y, tras esto, la aplicación obtiene las plazas ocupadas recientemente de la base de datos local y las dibuja en el mapa.

Ver detalles plaza

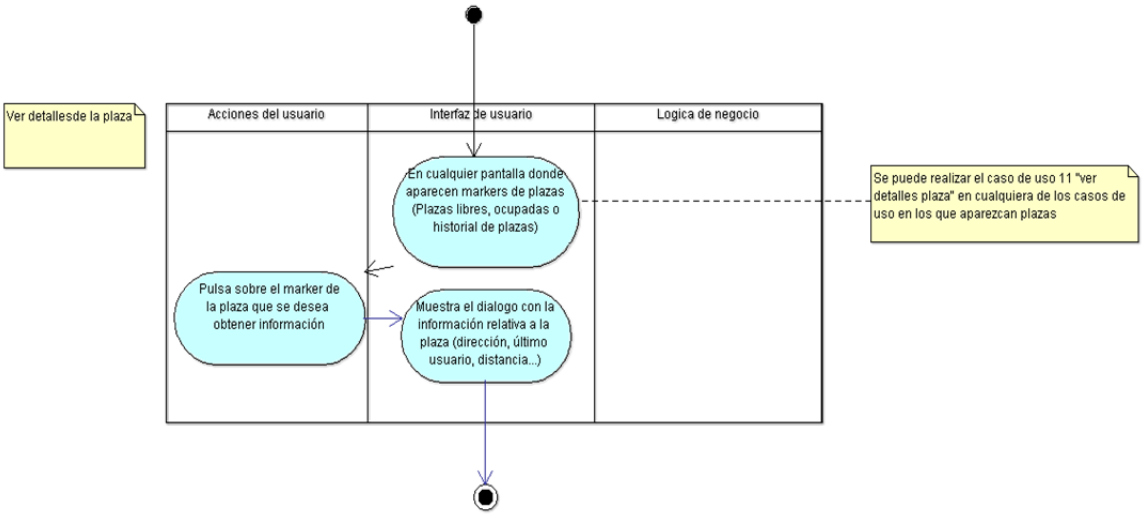


Figura 97: *Diagrama de actividad ver detalles plaza*

El usuario podrá visualizar los detalles de una plaza en cualquiera de las pantallas en las que se muestre un mapa con plazas. Además también podrá ver los detalles de su posición del mismo modo. Para ello deberá pulsar sobre la plaza deseada, abriéndose entonces un dialogo que contendrá toda la información de interés acerca de la plaza (dirección, distancia, último usuario que la utilizó..) y de su posición (dirección, precisión del GPS).

Abrir navegador

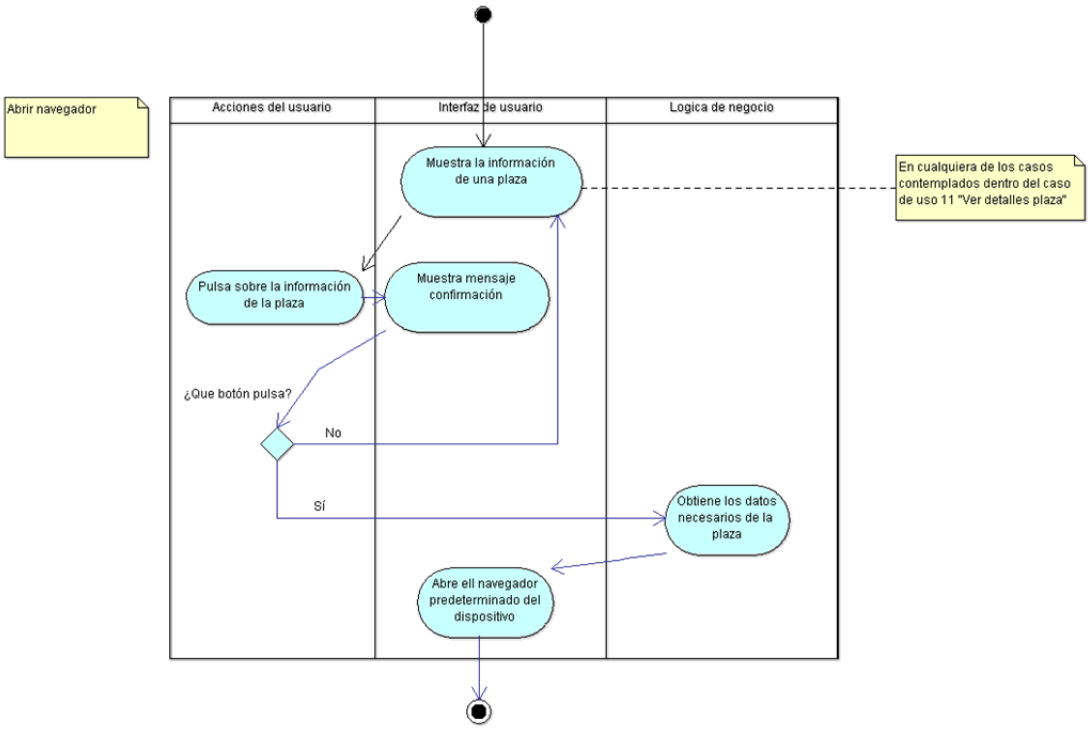


Figura 98: *Diagrama de actividad abrir navegador*

Esta opción es muy útil cuando el usuario no conoce la localidad o la zona dónde está buscando estacionamiento. Para poder abrir el navegador, el usuario deberá consultar los detalles de una plaza como se ha explicado en el apartado anterior. Partiendo de este punto deberá pulsar sobre los detalles, apareciéndole un mensaje de confirmación, que si pulsa si abrirá el navegador y en caso que pulse sobre no, seguirá mostrando la información de la plaza seleccionada.

Activar datos móviles

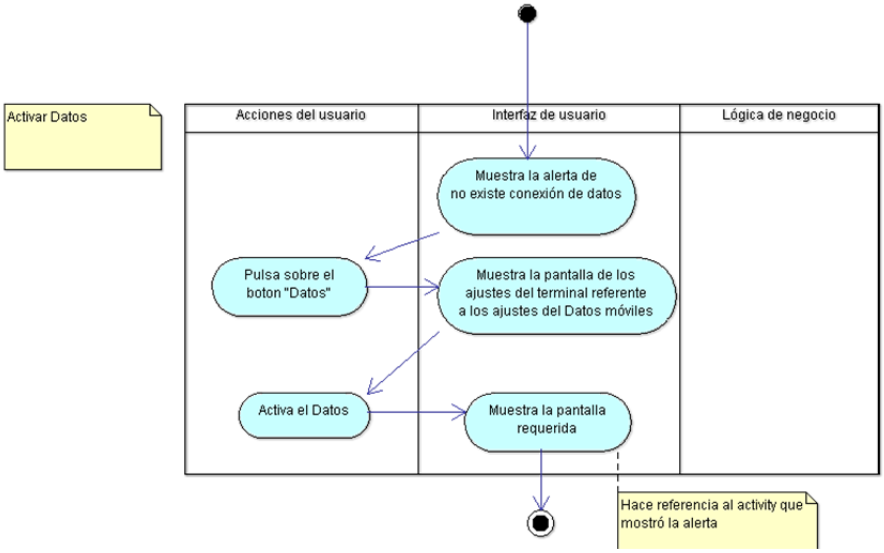


Figura 99: *Diagrama de actividad activar datos móviles*

Para hacer uso de la aplicación AparcaDroid, se necesita conexión a Internet, por ello, cuando no dispongamos de ella se mostrará un mensaje de alerta. En este mensaje de alerta, cuando el usuario pulse sobre el botón Datos, se abrirá la pantalla de los ajustes del terminal que corresponde a los ajustes de los datos móviles. Tras activar la opción de datos móviles, el usuario pulsará el botón atrás del terminal y en este momento podrá seguir interactuando con la pantalla que lo hacía en el momento que se le mostró el dialogo de alerta.

Activar Wi-Fi

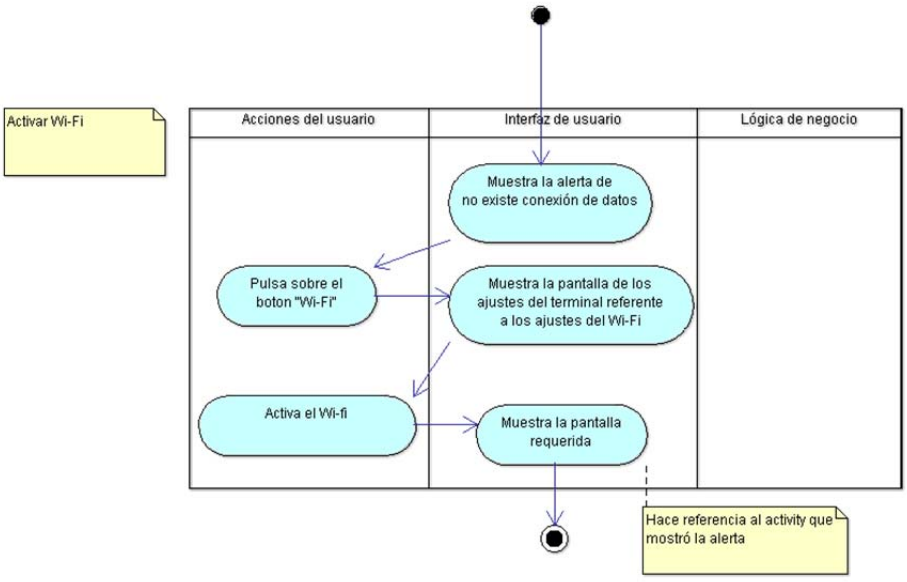


Figura 100: *Diagrama de actividad activar Wi-Fi*

Como se indica en el caso anterior, la aplicación AparcaDroid cliente necesita de conexión a Internet. Si no disponemos de una tarifa plana de datos o simplemente no deseamos gastarlos en esto, también se permite el uso de la aplicación a través de una conexión Wi-Fi. Para ello, cuando aparezca el diálogo de alerta, el usuario deberá pulsar sobre el botón Wi-Fi que nos abrirá sus ajustes del sistema y una vez activado pulsando el botón atrás del terminal, ya podrá interactuar con la actividad con la cual lo hacía en el momento que apareció el mensaje de alerta.

Activar GPS

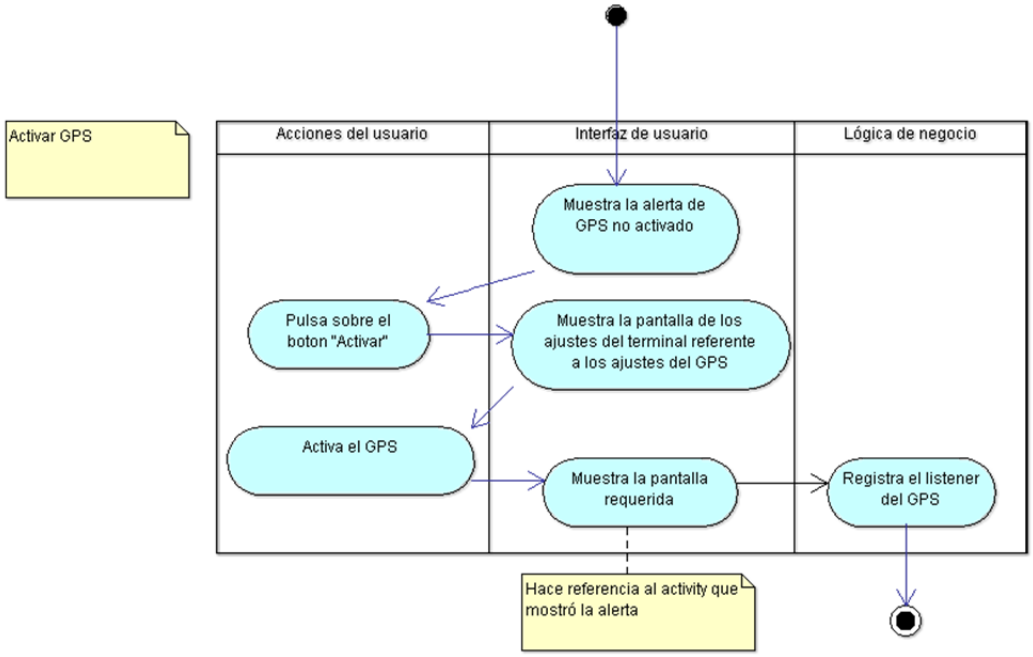


Figura 101: *Diagrama de actividad activar GPS*

En el momento que nos logueamos, es necesaria la conexión del GPS en las pantallas que se muestran mapas, para obtener nuestra posición y así realizar las operaciones necesarias. En caso de que el GPS no este activado, se mostrará una alerta y cuando el usuario pulse sobre el botón Activar, se abrirá la pantalla de configuración del sistema que corresponde al apartado del GPS. Una vez en esta pantalla el usuario deberá activar el GPS y pulsar el botón Atrás del terminal. En este momento, la aplicación mostrará la pantalla requerida y registrará el *listener* del GPS.

Salir

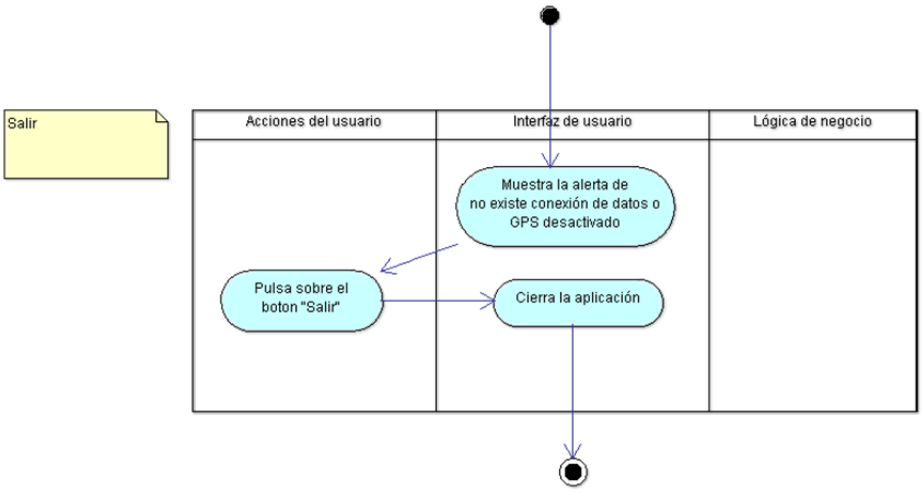


Figura 102: *Diagrama de actividad salir*

En cualquiera de los mensajes de alerta que muestra la aplicación para solicitar la activación de ciertos recursos (GPS, Datos, Wi-Fi), esta opción permitirá salir de la aplicación si no deseamos hacer uso de estos recursos. Para ello pulsaremos en el botón Salir del mensaje de alerta y, entonces, la aplicación se cerrará.

Cerrar Sesión

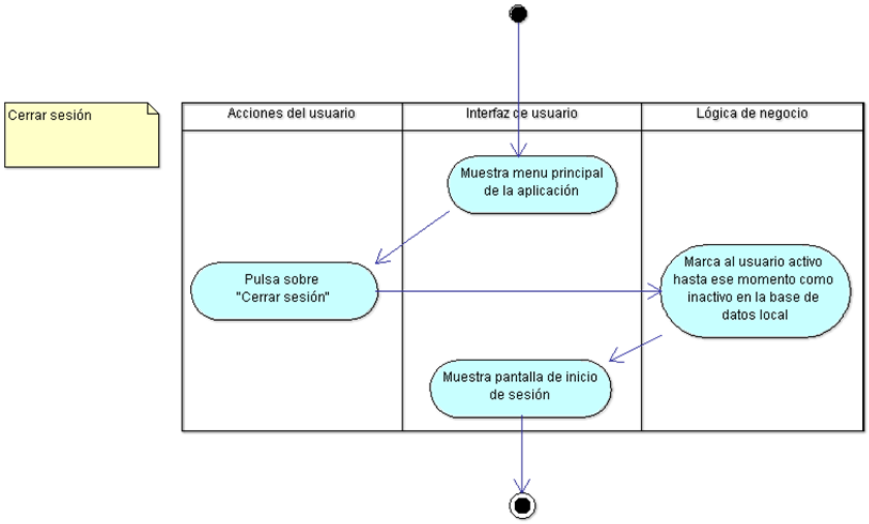


Figura 103: *Diagrama de actividad cerrar sesión*

Cuando el usuario lo desee y estando abierto el menú principal de la aplicación, un usuario podrá cerrar su sesión. Para ello deberá pulsar sobre el botón del menú que con el nombre Cerrar sesión. Una vez hecho esto la aplicación marcará al usuario activo como inactivo en la base de datos local, y tras esto, cerrara la pantalla actual y mostrará la pantalla de inicio de sesión.

6.5.Diagramas de secuencia

En el presente apartado se plasman y describen los diagramas de secuencia, diferenciando los de la aplicación servidor y los de la aplicación AparcaDroid cliente.

6.5.1. Servidor

Registrar usuario

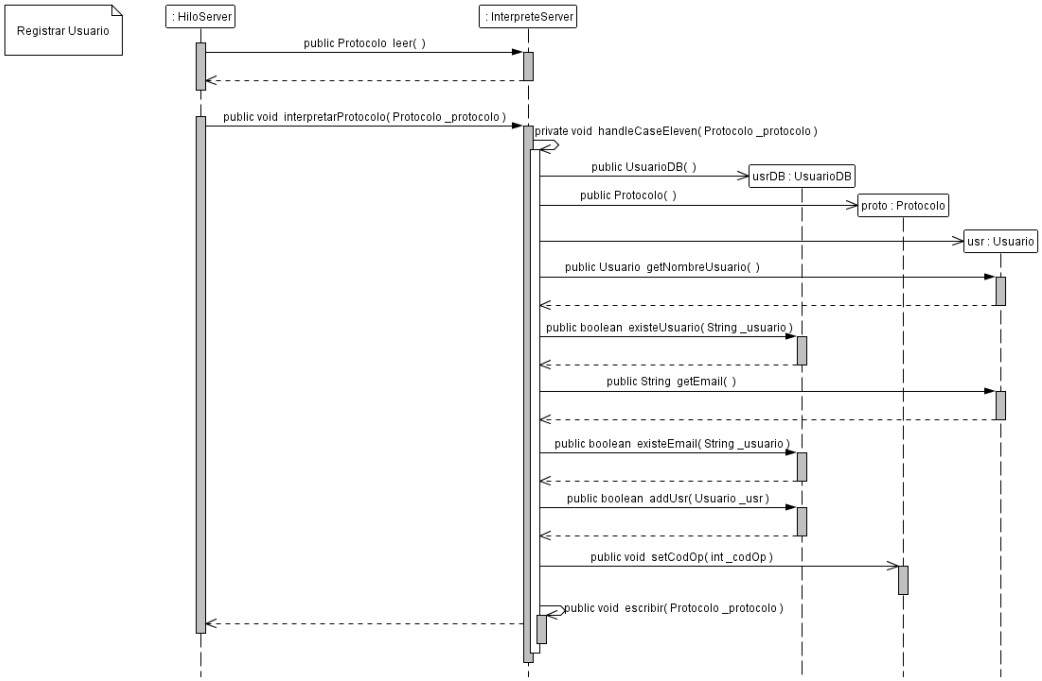


Figura 104: *Diagrama de secuencia registrar usuario*

El registro de usuario es la primera secuencia de operaciones que deberá realizar un usuario para poder empezar a utilizar la aplicación. El diagrama anterior muestra los métodos de las clases que se llaman para ello cuando se efectúa correctamente. Solamente destacar que cuando un usuario se registra, se añaden los datos correspondientes en la base de datos de la aplicación.

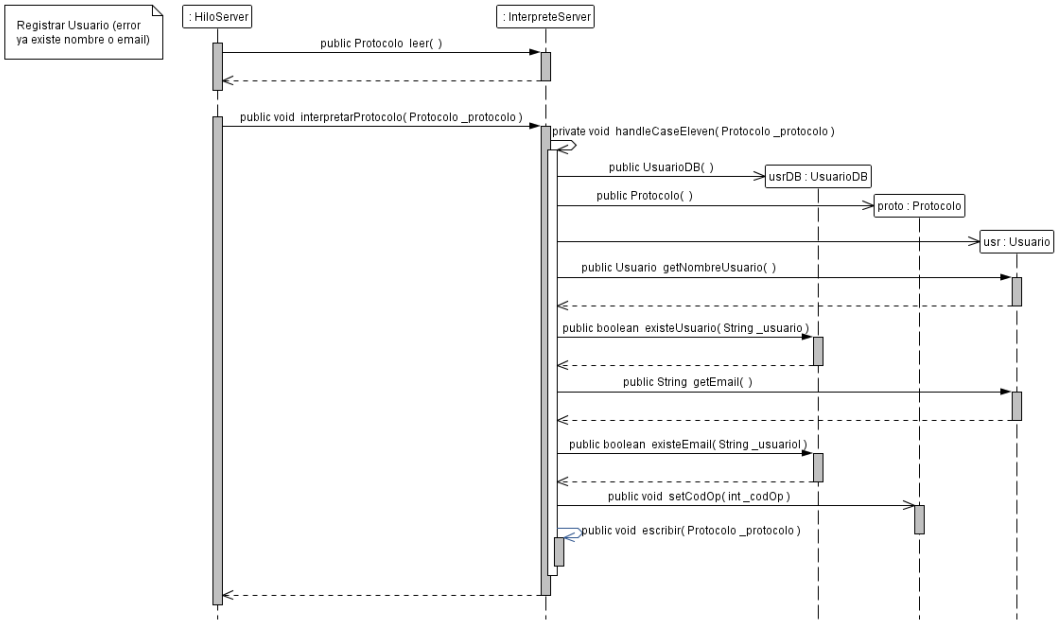


Figura 105: *Diagrama de secuencia registrar usuario erróneamente*

Similar al primero, este diagrama muestra los pasos que realiza la aplicación cuando se produce un error en el registro de un usuario.

Autenticarse en el sistema

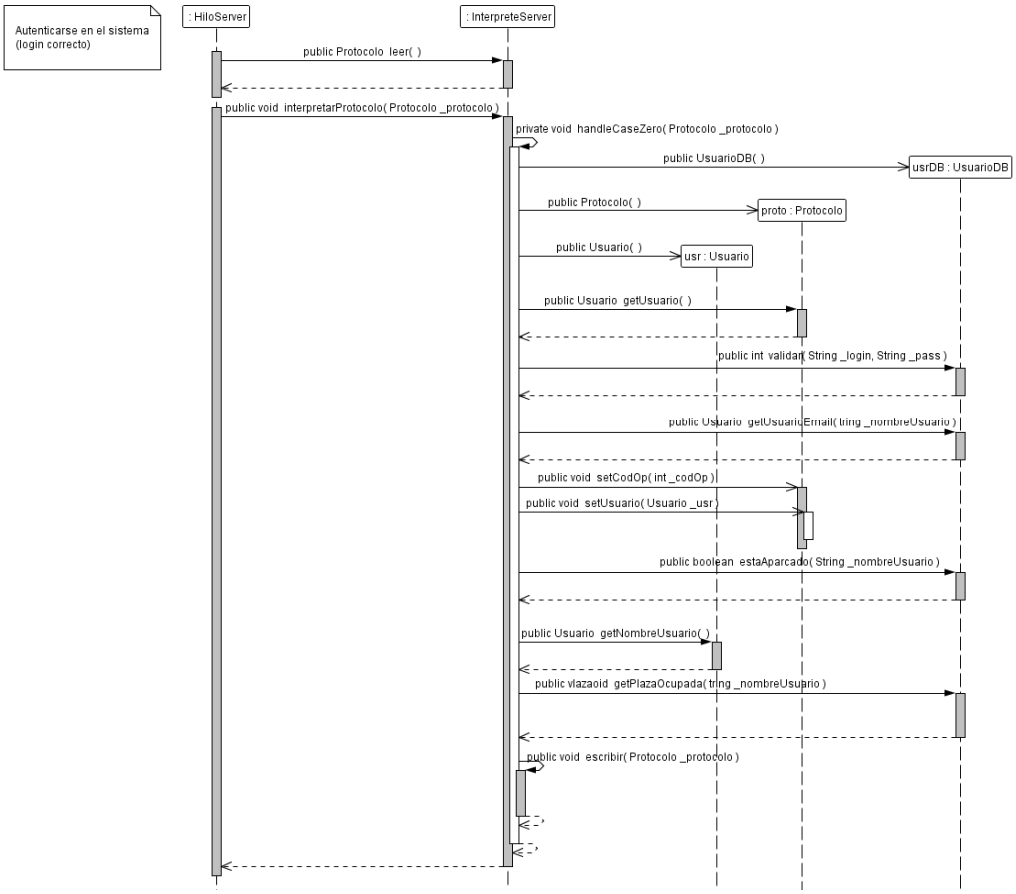


Figura 106: *Diagrama de secuencia autenticarse en el sistema*

El diagrama anterior muestra los métodos que se invocan en la aplicación servidor cuando recibe una petición de login y este se efectúa correctamente. Dicho login es el paso necesario, una vez registrado, para poder utilizar los servicios que ofrece el sistema. Destacar, que una vez validados los datos, si el usuario ocupa una plaza se obtendrá dicha plaza y se adjuntara al mensaje de respuesta.

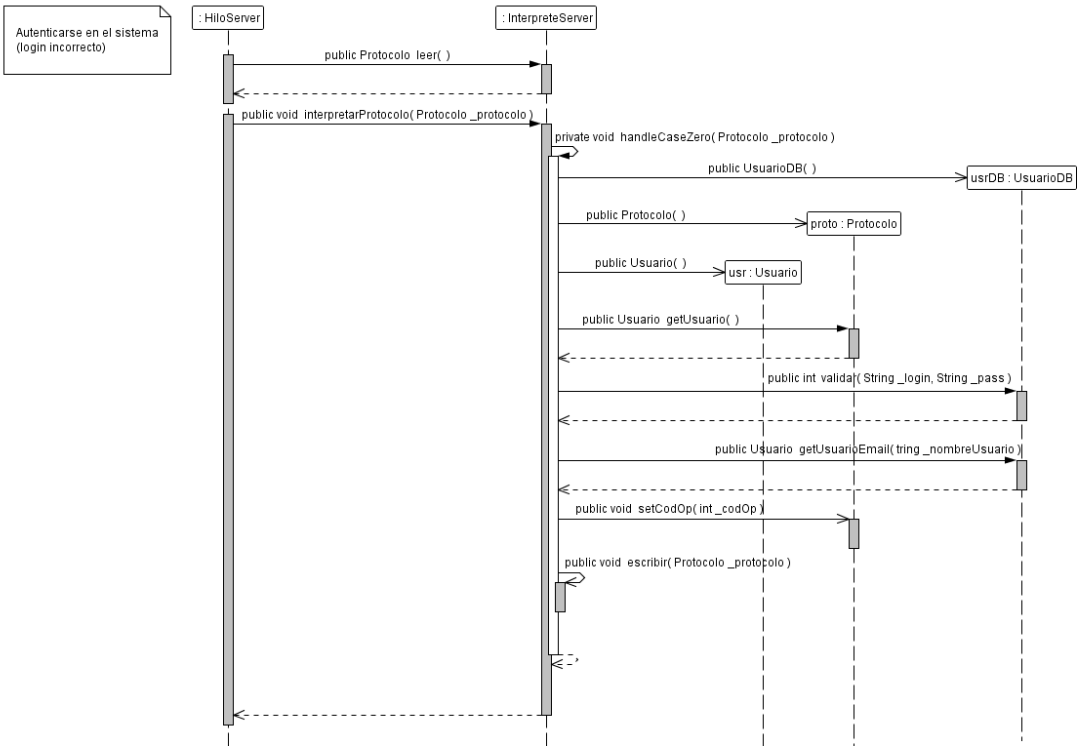


Figura 107: *Diagrama de secuencia autenticarse en el sistema erróneamente*

En el diagrama anterior, se muestra el proceso de *login* pero en el caso de que se produzca un error. En este caso, será causado porque o bien el email, o bien la contraseña asociada a ese email son incorrectos.

Ver plazas libres

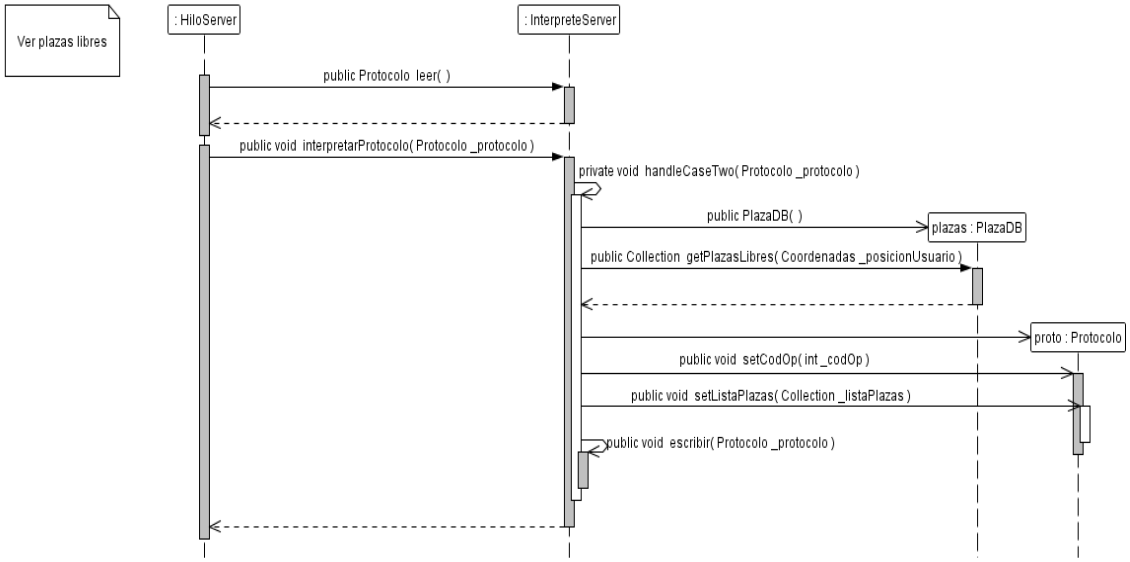


Figura 108: *Diagrama de secuencia ver plazas libres*

En el diagrama anterior se muestran las acciones que realiza el servidor cuando recibe una solicitud de obtención de las plazas libres. Para ello, primeramente, valida los datos y a partir de las coordenadas indicadas (corresponden con la posición del usuario que solicita), obtiene de la base de datos las plazas que se encuentran libres a su alrededor.

Ocupar plaza

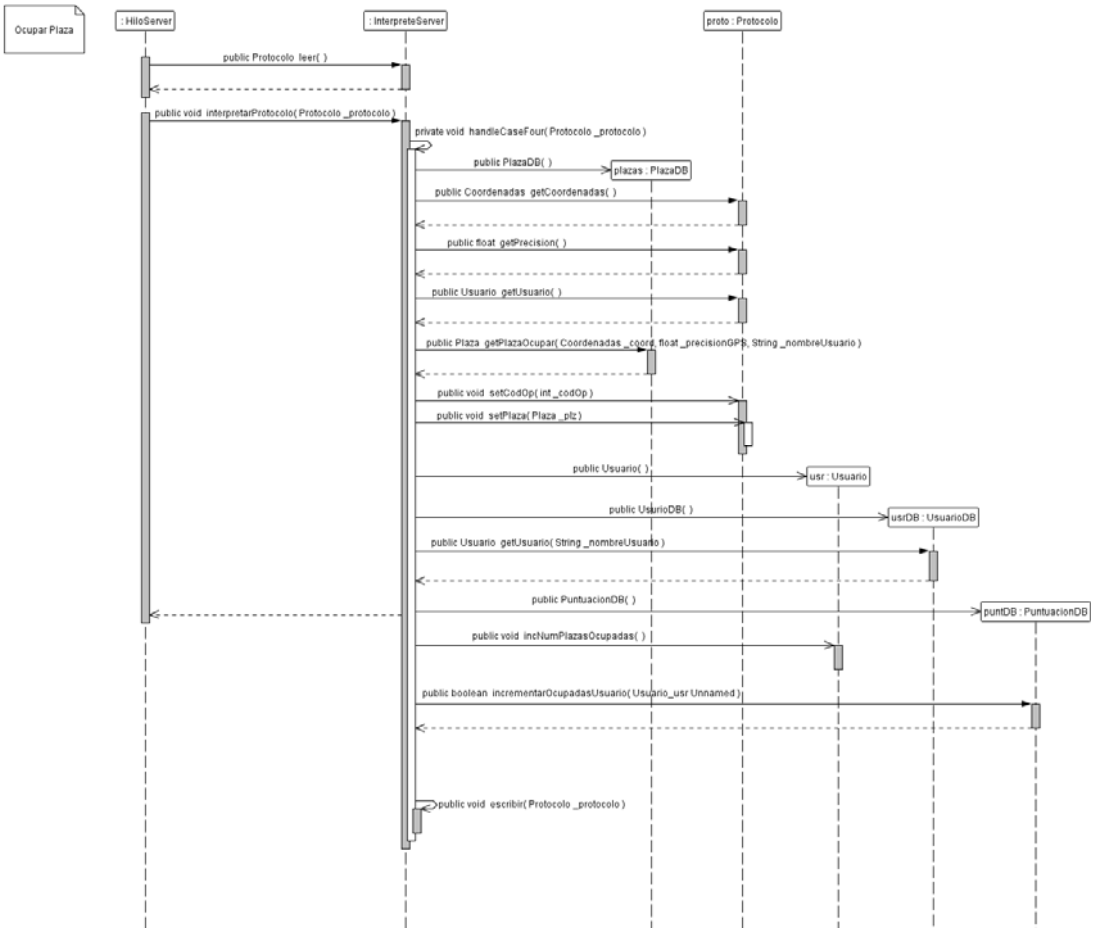


Figura 109: *Diagrama de secuencia ocupar plaza*

Este diagrama muestra los métodos que ejecuta la aplicación servidor a la hora de procesar una solicitud de ocupación de plaza. Para ello obtiene la posición del usuario y la precisión del GPS del dispositivo cliente. Una vez hecho esto determinará qué plaza se debe ocupar o, en caso de que no exista ninguna registrada en esa posición, creará una nueva. Por último, establecerá como ocupada la plaza elegida y, actualizara los datos o los insertará en la base de datos dependiendo de si es una plaza existente o una nueva, respectivamente.

Obtener plaza Ocupada

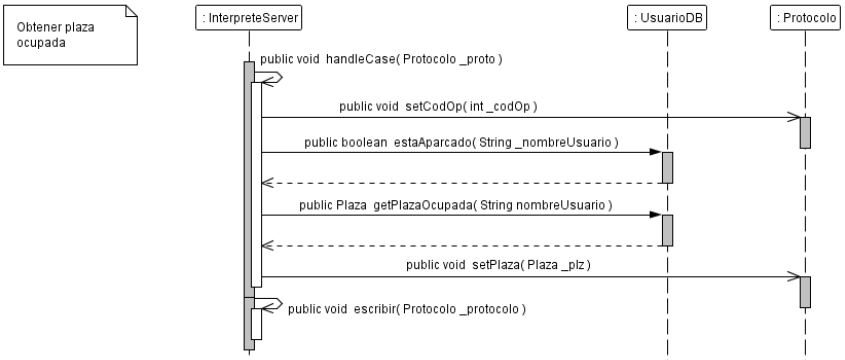


Figura 110: *Diagrama de secuencia obtener plaza ocupada*

En el diagrama de este apartado se muestra la secuencia de métodos que ejecuta el servidor cuando en alguna de las operaciones que se le solicitan se debe mostrar la plaza que ocupa un usuario. Para ello, primeramente comprueba si está aparcado y, en caso de que lo esté, obtiene la plaza que ocupa de la base de datos.

Liberar plaza

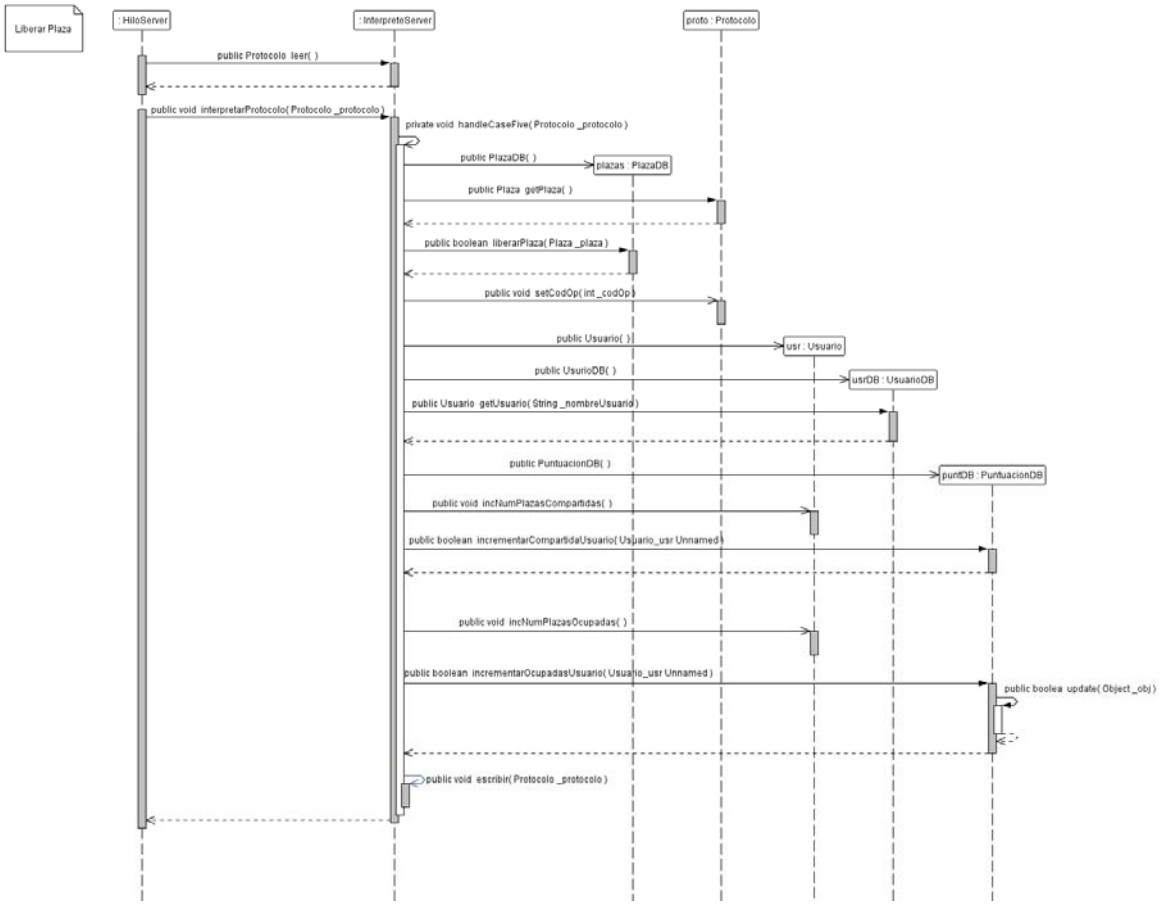


Figura 111: *Diagrama de secuencia liberar plaza*

El presente diagrama muestra los métodos que ejecuta el servidor cuando recibe una petición para liberar una determinada plaza. Para ello obtiene de la base de datos la plaza que ocupaba el usuario y la libera, incrementando después el número de plazas compartidas de dicho usuario y lo guarda en la base de datos.

Modificar perfil

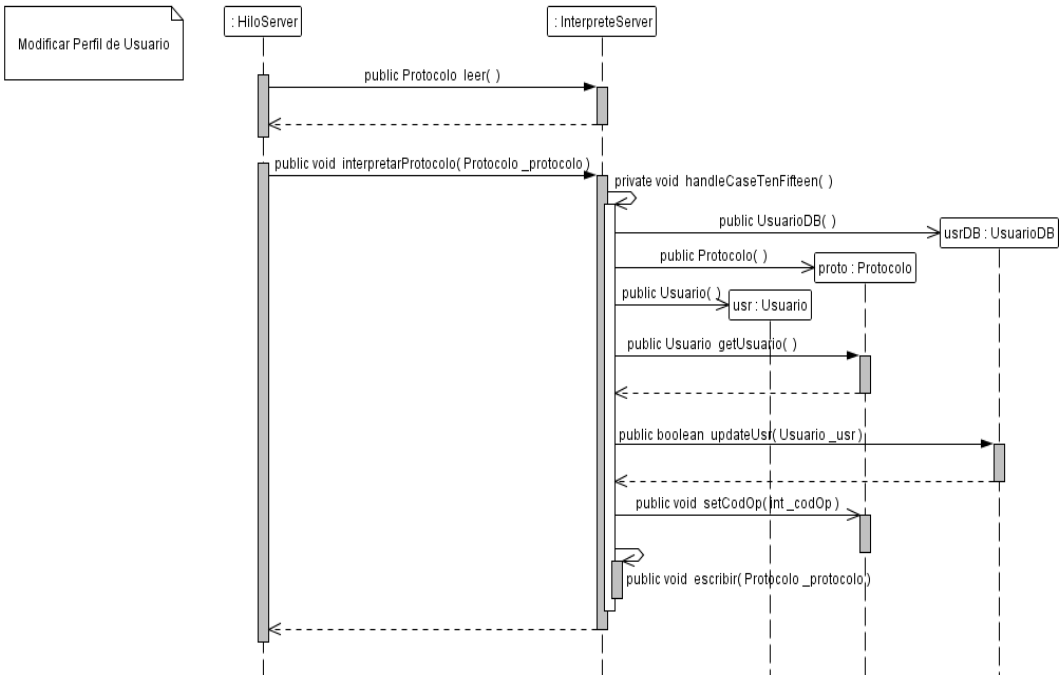


Figura 112: *Diagrama de secuencia modificar perfil*

En el diagrama anterior se plasman los métodos que ejecuta el servidor de nuestra aplicación cuando recibe una petición para modificar los datos del perfil de un usuario. Para ello obtendrá los nuevos datos del protocolo recibido y posteriormente, actualiza los datos almacenados en la base de datos con los nuevos obtenidos.

Conectar

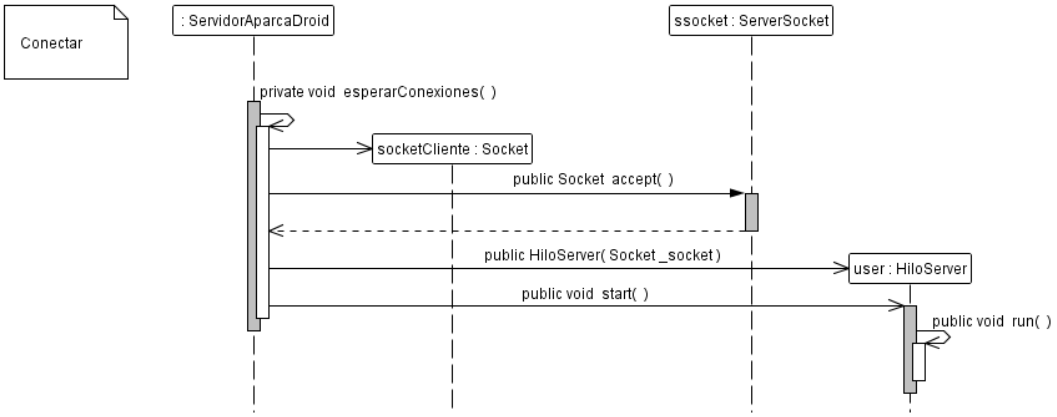


Figura 113: *Diagrama de secuencia conectar*

El diagrama anterior muestra los métodos que se invocan cuando el servidor recibe una petición de conexión. Cada vez que una aplicación cliente se conecte con el servidor, este creará una nueva instancia de la clase `HiloServer` para gestionar la petición. Al ejecutar el método `start()` de dicha clase, hará que se ejecute en un *thread* diferente al principal.

Desconectar

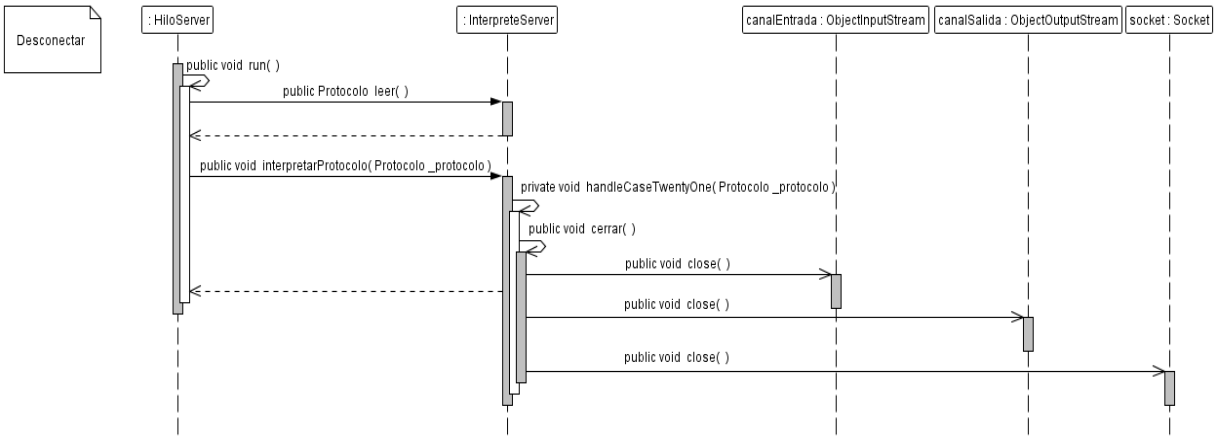


Figura 114: *Diagrama de secuencia desconectar*

Cuando se recibe el mensaje de desconexión, se procederá a finalizar la comunicación entre la aplicación cliente y la aplicación servidor. Esto se hace se cerrando los `Streams` de entrada y salida creados a partir del socket y, a continuación se cerrando el propio *socket*.

Leer

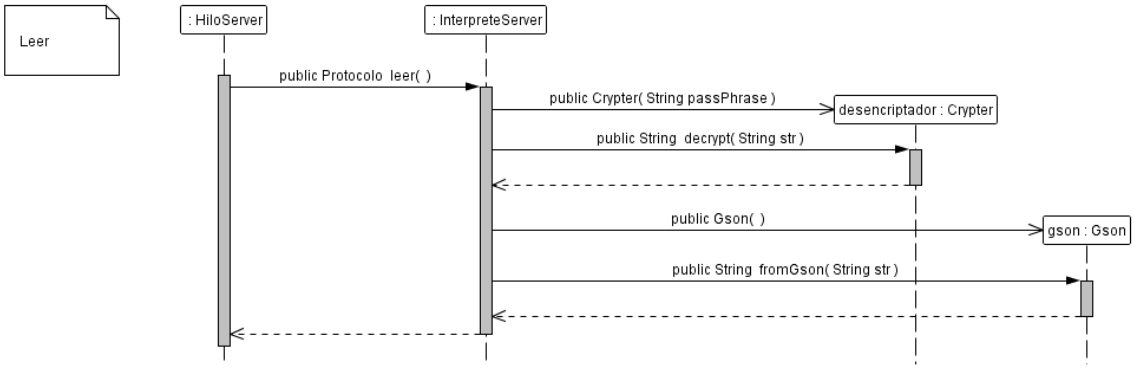


Figura 115: *Diagrama de secuencia leer*

En el diagrama anterior, se ha detallado, las llamadas a métodos que realiza la aplicación servidor cuando lee del canal. Esta funcionalidad se ha detallado en un diagrama aparte para no sobrecargar el resto de diagramas en los que aparece el método `public Protocolo leer()`.

Escribir

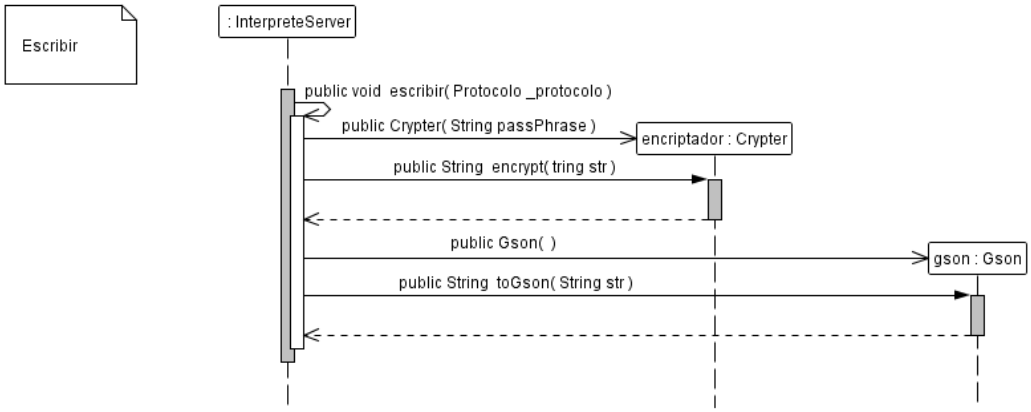


Figura 116: *Diagrama de secuencia escribir*

En este diagrama se muestra los métodos que ejecuta la aplicación servidor cuando realiza la escritura en el canal de comunicación. Al igual que en el caso anterior, se ha decidido detallar los pasos de la ejecución del método `public void escribir(Protocolo _proto)` en un diagrama aparte, para no saturar el resto de diagramas en los que aparece este método.

Mantenimiento plazas

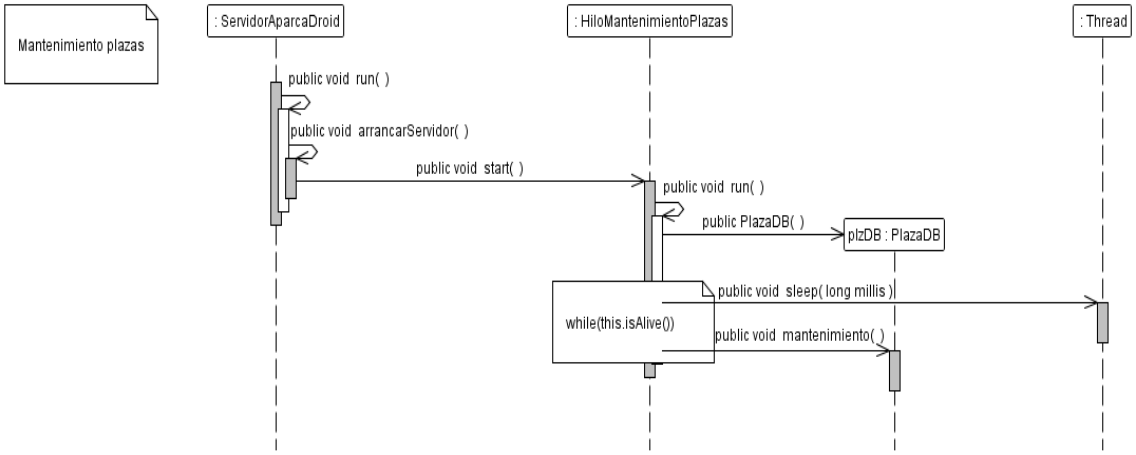


Figura 117: *Diagrama de secuencia mantenimiento plazas*

Este diagrama muestra los métodos que ejecuta la aplicación servidor desde que inicia el hilo de mantenimiento de plazas, hasta que realiza dicho mantenimiento de plazas.

6.5.2. Cliente

Registrar usuario

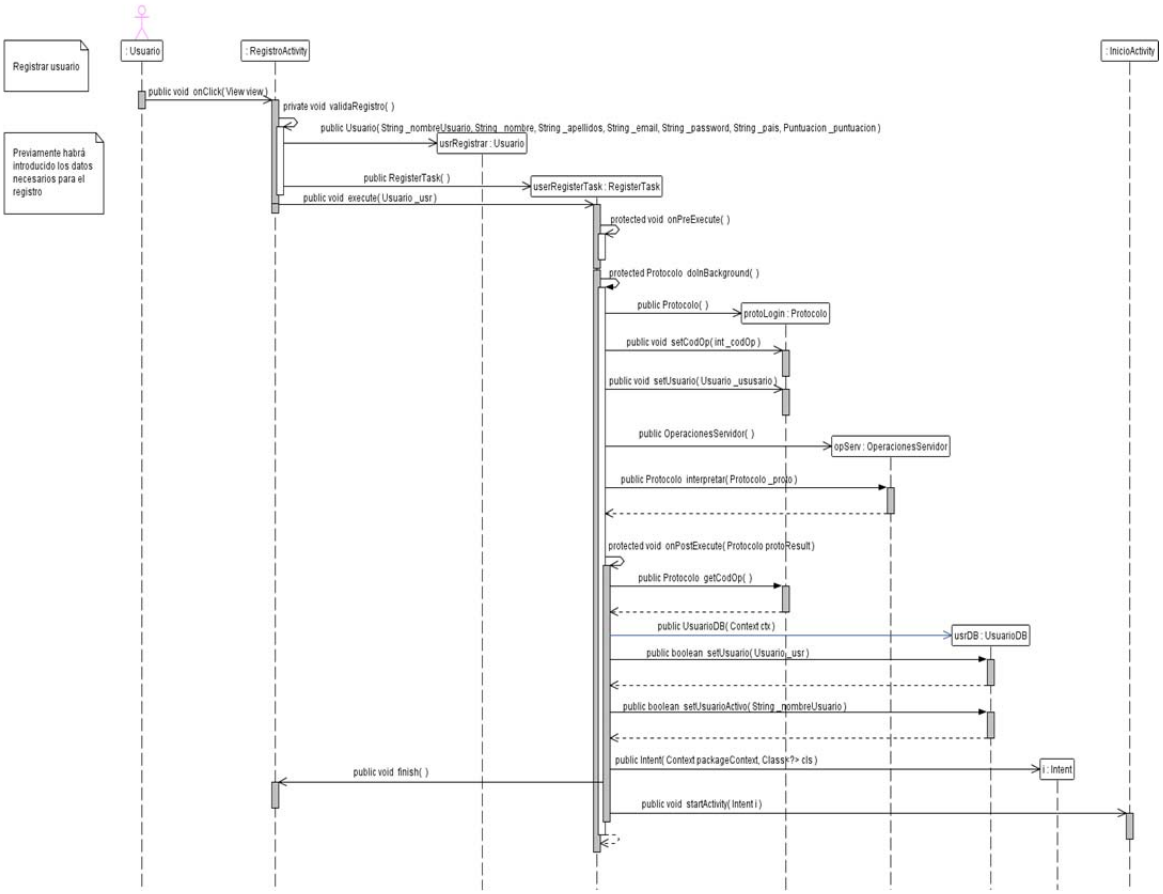


Figura 118: *Diagrama de secuencia registrar usuario*

Registrarse es la primera acción que debe realizar el usuario antes de poder iniciar sesión para poder interactuar con la aplicación. En este diagrama se muestran los métodos y otras acciones que ejecuta la aplicación cliente AparcaDroid a la hora de registrar un usuario.

Primeramente el usuario deberá introducir los datos requeridos para el registro y, tras esto, pulsar sobre el botón Terminado. Dicha pulsación será capturada por el método `onClick(View view)` del *Activity RegistroActivity*, en el cual se ejecutara el método que valida los datos y se creará una instancia de Usuario que, posteriormente, será enviada para efectuar el registro. Tras esto instancia y ejecuta el `AsyncTask` de esta clase, cuya instancia se llama `userRegisterTask` y la clase `RegisterTask`. En su método `doInBackground()` realiza la comunicación con el servidor, enviando la instancia de Protocolo con el código de operación adecuado y la instancia del usuario registrar. Por último, en el método `public void onPostExecute(Protocolo protocolo)`, tras recibir la respuesta del servidor y comprobar que se ha realizado el registro correctamente, guarda el usuario en la base de datos local, lo marcará como

usuario activo y abrirá el *Activity InicioActivity* finalizando el actual RegistroActivity.

Autenticarse en el sistema

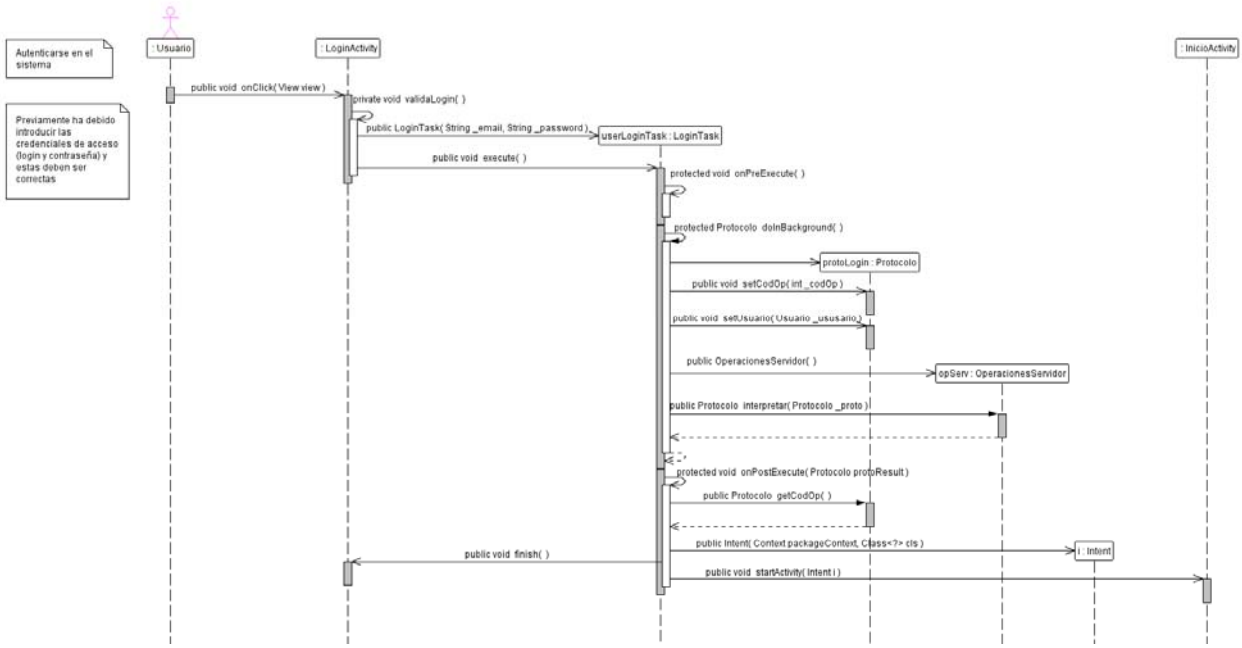


Figura 119: *Diagrama de secuencia autenticarse en el sistema.*

Autenticarse en el sistema es el paso siguiente después de registrarse, esto permitirá usar todas las funcionalidades de la aplicación. En el diagrama anterior se detallan los pasos, acciones y métodos que ejecuta la aplicación cliente a la hora de autenticar a un usuario. Para la realización de este diagrama se ha considerado que el usuario ya ha introducido las credenciales de acceso y, por eso, se muestra solo desde el momento en que dicho usuario pulsa sobre el botón Iniciar sesión.

La pulsación sobre el botón Iniciar sesión es capturada por el `metodo onClick(View view)` de la clase *LoginActivity*. Tras ello ejecuta el método `validaLogin()` que comprueba que los campos necesarios se hayan introducido y crea la instancia `userLoginTask` de la clase LoginTask. En el método `onPreExecute()` se hará visible el panel con la barra de progreso. Posteriormente en el método `protected Protocolo doInBackground()` se creará la instancia de la clase Protocolo que se va a enviar, la cual contendrá el código de operación correspondiente y los datos del usuario. Por último, tras recibir la contestación del servidor se ejecuta el método `onPostExecute(Protocolo protoResult)`, que comprueba que la operación se haya realizado correctamente y realiza las operaciones pertinentes. En el caso de que sea correcta, crea una instancia de `Intent` con los datos del *Activity InicioActivity* que posteriormente se cargará, finaliza el actual e inicia el anteriormente citado.

Ver plazas libres

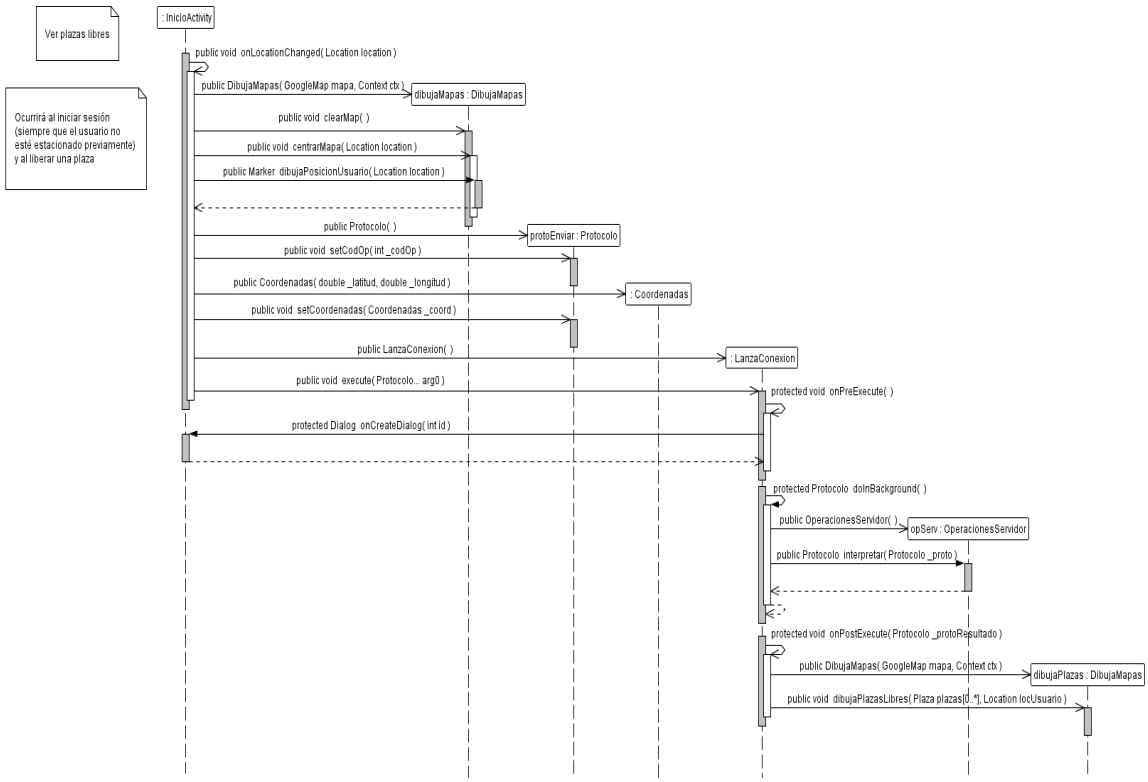


Figura 120: *Diagrama de secuencia ver plazas libres*

Un usuario podrá ver las plazas libres cuando inicie sesión, siempre y cuando no este estacionado ya y, además, cuando libere una plaza. En el diagrama anterior se muestran los métodos que ejecuta la aplicación cliente para obtener las plazas libres alrededor del usuario.

Este proceso empieza cuando se obtiene la primera localización del usuario, momento en que la aplicación comienza a recopilar los datos necesarios y a efectuar la petición de las plazas libres al servidor. Cuando recibe la respuesta, en el caso de que existan dibujará las plazas en el mapa donde ya se había dibujado la posición del usuario.

Ocupar plaza

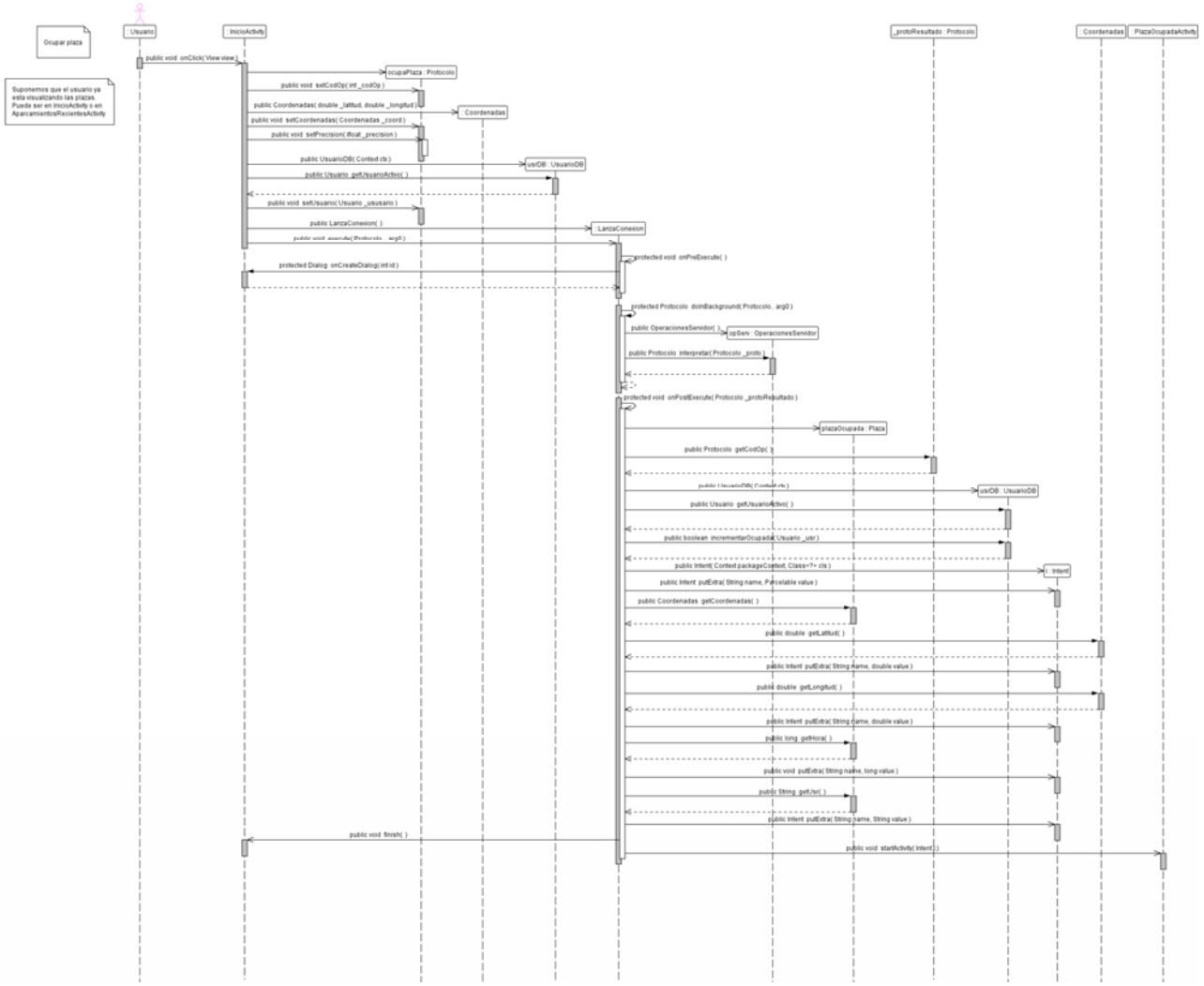


Figura 121: *Diagrama de secuencia ocupar plaza*

Estando en la pantalla de visualizar las plazas libres o en la sección Aparcamientos recientes el usuario puede ocupar una plaza siempre que desee pulsando el botón Ocupar plaza. Posteriormente la aplicación recopilará la información necesaria acerca de la localización del usuario, la precisión del GPS y del propio usuario, mandando todo esto al servidor, junto con la petición de ocupación de plaza. Tras recibir la respuesta, la cual contiene la plaza a ocupar, la aplicación modificará la puntuación del usuario (incrementa número de plazas ocupadas).

Como paso previo antes de cargar el nuevo *Activity*, creara un `Intent` y le agregará los datos que desea compartir con la nueva *Activity* que se cargará y esto lo hace mediante el método `putExtra()`. Por último, finaliza el *Activity* que se está visualizando actualmente mediante el método `finish()` y carga el nuevo ejecutando el método `startActivity(Intent i)`.

Ver plaza ocupada

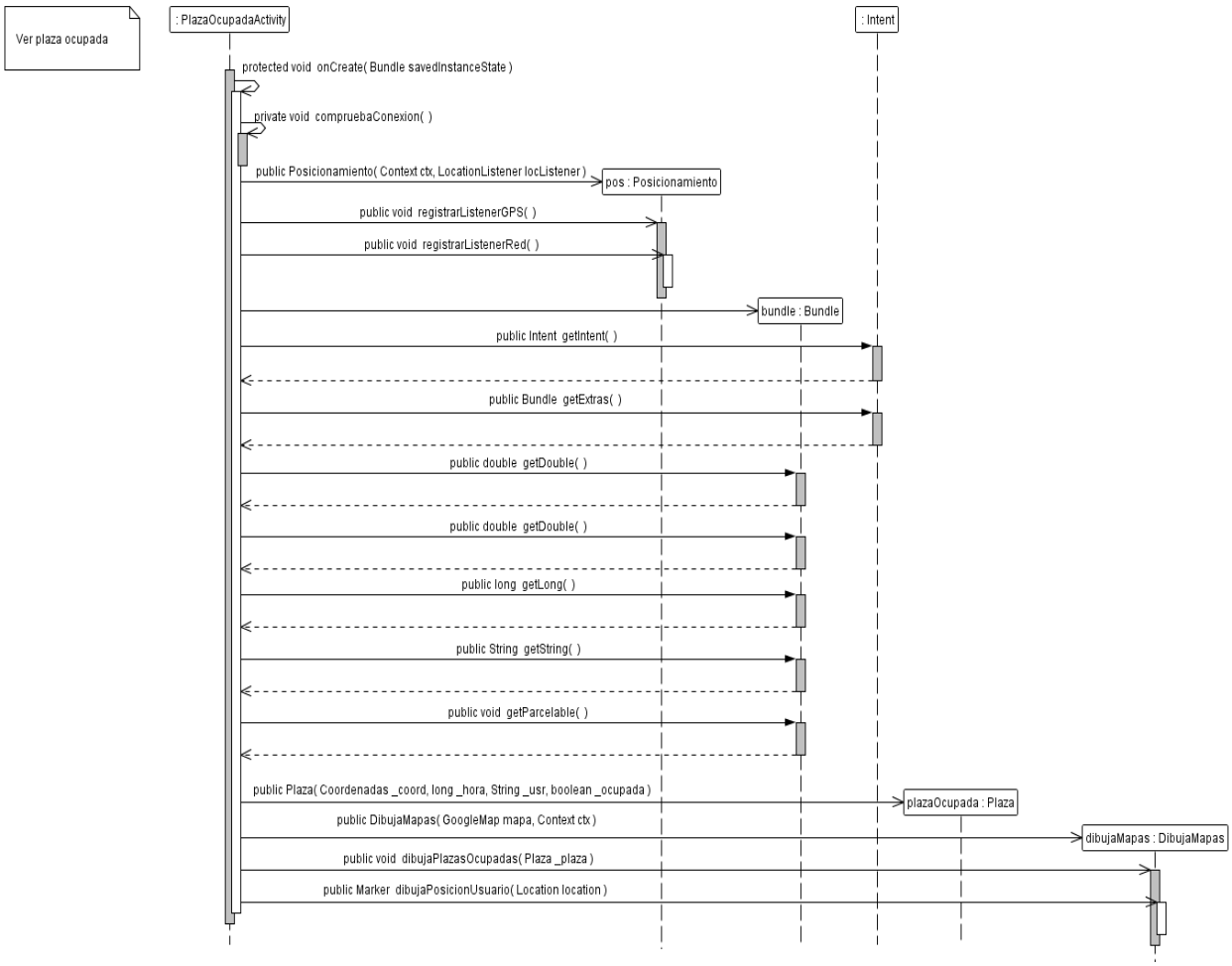


Figura 122: *Diagrama de secuencia ver plaza ocupada*

Para poder visualizar la plaza que ocupa, el usuario deberá realizar las acciones expuestas en el apartado *OcuparPlaza*, o estar previamente estacionado cuando inicia sesión en la aplicación.

Para mostrarle la plaza que ocupa al usuario, el *Activity* *PlazaOcupadaActivity*, en su método `onCreate()`, obtiene el `Intent` que contiene la información que le ha pasado el *Activity* anterior. Para obtener los datos del `Intent`, primeramente obtendrá un `Bundle` mediante el `metodo getExtras()` y posteriormente ejecutara los métodos `getDouble()`, `getLong()`, `getString()` y `getParcelable()`. A continuación, con estos datos obtenidos del `Bundle`, crea una instancia de la clase Plaza. Por último dibujará la plaza en el mapa y lo centrará en la localización de dicha plaza.

Liberar plaza

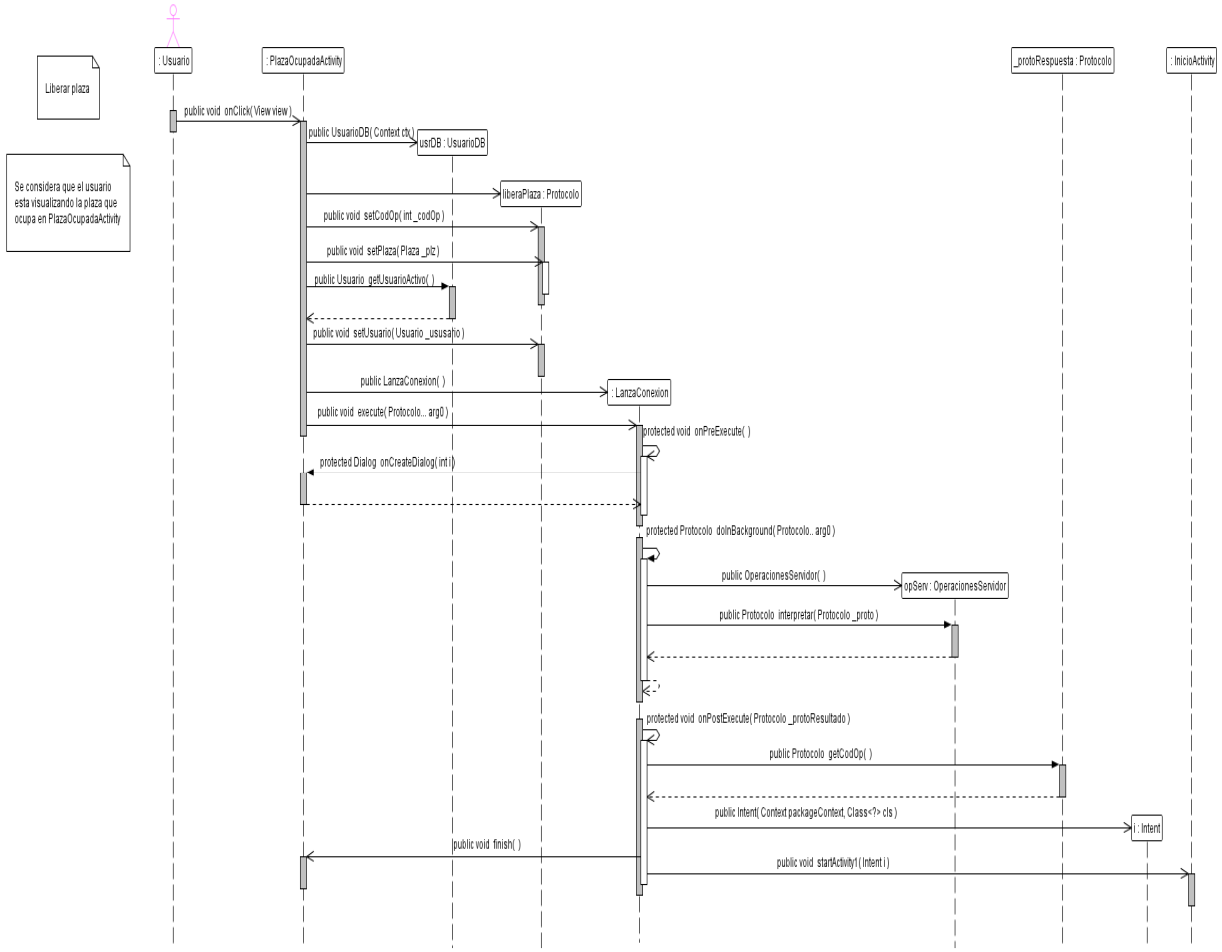


Figura 123: *Diagrama de secuencia liberar plaza*

Tras ocupar una plaza, cuando el usuario la va a abandonar, antes debe liberarla. En el diagrama del presente apartado, se muestra la secuencia de métodos que ejecuta la aplicación *AparcaDroid* cliente para realizar esta operación.

Para ello el usuario deberá pulsar en el botón Liberar plaza, momento en el que la aplicación comienza a recopilar los datos del usuario, la plaza y ejecuta el `AsyncTask` encargado de realizar la conexión con el servidor y las operaciones relacionadas que, en este caso será *LanzaConexión*. Tras establecer la comunicación con el servidor y haber recibido su respuesta, al realizarse la operación correctamente, la aplicación finalizará el *Activity* actual e iniciará el *Activity InicioActivity* donde podrá visualizar de nuevo las plazas libres.

Abrir/cerrar menú

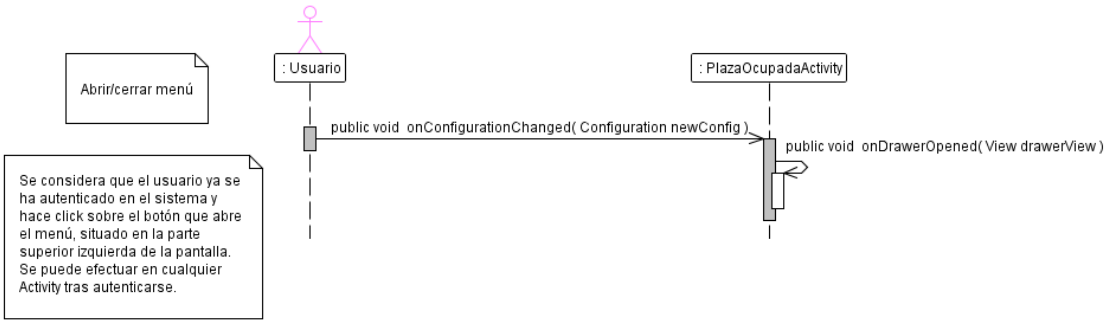


Figura 124: *Diagrama de secuencia abrir/cerrar menú*

En el anterior diagrama se muestran los métodos que ejecuta la aplicación cuando el usuario hace una pulsación sobre el botón que abre y cierra el menú de la aplicación (parte superior izquierda de la pantalla). Esta secuencia de ejecución será la misma tanto para abrir como para cerrar el menú lateral.

Ver perfil

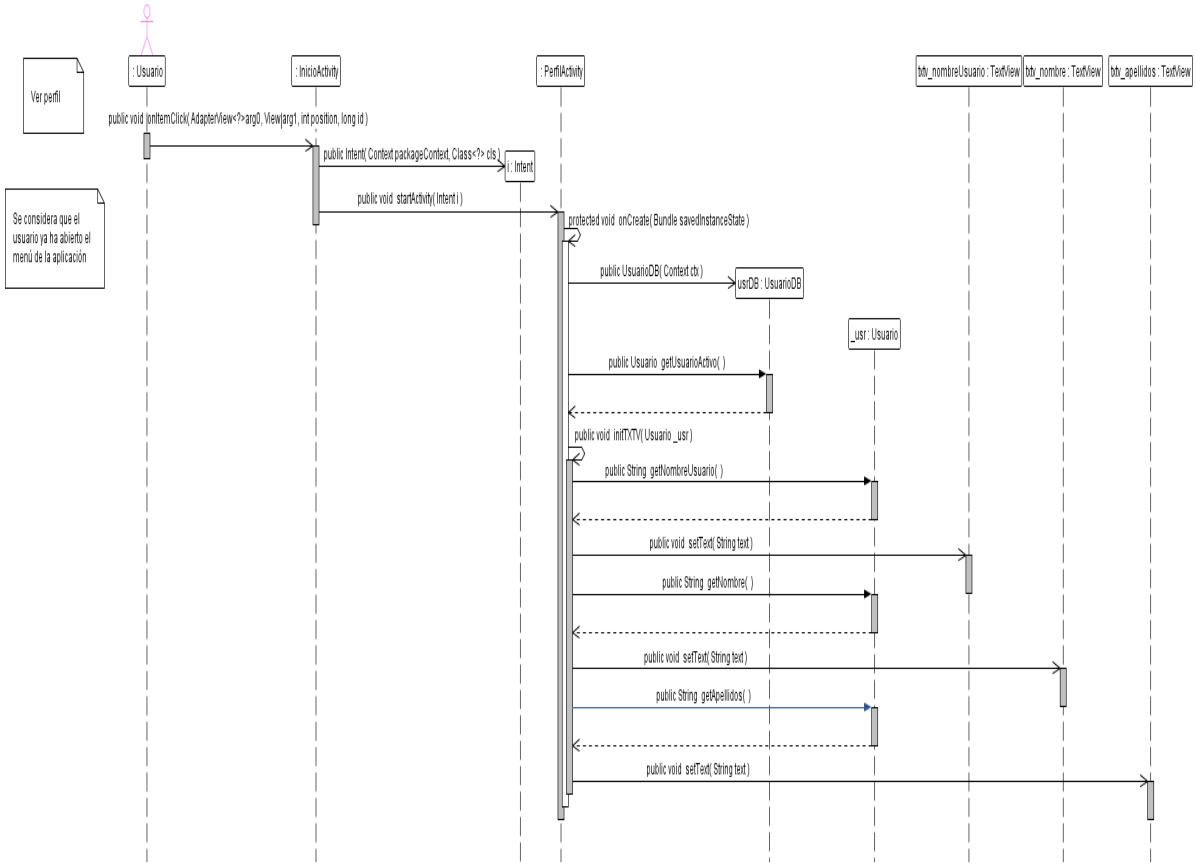


Figura 125: *Diagrama de secuencia ver perfil*

En el diagrama anterior, se muestra la secuencia de ejecución que realiza la aplicación cliente desde que el usuario pulsa sobre la opción del menú Perfil, hasta que se le muestra su perfil de usuario.

Para ello, en el método `onItemClick()` de la opción seleccionada, instancia un nuevo `Intent`, que servirá para abrir la pantalla *PerfilActivity* encargada de mostrar los datos del perfil. En el método `onCreate(Bundle savedInstanceState)` de dicha *Activity* se obtendrán los datos del usuario activo en ese momento, de la base de datos local del dispositivo. Posteriormente, se inicializarán los *TextView* de la pantalla en el método `initTXTV(Usuario _usr)` con los datos obtenidos anteriormente.

Modificar perfil

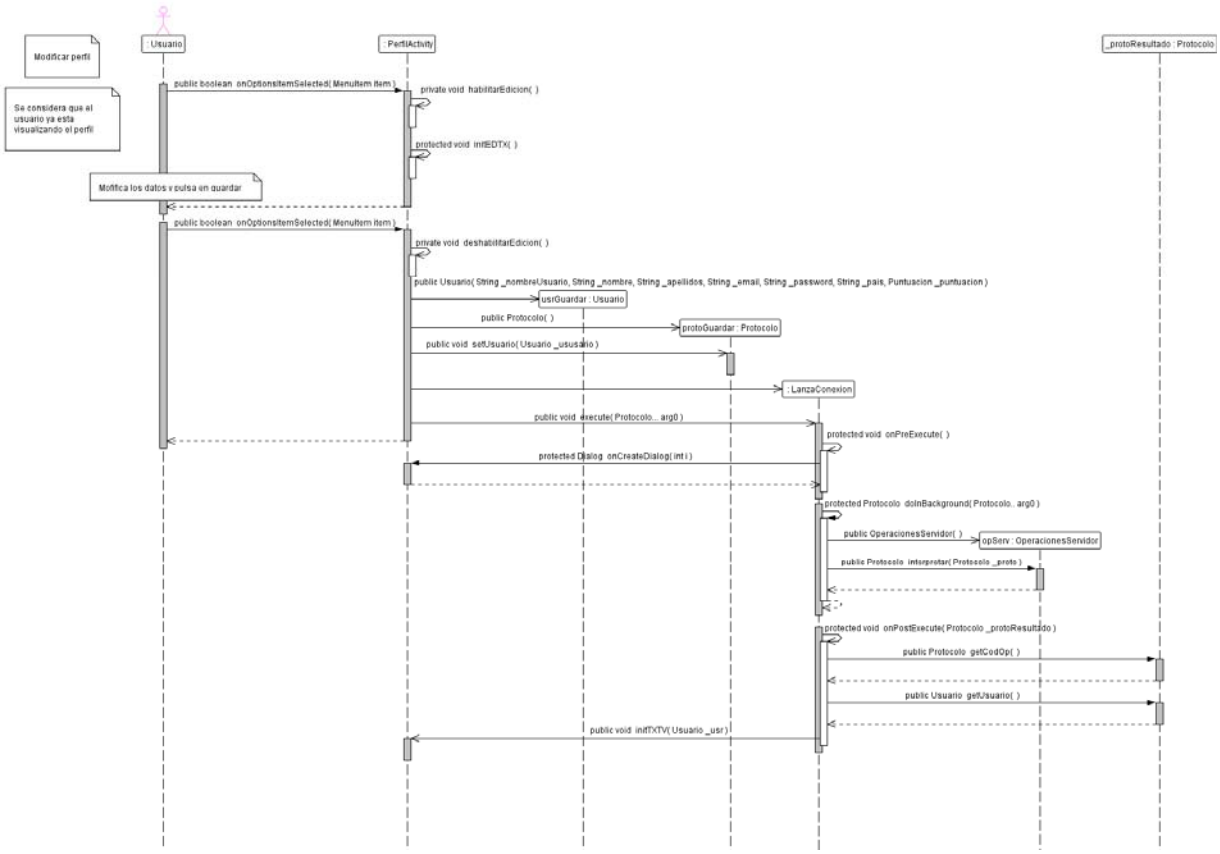


Figura 126: *Diagrama de secuencia modificar perfil*

En este diagrama se muestran los métodos que va a ejecutar la aplicación cliente cuando un usuario desea modificar su perfil. Se considera que el usuario ya se encuentra visualizando dicho perfil, y se plasman los pasos desde que el usuario pulsa en el botón que habilita la edición, hasta que se muestra el perfil una vez guardado los cambios en el servidor y el cliente.

En primer lugar, tras pulsar en el botón de editar se ejecutará el método `onOptionsItemSelected(Menu item)` que captura las pulsaciones sobre el

ActionBar y, se habilitará el formulario de edición inicializándolo con los valores que se mostraban en el perfil. Tras realizar los cambios pertinentes, el usuario deberá pulsar en el botón de guardar los cambios, momento en el cual se ejecutara otra vez el método `onOptionsItemSelected(Menu item)`, obteniendo los datos de los `TextView` y deshabilitando la edición. Tras esto se comenzará a realizar las acciones necesarias para conectar con el servidor y enviar la petición para actualizar el perfil de usuario. Por último, una vez recibida la confirmación del servidor, se mostrará el perfil y se inicializará el perfil con los nuevos datos.

Ver aparcamientos recientes

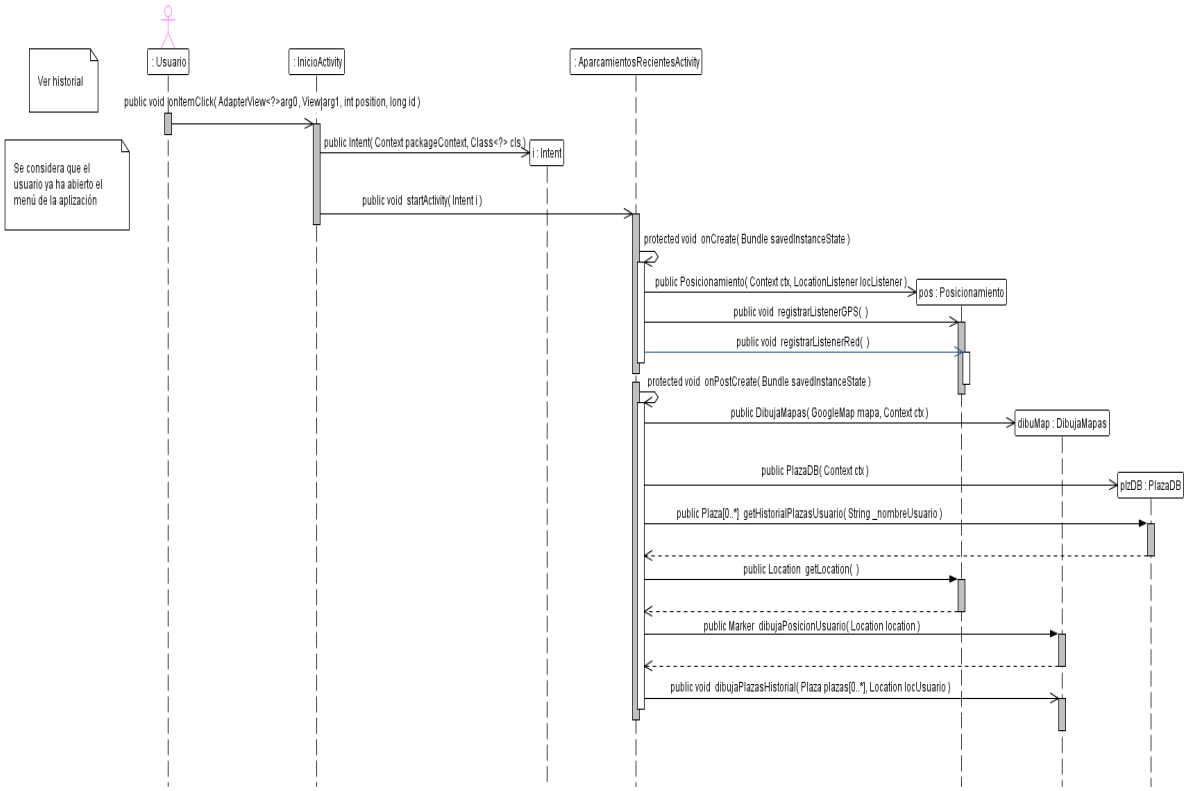


Figura 127: *Diagrama de secuencia ver aparcamientos recientes*

En el diagrama de este apartado, se muestran los métodos y demás acciones que ejecuta la aplicación *AparcaDroid* cliente para ver los aparcamientos recientes o historial de plazas de un usuario autenticado en el sistema. Para la realización del diagrama se ha considerado que ya se muestra en pantalla el menú de la aplicación, por lo que comienza desde el momento en que el usuario pulsa sobre la opción del menú Aparcamientos recientes.

En primer lugar, captura la pulsación sobre la opción del menú Aparcamientos recientes con el método `onOptionsItemSelected(Menu item)` y, tras esto, crea la instancia de `Intent` que nos permitirá iniciar el *Activity*

AparcamientosRecientesActivity que será el encargado de mostrar las plazas del historial almacenadas en el dispositivo. En el método `onCreate(Bundle savedInstanceState)` del *Activity*, se crea la instancia de la clase Posicionamiento y registra los *listener* del GPS y de la Red, que nos permitirán obtener la localización del usuario. Por último, en el método `onPostCreate (Bundle savedInstanceState)`, realiza la obtención de las plazas de la base de datos local, obtiene la localización del usuario y dibuja en el mapa tanto la posición del usuario como las plazas obtenidas.

Ver detalles plaza

Figura 128: *Diagrama de secuencia ver detalles plaza*

En este diagrama se muestra los métodos que ejecuta la aplicación cliente AparcaDroid cuando un usuario realiza una pulsación sobre un *marker* para ver los detalles de la plaza que representa dicho *marker*.

Abrir navegador

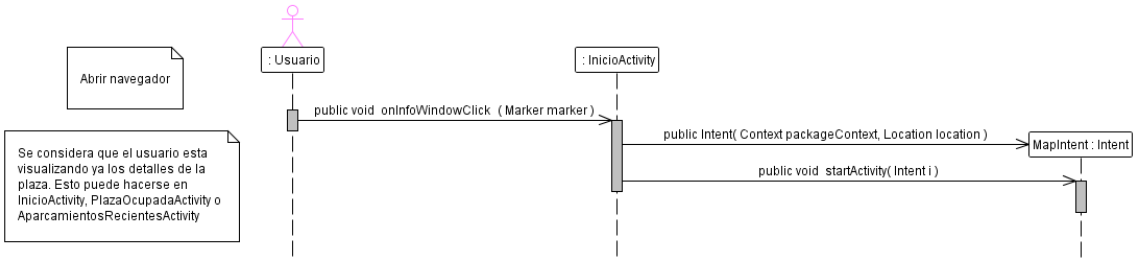


Figura 129: *Diagrama de secuencia abrir navegador*

En el diagrama de secuencia de este apartado se muestra los métodos que ejecuta la aplicación cliente cuando un usuario hace una pulsación sobre el *InfoWindow* de un *marker*. Por ello se considera que el usuario ya está visualizando la información relativa a una plaza.

En primer lugar, el usuario deberá hacer una pulsación sobre el *InfoWindow* (ventana que muestra la información de la plaza) referente a la plaza a la que quiere ser guiado por el Navegador. Dicha pulsación será capturada por el método `public void onInfoWindowClick(Marker marker)`

que posteriormente creará una instancia de `Intent` con la información necesaria para abrir la aplicación del Navegador y, por último, ejecutará el método `public void startActivity(Intent i)` que lanzará la aplicación del navegación.

Conectar

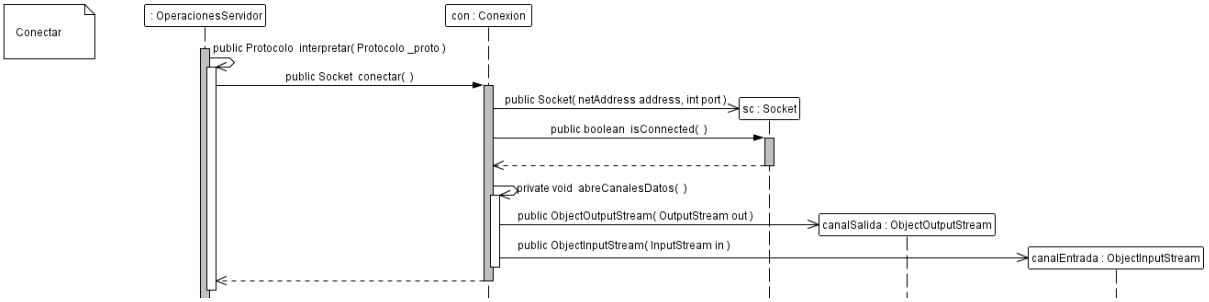


Figura 130: *Diagrama de secuencia conectar*

En el diagrama anterior se muestra los métodos que ejecuta la aplicación cliente a la hora de realizar la conexión con el servidor. Primeramente, inicia la conexión instanciando un Socket utilizando la dirección IP del servidor y el puerto y, posteriormente, abre los canales de entrada y salida de datos utilizando la instancia del Socket obtenida anteriormente.

Desconectar

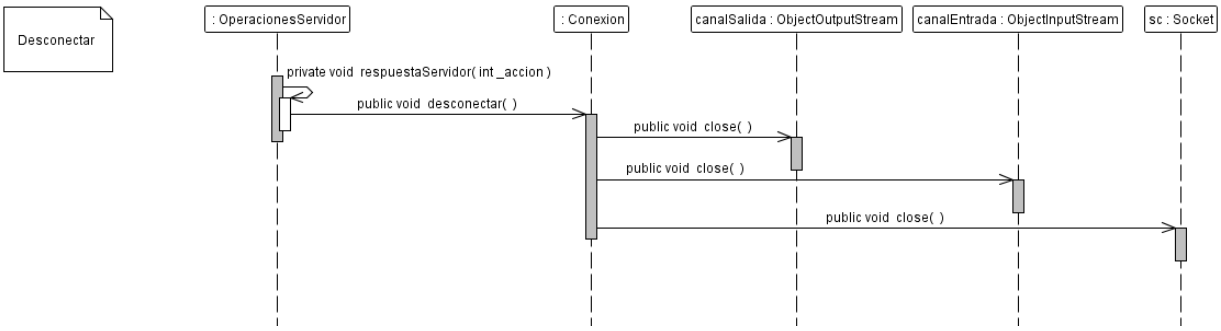


Figura 131: *Diagrama de secuencia desconectar.*

En el diagrama del presenta apartado se muestra los métodos que ejecuta la aplicación *AparcaDroid* cliente cuando se va a desconectar del servidor. Para ello primero cierra los canales de salida y entrada, y posteriormente, cierra el Socket.

Leer

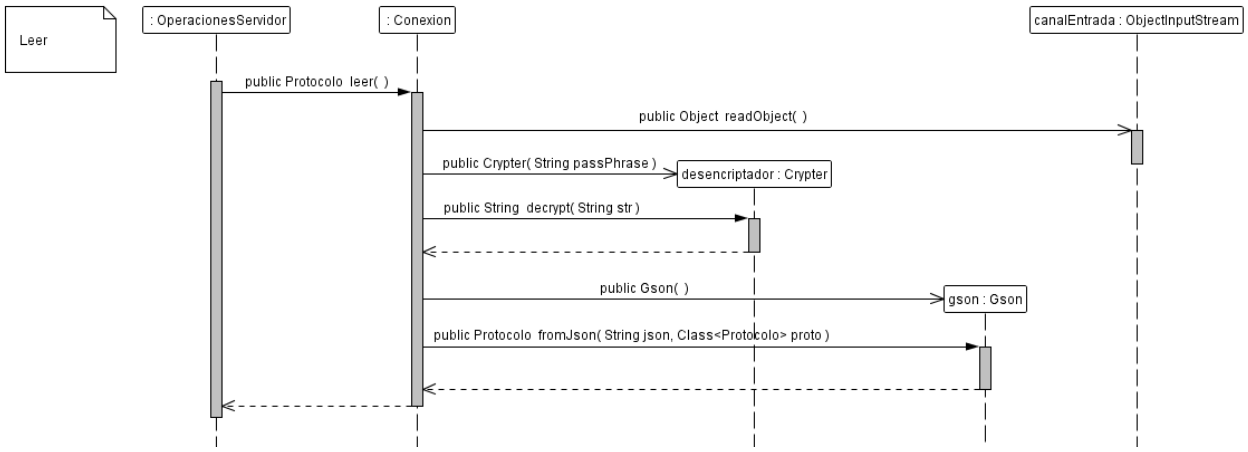


Figura 132: *Diagrama de secuencia leer*

En el anterior diagrama se muestran los métodos que se ejecutan en la aplicación *AparcaDroid* cliente a la hora de leer del canal de comunicación con el servidor considerando que ya se ha realizado la conexión. Para ello, primeramente lee los datos del canal con el método `public Object readObect()` y posteriormente, desencripta los datos leídos con el método `public String decrypt(String str)` de la clase `Crypter`. Por último, convierte el *Json* desencriptado a un objeto de la clase Protocolo, utilizando el método `public Protocolo fromJson(String json, Class<protocolo> proto)`.

Escribir

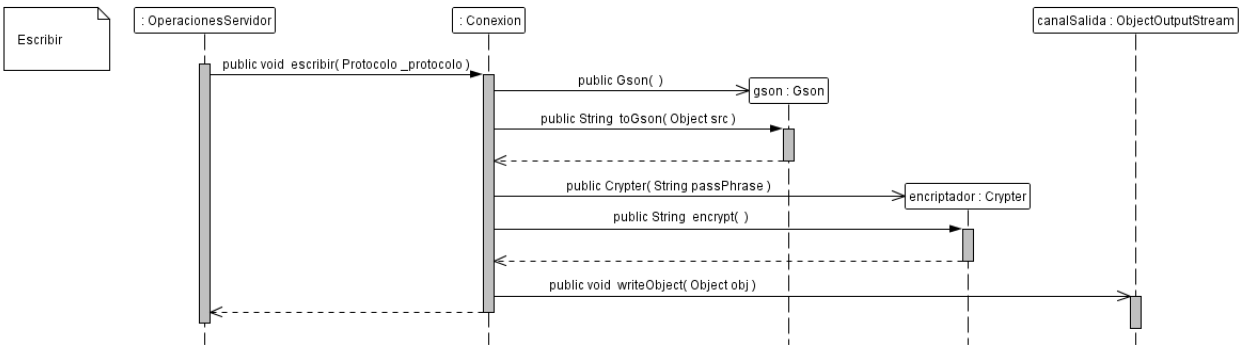


Figura 133: *Diagrama de secuencia escribir*

En este diagrama se muestra el proceso de ejecución de métodos que realiza la aplicación cliente a la hora de escribir datos en el canal para

comunicarse con el servidor, teniendo en cuenta que ya se ha realizado la conexión.

En primer lugar convertirá la clase protocolo que se debe enviar a un objeto de tipo *Json*, cosa que hara utilizando el método `public String toGson (Object obj)` de la librería *Gson*. Tras esto, encriptará la cadena de texto obtenida utilizando el método `public String encrypt(String str)`. Por último se escribirá el Json encriptado en el canal de comunicación con el servidor `canalSalida`.

Activar datos móviles

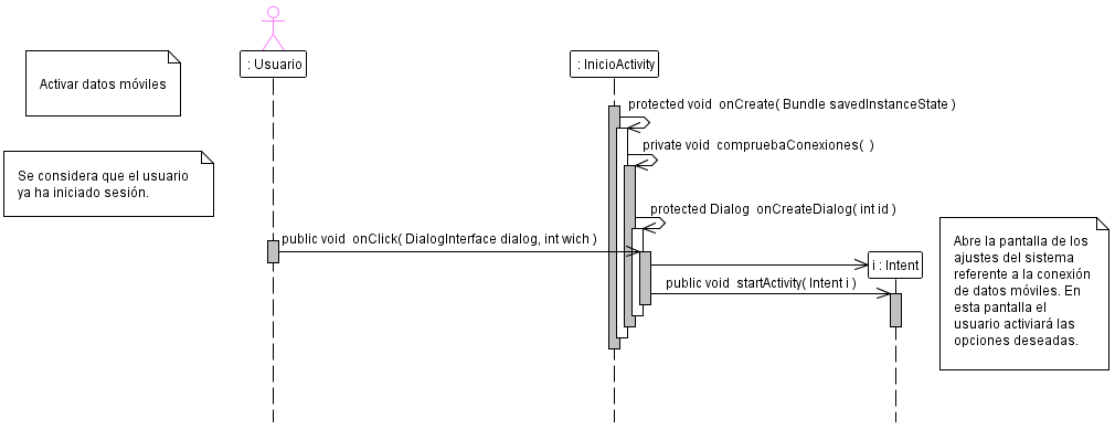


Figura 134: *Diagrama de secuencia activar datos móviles*

Cuando no se dispone de conexión a Internet, se mostrará un diálogo de alerta solicitando que activemos los datos móviles o el Wi-Fi del dispositivo. En el presente diagrama de secuencia se muestra la sucesión de métodos que ejecuta la aplicación AparcaDroid cliente desde que se crea el diálogo de alerta, hasta que se carga la pantalla de los ajustes del sistema referente a dichos datos móviles.

En el método `onCreate(Bundle savedInstanceState)` se comprueba si están activos los recursos necesarios con el método `compruebaConexiones()` y, en el caso de que no exista conexión a internet, ejecuta el método `onCreateDialog(int id)` para mostrar el diálogo de alerta, tras esto espera la pulsación del usuario que se captura a través del método `onClick(DialogInterface dialog, int wich)`. Por último crea una instancia de `Intent` que servirá para cargar la pantalla de los ajustes del sistema referente a los datos móviles.

Activar Wi-Fi

Del mismo modo que en el caso anterior, cuando no se disponga de conexión a Internet, se mostrará el dialogo de alerta para permitir al usuario activar tanto los datos móviles como el Wi-Fi. Por ello, al ejecutarse una secuencia de acciones idéntica desde la pulsación del usuario, el diagrama de secuencia será igual que en el caso anterior (Activar datos móviles) con la diferencia de que en vez de iniciar la pantalla de los ajustes de datos móviles se iniciara la pantalla de los ajustes de la conectividadad Wi-Fi, permitiéndole al usuario conectarse a la red deseada y configurar dicho Wi-Fi según sus preferencias.

Activar GPS

Al igual que en los casos en que se necesite conexión a internet, cuando sea necesario el posicionamiento GPS y, este no esté activado, se mostrará un diálogo de alerta que permitirá al usuario acceder a la pantalla de ajustes del sistema referente a la configuración del GPS.

De este modo, el diagrama de secuencia muestra los métodos que ejecuta la aplicación cliente desde que se crea el diálogo de alerta y el usuario pulsa sobre la opción Activar, hasta que carga la pantalla de los ajustes del GPS.

Salir

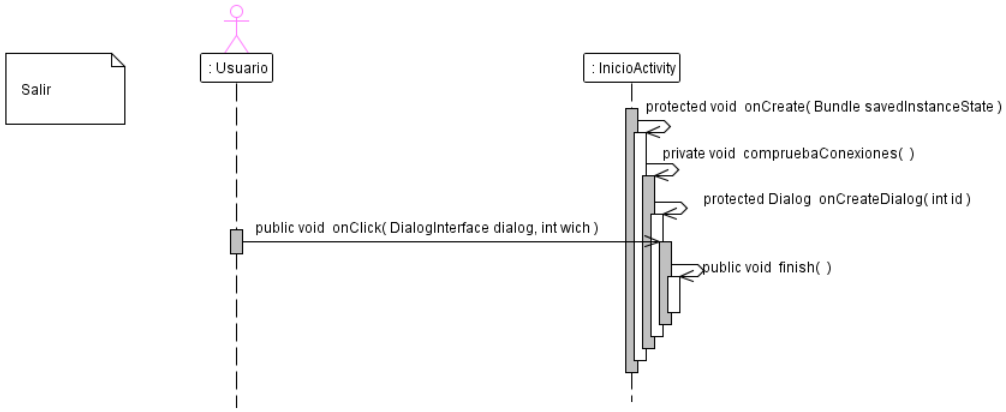


Figura 135: *Diagrama de secuencia salir*

Además de las opciones de activación de las conectividades requeridas en los tres puntos anteriores, en los diálogos de alerta mostrados, también se permitirá al usuario salir de la aplicación si no desea usar ninguno de estos recursos. Por ello este diagrama muestra la secuencia de métodos que ejecuta

la aplicación *AparcaDroid* cliente cuando un usuario pulsa sobre el botón Salir en estos diálogos.

Cerrar sesión

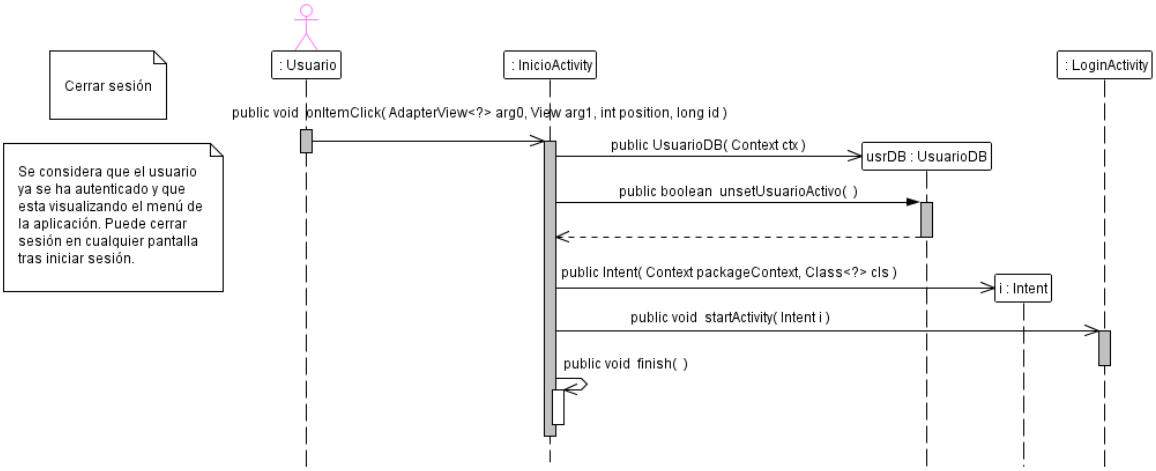


Figura 136: *Diagrama de secuencia cerrar sesión*

En el momento que lo desee, siempre y cuando haya iniciado sesión, un usuario puede cerrar dicha sesión. Por ello el diagrama de secuencia anterior muestra la secuencia de métodos que ejecuta la aplicación cliente desde que el usuario pulsa sobre el botón Cerrar sesión del menú principal, hasta que esta se cierra.

Suponemos que el usuario ya está visualizando el menú, de modo que, la primera acción que realiza es capturar el evento con el método `onItemClick(AdapterView<?> arg0, View arg1, int position, long id)`. Tras ello marcará al usuario activo como inactivo utilizando el método c y, por último, instancia un `Intent` que permitirá iniciar el *Activity* de inicio de sesión *LoginActivity* y finaliza el actual *Activity*.

Interpretar protocolo

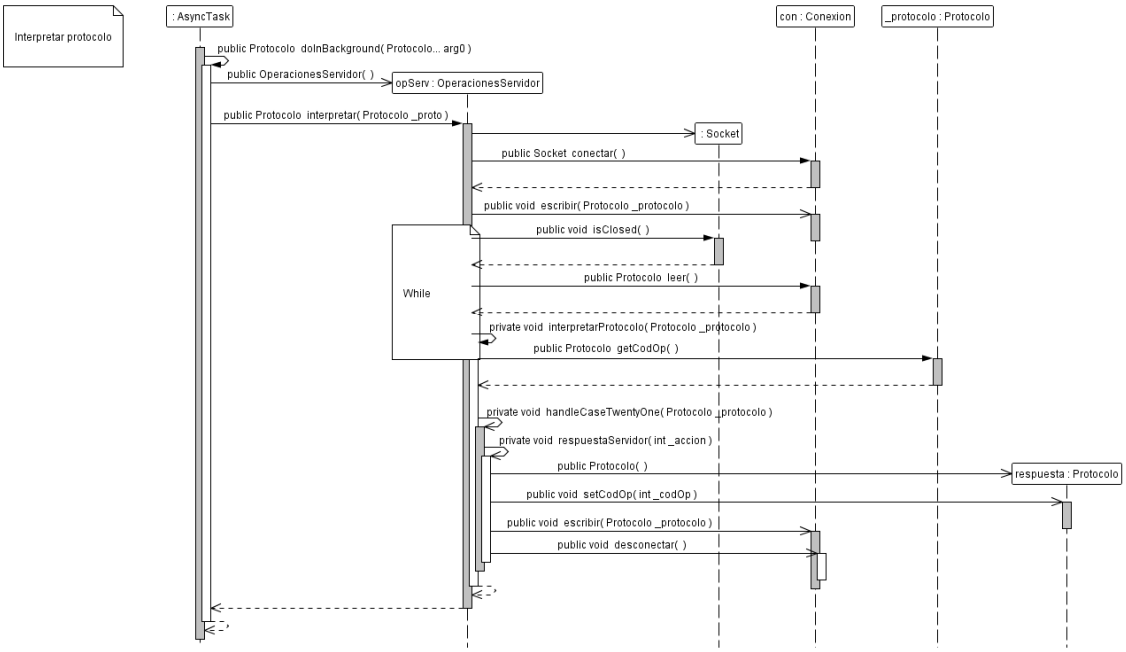


Figura 137: *Diagrama de secuencia interpretar protocolo*

En el anterior diagrama de secuencia se muestra el proceso de métodos que ejecuta la aplicación servidor desde que crea una instancia de la clase *OperacionesServidor* para realizar una petición al servidor, hasta que recibe la respuesta y desconecta.

Tras la instancia se invoca al método `public Protocolo interpretar(Protocolo _proto)` que inicia las acciones de para conectarse al servidor y devuelve el Protocolo con la respuesta. Primeramente instancia el Socket utilizando la dirección IP y el puerto del servidor, posteriormente conecta y escribe en el canal el Protocolo que contiene la petición con el método `escribir (Protocolo _proto)`. Acto seguido, mientras el Socket permanece abierto, estará a la espera de recibir la respuesta del servidor por lo que leerá del canal hasta recibir la respuesta con el método `public Protocolo leer()`. Una vez leído interpreta la respuesta y, en caso de ser correcta, envía la confirmación al servidor con el método `private void respuestaServidor(int _accion)`, escribiendo en el canal un protocolo con la acción indicada y desconectando con el método `public void desconectar()`.

7. IMPLEMENTACIÓN

En este apartado, se comentarán diversos aspectos relacionados con la implementación de las aplicaciones. Para ello, se dará una visión general de las tecnologías y herramientas utilizadas durante el desarrollo de la aplicación, tanto en la fase de diseño como en la de implementación.

7.1. DEREditor

Software de diseño de diagramas Entidad – Relación para bases de datos relacionales [18]. Además permite la generación de código mientras se realiza el diseño. Se encuentra disponible bajo licencia GNU GPL. Este software se ha utilizado para realizar los diagramas entidad-relación de las bases de datos usadas en nuestro sistema (base de datos de la aplicación cliente y de la aplicación servidor).

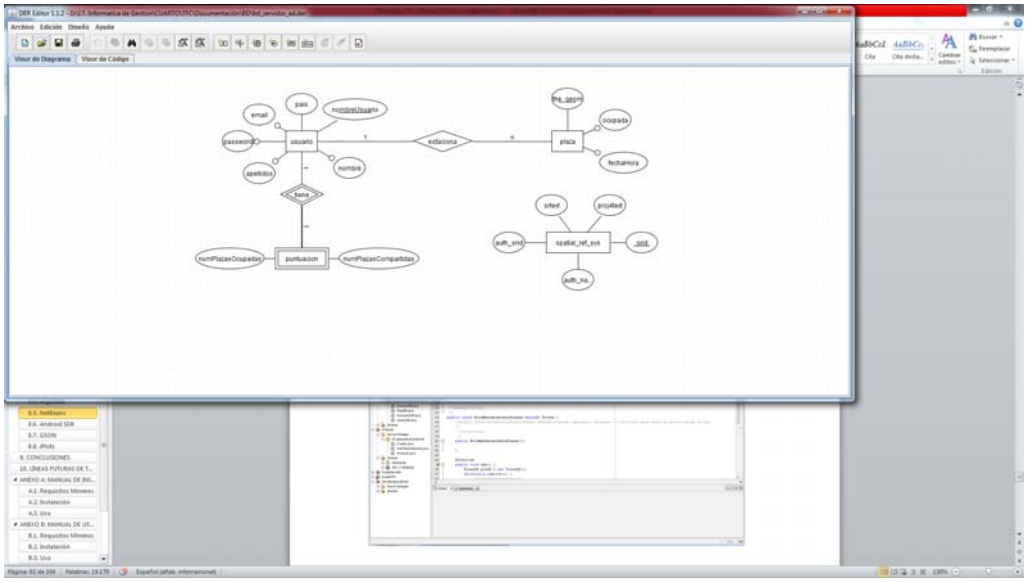


Figura 138*: Interfaz de DEREditor*

7.2. PostgreSQL

Se trata de un Sistema Gestor de Bases de Datos relacional. Es de libre distribución y se publica bajo licencia BSD [6].

Existen varias interfaces de programación de aplicaciones que permiten a aplicaciones escritas en diversos lenguajes de programación, acceder a bases de datos PostgreSQL, incluyendo C, C++, C#, Java, PHP, Perl, Python, Ruby…

El conocimiento previo de este gestor de bases de datos, unido a que se trata de software libre me llevó a elegirlo frente a otras alternativas para su uso en la aplicación.

7.3. pgAdmin

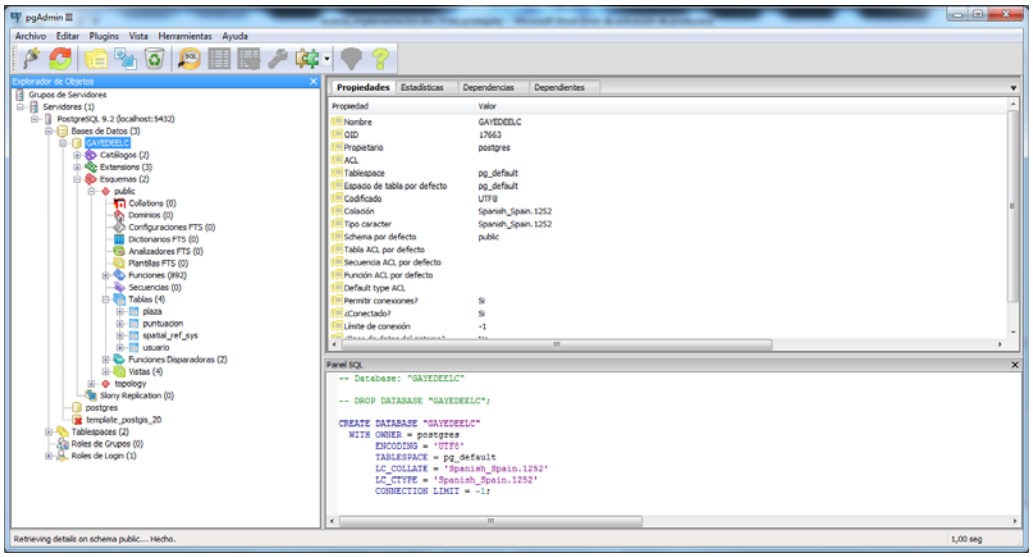


Figura 139: *Interfaz de pgAdmin*

Software que permite conectarse a bases de datos PostgreSQL que se ejecuten en cualquier plataforma. Se trata de un entorno visual instalable en plataformas Linux, FreeBSD, Solaris, Mac OSX y Windows [19].

Este software nos facilita la gestión y administración de bases de datos, pudiendo utilizar tanto instrucciones SQL como el entorno gráfico para realizar las operaciones requeridas (consulta, manipulación y gestión de datos).

7.4. ArgoUML

ArgoUML [20] es una aplicación para crear diagramas UML escrita en Java y publicada bajo licencia BSD. Al tratarse de una aplicación Java, es multiplataforma. Carece de soporte completo para algunos tipos de diagramas, como los diagramas de secuencia y los de colaboración. Se ha utilizado este software para la creación de diversos diagramas UML incluidos en la presente memoria.

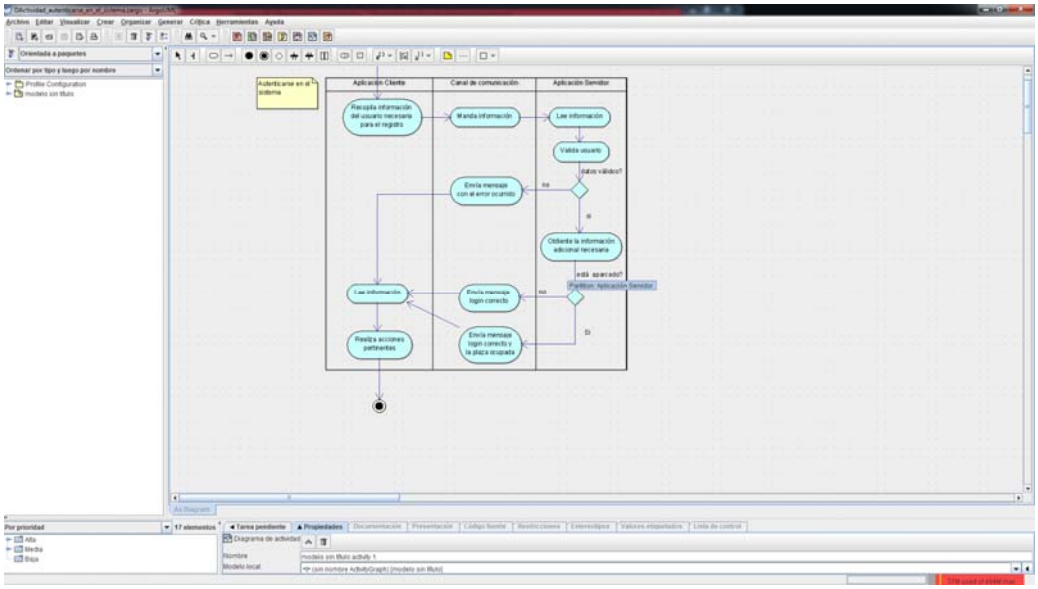


Figura 140: *Interfaz ArgoUML*

7.5. NetBeans

Se trata de un entorno de desarrollo hecho principalmente para Java aunque dispone de un extenso número de módulos que permiten la programación en otros lenguajes como: C, C++, HTML5, PHP... [21] Desde 2007 se distribuye bajo licencia CDDL y GPL versión 2. Este entorno de desarrollo se ha utilizado para desarrollar los paquetes del sistema que son Java puro, es decir, las clases comunes y las referentes a la aplicación servidor.

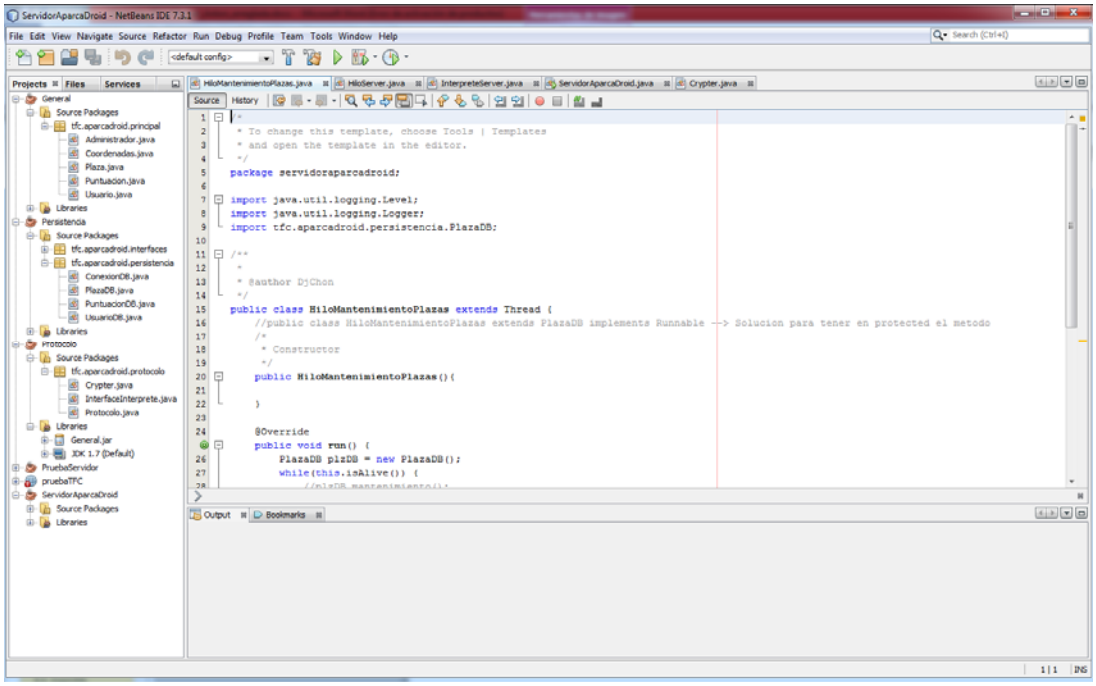


Figura 141: *Interfaz NetBeans*

7.6. Android SDK

El Software Development Kit (SDK) de Android [22] incluye un conjunto de herramientas de desarrollo. Este conjunto de herramientas lo forman un depurador de código, biblioteca, simulador de teléfonos basado en QEMU, documentación, ejemplos de código y tutoriales. Está disponible para plataformas Linux, Mac OSX 10.4.9 o posterior y, Windows XP o posterior.

Dispone de una plataforma de desarrollo o IDE (Integrated Development Environment) oficial basada en Eclipse la cual integra el complemento ADT (Android Development Tools). El SDK soporta también versiones antiguas de Android, de modo que se pueden instalar cualquier versión de Android (anteriores o más modernas) en el momento que sea necesario, de forma que podamos desarrollar software para cualquier versión instalada en nuestro equipo.

Una aplicación Android está compuesta por un conjunto de ficheros empaquetados en formato .apk y guardada en el directorio /data/app del sistema operativo Android. Directorio que, por motivos de seguridad, necesita permisos de superusuario o root para poder acceder a él. Con todo esto, un paquete APK incluye ficheros .dex (ejecutables Dalvik, un código intermedio compilado), recursos, etc.

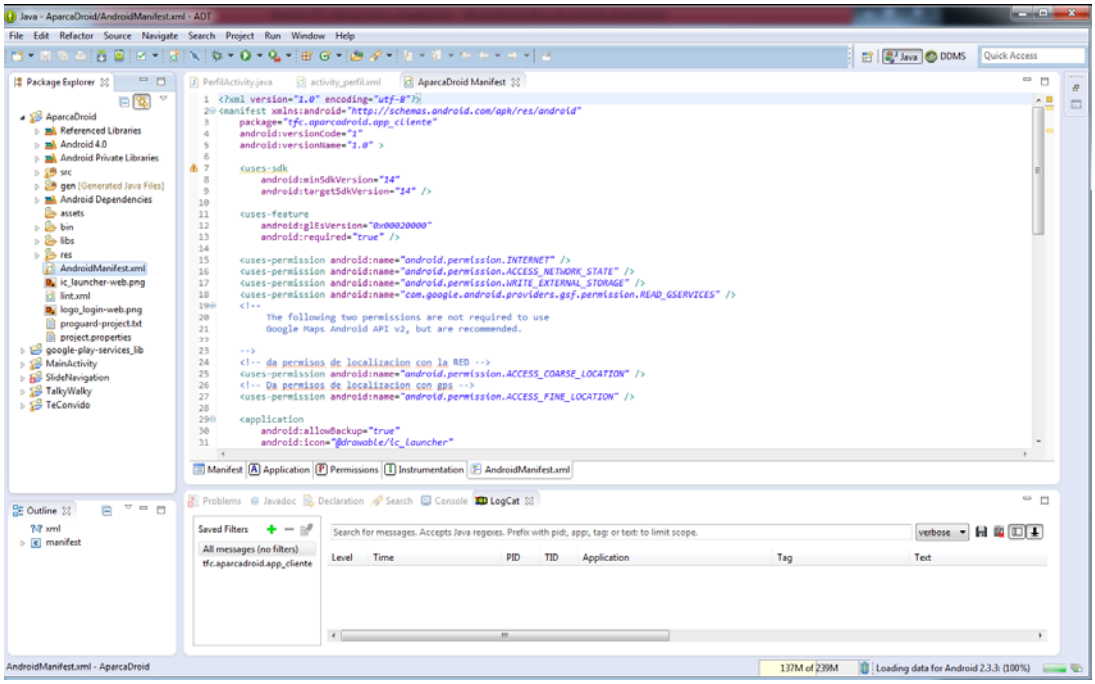


Figura 142: *Interfaz Eclipse + ADT*

7.7. GSON

GSON [23], es una librería de código abierto para el lenguaje de programación Java que permite la serialización y deserialización entre objetos Java y su representación en JSON.

7.8. iPlotz

iPlotz [24] es una aplicación para crear mockups (prototipos) de interfaces gráficas de usuario. Es una aplicación de pago, pero dispone de una versión de prueba gratuita que es la utilizada para el diseño de cada uno de los layout de nuestra aplicación.

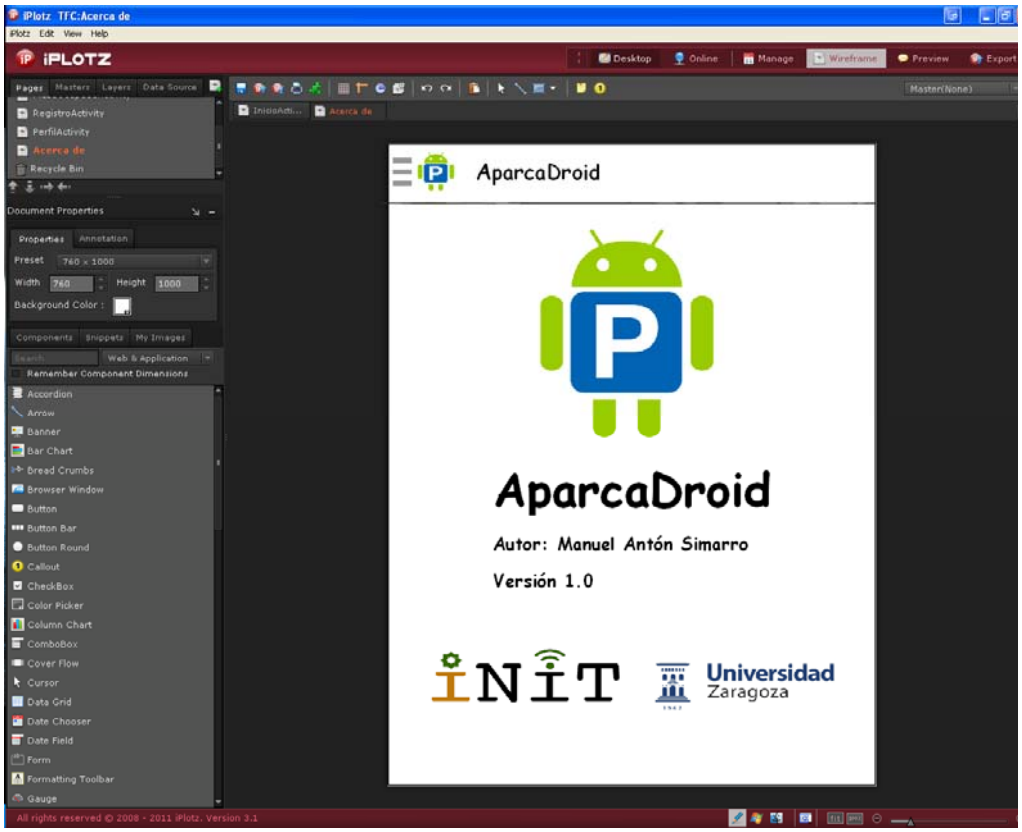


Figura 143: *Interfaz iPlotz*

8. LICENCIA

Las dos aplicaciones de las que consta este trabajo se han licenciado con GNU GPL versión 3. Esta licencia es compatible con Apache License 2.0 la cual licencia los componentes principales de la aplicación como son la API de Android, la librería Gson y Google Maps. Es la más ampliamente usada, y su propósito principal es proteger el software de intentos de apropiación que restrinjan las libertades de los usuarios.

Las condiciones y restricciones de la licencia son las siguientes:

- Permite los usuarios la libertad de usar, estudiar, compartir y modificar el software.
- Puede ser usada por cualquiera, ya que su finalidad es proteger los derechos de los usuarios.
- Es una licencia con *copyleft* lo que significa que los trabajos derivados sólo pueden ser distribuidos bajo los términos de la misma licencia. Lo cual asegura que el software está protegido cada vez que es distribuido, modificado o ampliado.
- El software puede ser aplicado bajo todos los propósitos, incluso los comerciales o como herramienta de creación de software propietario.
- En uso puramente privativo o interno, el software puede ser modificado sin liberar el código fuente. En caso contrario, el código fuente y cualquier cambio debe estar disponible para los usuarios, ya que en este caso los derechos de los usuarios están protegidos por el *copyleft*.
- Si un programa utiliza fragmentos de código bajo licencia GNU GPL y es distribuido, su código fuente completo debe estar disponible bajo la misma licencia.
- Los usuarios o compañías que distribuyen software bajo esta licencia, pueden cobrar dicho software o distribuirlo gratuitamente.
- Un distribuidor no pude imponer “restricciones sobre los derechos otorgados por la licencia GPL”.
- Prohíbe la distribución de software bajo un acuerdo de confidencialidad o contrato.

- Los programas distribuidos como binarios precompilados, deberán estar acompañados de una copia del código fuente.

- Cada copia del software bajo licencia GNU GPL, debe suministrarse junto con una copia de la licencia.

- La versión 3, utilizada en el presente software, permite que el código fuente esté disponible en distintas plataformas. Cosa que incluye la descarga desde un servidor de red o a través de peer-to-peer, siempre que el código compilado esté disponible y que haya "instrucciones claras" sobre dónde encargar el código fuente.

- No se permite la aplicación de derechos de copyright a una obra bajo licencia GPL, al menos que el autor los aplique explícitamente.

- Sólo los titulares de los derechos individuales tienen la autoridad para demandar una violación de licencia.

Los términos de uso completos de la licencia se pueden encontrar dentro de la carpeta Licencia suministrada junto con los fuentes y demás documentación del proyecto en un archivo llamado "The GNU General Public License v3.pdf". También es posible encontrar una traducción no oficial, que implica el no tener validez legal, en el archivo "The GNU General Public License v3 (Spanish_non_official).pdf" dentro de la misma carpeta.

9. CONCLUSIONES

El principal objetivo que se pretende conseguir con la realización de este proyecto, es diseñar e implementar una aplicación que ayude a los conductores a la hora de buscar aparcamiento en las ciudades, con el consiguiente ahorro de tiempo y combustible, sobre todo en aquéllas donde el número de vehículos sea mayor.

Para ello se usa un dispositivo Android, que hoy en día, es un sistema operativo para dispositivos móviles que está mayoritariamente extendido tanto en nuestro país como a nivel mundial.

Debido al problema de la fragmentación de Android (existen varias versiones del sistema operativo conviviendo al mismo tiempo), se debe considerar concienzudamente que API se utilizará para programar la aplicación cliente, ya que, a partir de la versión 4.0 existen algunas funcionalidades nuevas que no son compatibles con las anteriores versiones y otras que han desaparecido en éstas nuevas versiones.

Otro problema fue implementar el algoritmo para la selección de la plaza a ocupar, ya que hay que considerar el error del GPS que, aunque cada vez es menor, es una variable a tener en cuenta ya que puede cambiar radicalmente el resultado de dicha operación y comprometer.

A pesar de todo esto, los objetivos perseguidos se han cumplido en su totalidad, habiéndose implementado una aplicación totalmente funcional. Además, se ha conseguido que se pueda instalar en aproximadamente el 86% de los terminales Android según los datos indicados en el apartado 2.1.3 y, con una ejecución fluida y eficiente. Para ello se han utilizado una gran variedad de componentes de diseño proporcionados por Android como pueden ser elementos del *ActionBar* o de *NavigationDrawer* que posibilitan la creación de una interfaz de usuario con mejor aspecto visual, permitiendo que la interacción del usuario con la aplicación sea más sencilla, mejorando sustancialmente la experiencia del usuario.

Se ha implementado un sistema fiable y que consigue gracias al uso del GPS y la lógica de la aplicación, un posicionamiento preciso que permita identificar correctamente la plaza que se ocupa y, posteriormente, liberarla. De este modo, un usuario que use la aplicación podrá encontrar plazas de aparcamiento libres, donde poder estacionar su vehículo, de una manera fácil y rápida.

Obviamente, cuanto mayor sea el número de usuarios de AparcaDroid, su funcionamiento será mucho mejor, ya que aumentará el número de plazas libres y su distribución por toda la ciudad.

10. LÍNEAS FUTURAS DE TRABAJO

En cuanto a las líneas futuras de trabajo se pueden destacar las siguientes:

1. **Nuevas funcionalidades:**
    a. Implementación de los componentes necesarios para la selección del tipo de plaza (Normal, Zona Azul, Zona de Residentes, Plaza de Minusválido, etc.).
    b. Configuración de los parámetros para la obtención de las plazas libres y búsqueda a partir de la dirección.
2. **Puesta en funcionamiento:** Al tratarse de un Proyecto Final de Carrera, la idea todavía se encuentra en una fase inicial, aunque con los medios adecuados se podría convertir en una realidad.
3. **Mantenimiento y optimización del servicio:** Se realizarán tareas de optimización y actualización de las aplicaciones cliente y servidor en función de las nuevas versiones que vayan apareciendo de Android y Java respectivamente.
4. **Expansión a otras plataformas:** Se podrá crear la aplicación cliente para otras plataformas móviles de actualidad para ampliar el número de usuarios de nuestro servicio y, de este modo, mejorarlo. En la actualidad, dichos Sistemas Operativos podrían ser: IOS, BlackBerry, Windows Phone o Firefox OS.
5. **Internacionalización de la aplicación cliente:** Se continuará con la mejora de la aplicación traduciéndola a más idiomas para que ésta pueda alcanzar una cuota de mercado mayor dentro de países de habla no hispana.
6. **Inclusión de publicidad:** Una vez puesto en marcha el servicio, se podría incluir publicidad en la aplicación cliente para obtener algún beneficio de ello, y así ayudar a su viabilidad y futuras mejoras.

BIBLIOGRAFÍA

[1] Estudio mundial sobre aparcamiento elaborado por IBM.
http://www-03.ibm.com/press/es/es/pressrelease/35510.wss
Agosto de 2014

[2] Android
http://es.wikipedia.org/wiki/Android
Agosto de 2014

[3] Android Dashboard
https://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net
Agosto de 2014

[4] A-GPS
http://es.wikipedia.org/wiki/GPS_Asistido
Agosto de 2014

[5] Google Maps
http://es.wikipedia.org/wiki/Google_Maps
Agosto de 2014

[6] PostgreSQL
http://es.wikipedia.org/wiki/PostgreSQL
Agosto de 2014

[7] PostGIS
http://es.wikipedia.org/wiki/PostGIS
Agosto de 2014

[8] Open Spot
http://techcrunch.com/2010/07/09/google-parking-open-spot/
Agosto de 2014

[9] SF Park
http://sfpark.org/
Agosto de 2014

[10] Aparcamientos España
http://www.androidpit.es/es/android/market/aplicaciones/aplicacion/com.ao.aparcamientos.es/aparcamientos-espana
Agosto de 2014

[11] AA Parking
https://play.google.com/store/apps/details?id=com.theaa.android.parking&hl=es
Agosto de 2014

[12] Parking Finder
http://isuriv.wordpress.com/2009/06/24/parking-finder-aparca-con-android/

[13] SPark
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5062057&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5062057
Agosto de 2014

[14] ePark
http://www.elmundo.es/elmundomotor/2013/05/22/conductores/1369237627.html
Agosto de 2014

[15] IEEE. "IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications", IEEE Computer Society, 1998.
http://standards.ieee.org/findstds/standard/830-1998.html
Agosto de 2014

[16] Puntos función
http://www.tgti.es/sites/default/files/PuntoFuncion_1_0.pdf
Agosto de 2014

[17] Convenio colectivo.
https://236ws.dpteruel.es/DPT/bopt.nsf/0/88737CD8A0D55E3BC12577E50028279A/$FILE/24%20NOVIEMBRE.pdf
Agosto de 2014

[18] Dereditor
http://sourceforge.net/projects/dereditor/
Agosto de 2014

[19] pgAdmin
http://www.pgadmin.org/
Agosto de 2014

[20] ArgoUML
http://argouml.tigris.org/
Agosto de 2014

[21] NetBeans
https://netbeans.org/
Agosto de 2014

[22] Android SDK
http://es.wikipedia.org/wiki/Desarrollo_de_programas_para_Android
Agosto de 2014

[23] Gson
https://code.google.com/p/google-gson/
Agosto de 2014

[24] iPlotz
http://iplotz.com/
Agosto de 2014

[25] Android Developer
http://developer.android.com/index.html
Agosto de 2014

[26] sgOliver
http://www.sgoliver.net/
Agosto de 2014

[27] StackOverflow
http://stackoverflow.com/
Agosto de 2014

[28] Androcode
http://androcode.es/
Agosto de 2014

[29] Androideity
http://androideity.com

[30] Adictos al Trabajo
http://www.adictosaltrabajo.com/tutoriales/

ANEXO A: MANUAL DE INSTALACIÓN

En este manual se muestra el proceso a seguir para instalar la aplicación servidor, así como la forma de ejecución y configuración de la misma.

A.1. Requisitos Mínimos

Componente	Requisitos mínimos
Sistema operativo	Windows 7
Procesador	Intel Core 2 Duo 2.0GHz
Disco duro	100 MB
Memoria RAM	4GB
Resolución de pantalla	1024x768 píxeles
Java	JRE 7
PostgreSQL	9.2
PostGIS	2.0

Anexo A. Tabla 1: *Requisitos mínimos aplicación servidor*

Los requisitos de la tabla anterior, son los mínimos con los que se garantiza un funcionamiento adecuado de la aplicación servidor.

En cuanto a la base de datos de la aplicación, podrá estar alojada en el mismo servidor que la aplicación principal. En este caso los 100MB mínimos no serán suficientes, por lo que tendremos que contar con un espacio de almacenamiento libre superior, que en un inicio, deberá ser de 1GB y, dependiendo del número de usuarios y plazas registrados, habrá que ampliarlo ajustándonos a la demanda.

A.2. Instalación

Es recomendable que la instalación de la Aplicación Servidor la lleve a cabo una persona con conocimientos informáticos, ya que hay pasos que pueden resultar demasiado complicados o confusos, para un usuario con conocimientos informáticos básicos.

En primer lugar, se deberá descargar e instalar (en caso de que no estén ya), en el orden indicado, los siguientes componentes:

- *Java JRE 7*. Puede descargarse aquí: https://www.java.com/es/download/
- *PostgreSQL*. Puede descargarse aquí: http://www.postgresql.org/download/
- *PostGIS*. Puede descargarse aquí: http://postgis.net/install

Se deberá crear la base de datos necesaria para el funcionamiento de la aplicación, para ello se podrá importar el script *bd.sql* incluido en la carpeta de la aplicación, bien desde consola o utilizando *pqAdmin* u otra herramienta de administración de bases de datos *PostgreSQL* que sea compatible con el complemento *PostGIS*.

La instalación de los componentes de la base de datos se deberá realizar únicamente en el equipo que deseamos que aloje dicha base de datos. En el caso de que esté alojada en otra máquina tendremos que editar el fichero de configuración correspondiente.

El último paso es asegurarnos que la máquina donde se ejecuta la aplicación tiene acceso a Internet, y que ésta tiene acceso a la base de datos (si está en otra máquina, hay que asegurarse que existe conexión de red entre ambas).

A.3. Uso

Para iniciar la aplicación, se deberá ejecutar el fichero *ServidroAparcaDroid.java* bien utilizando la consola, o bien haciendo doble clic sobre él. Tras ello podemos comprobar que se está ejecutando desde el *Administrador de Tareas* (pulsando *ctrl. + alt + supr*).

ANEXO B: MANUAL DE USUARIO

En este manual se muestra el proceso a seguir para instalar la aplicación cliente en un terminal Android, así como la forma de interactuar con ella durante su ejecución.

B.1. Requisitos Mínimos

Componente	Requisitos mínimos
Sistema operativo	Android 4.0 o superior
Procesador	Dual-Core 1GHz
Memoria RAM	512MB
Resolución de pantalla	768x1080 píxeles
Almacenamiento libre	5MB

Anexo B. Tabla 1: *Requisitos mínimos aplicación cliente*

Los requisitos de la tabla anterior, son los mínimos con los que se garantiza un funcionamiento adecuado de la aplicación servidor.

B.2. Instalación

La instalación de la aplicación cliente la puede realizar cualquier tipo de usuario, basta con tener conocimientos básicos de informática.

En primer lugar deberemos disponer del fichero de instalación *AparcaDroid*.apk en la memoria de nuestro teléfono, este *apk* se proporciona dentro de la carpeta de la aplicación. Por último, deberemos buscar el fichero de instalación con nuestro explorador de archivos preferido y pulsar sobre el para ejecutarlo. Una vez hecho esto pulsaremos en el botón Instalar y, tas aceptar al concluir la instalación, la aplicación ya estará lista para su uso.

B.3. Uso

Para iniciar la aplicación se deberá pulsar sobre el icono de ésta.

Anexo B. Figura 1: *Icono de la aplicación*

B.3.1. Activar la conexión a Internet

La aplicación necesita conexión a Internet para poder obtener los datos del servidor, y por ello, en el caso de que no disponga de la conectividad necesaria, mostrará una alerta en forma de diálogo. En dicho diálogo podremos elegir entre activar el Wi-Fi, los datos móviles o salir de la aplicación mediante la pulsación sobre el botón correspondiente. En cualquiera de las opciones para activar la conectividad, tanto Wi-Fi como datos, abrirá la pantalla de configuración del sistema correspondiente a cada una de ellas, de modo que, podamos configurarlas a nuestro gusto.

Anexo B. Figura 2: *Captura de pantalla activar conexión a Internet*

B.3.2. Activar GPS

Para mejorar el posicionamiento, y por lo tanto, la precisión a la hora de obtener las plazas libres y ocupar una plaza, se necesita utilizar el GPS del dispositivo. Por ello, cuando sea necesario y no este activado, la aplicación mostrará un diálogo de alerta, en el cual podremos elegir entre activar el GPS o salir de la aplicación. Estas acciones las podremos realizar mediante la pulsación sobre los botones correspondientes, de forma que si seleccionamos activar el GPS nos abrirá la pantalla de configuración del sistema correspondiente al GPS permitiéndonos configurarlo según nuestras preferencias.

Anexo B. Figura 3: *Captura de pantalla activar GPS*

B.3.3. Registrar Usuario

Para usar la aplicación, es necesario disponer de un usuario registrado. Por esto, debemos introducir los campos solicitados en los *editText* correspondientes. Dichos campos son *nombre de usuario*, *nombre*, *apellidos, contraseña* y *repetir contraseña*. Tras ello, se ha de pulsar en el botón *Terminado* y, una vez comprobados y validados, se procederá a efectuar el registro, informando al usuario si ha sido realizado correctamente o no.

Anexo B. Figura 4: *Captura de pantalla registrar usuario*

B.3.4. Autenticarse en el sistema

Una vez nos hayamos registrado, podremos iniciar sesión en el sistema tantas veces como deseemos. Para ello deberemos introducir el email de registro y la contraseña en los *editText* correspondientes. En caso de que se produzca algún error a la hora de iniciar sesión, el usuario será informado con los detalles de dicho error.

Anexo B. Figura 5: *Captura de pantalla autenticarse en el sistema*

B.3.5. Ocupar plaza

Una vez nos hayamos registrado y autenticado en el sistema, en el caso de que no lo hayamos hecho antes y estemos ocupando una plaza, podremos buscar un estacionamiento libre y una vez aparcados, pulsar en el botón *Ocupar Plaza* de modo que, la plaza correspondiente a nuestra posición será ocupada.

Anexo B. Figura 6: *Captura de pantalla ocupar plaza*

B.3.6. Liberar plaza

Tras habernos autenticado en el sistema, en el caso de que hubiéramos ocupado una plaza anteriormente, debemos liberar dicha plaza antes de abandonar dicho estacionamiento. Por ello la aplicación, en este caso, dispone de un botón Liberar Plaza que pulsando sobre el realizará esta acción.

Anexo B. Figura 7: Captura de pantalla liberar plaza

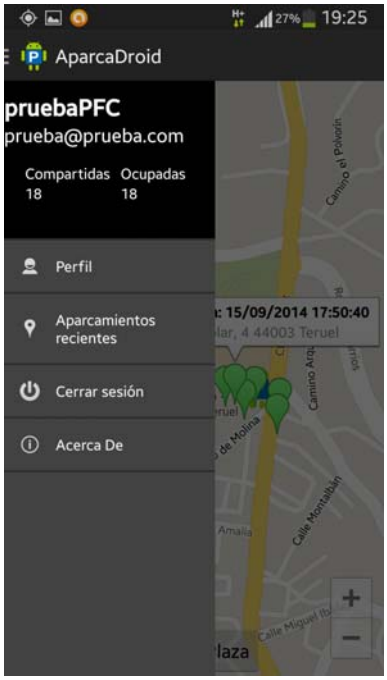
B.3.7. Ver/ocultar los detalles de una plaza y abrir el navegador

Durante la ejecución de la aplicación, las plazas aparecen indicadas mediante el uso de marcadores (*markers* en inglés). Para ver la información de cada una de las plazas, se tiene que pulsar sobre el marcador de la plaza de la cual deseemos consultar la información. Del mismo modo, para ocultar la información, deberemos volver a pulsar sobre dicho marcador.

Por otro lado, para abrir el navegador predeterminado del dispositivo, y así ser guiados hasta la plaza deseada, deberemos pulsar sobre el mensaje que muestra la información de la plaza.

B.3.8. Abrir /cerrar el menú de la aplicación

Para acceder al menú principal de la aplicación se deberá pulsar sobre el icono de la parte superior derecha de la pantalla o sobre el logo de la aplicación, también en la parte superior. Estando el menú cerrado, una pulsación lo abre y, estando abierto, otra pulsación lo cierra. También se puede cerrar el menú pulsando el botón atrás de nuestro terminal.



Anexo B. Figura 8: *Captura de pantalla abrir/cerrar menú de la aplicación*

B.3.9. Ver / modificar el perfil de usuario

Cada usuario puede consultar su perfil en la aplicación. Para ver y/o modificar el perfil, se deberá seleccionar la opción Perfil dentro del menú de la aplicación(en el apartado B.3.8 se describe como acceder a él). Dicho perfil contendrá la información

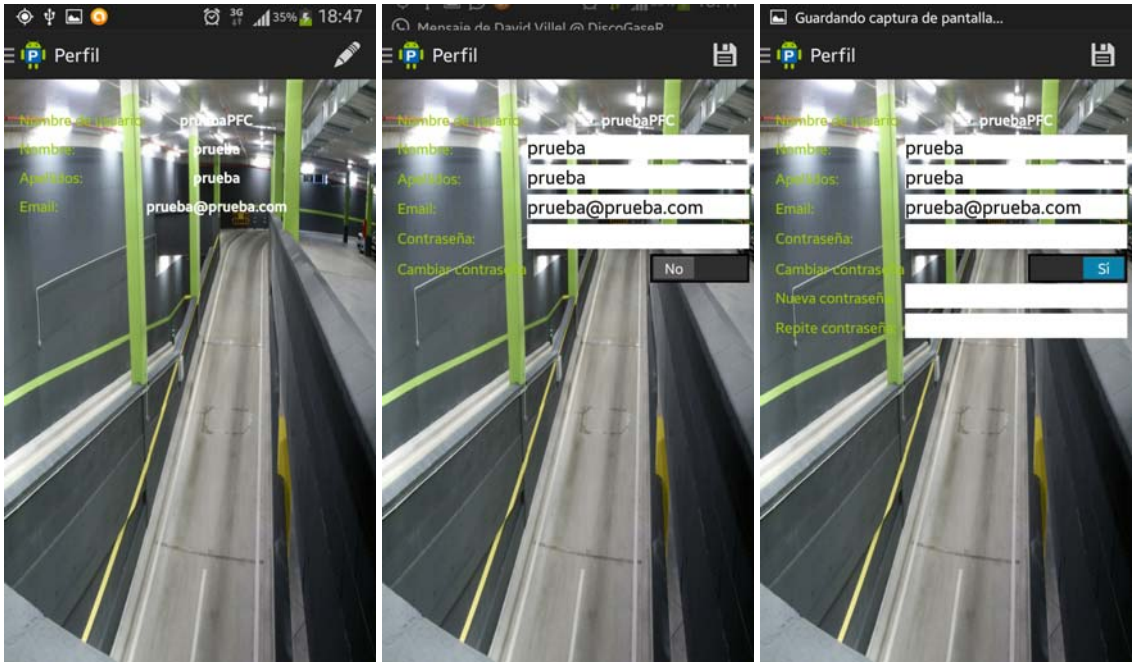
básica de usuario exceptuando la contraseña y, para modificarla, pulsar sobre el icono editar de la parte superior derecha de la pantalla.

La información que se permite editar es la siguiente:

- *Nombre*
- *Apellidos*
- *Email*
- *Contraseña*

Para modificar los datos se seleccionará el cuadro de edición correspondiente al campo que se desea modificar y se editará el texto mediante el uso del teclado del dispositivo.

Si queremos confirmar los cambios realizados, debemos pulsar sobre el icono guardar que existe en el mismo lugar que estaba anteriormente el botón editar (parte superior derecha de la pantalla.



Anexo B. Figura 9: *Capturas de pantalla ver, editar y cambiar contraseña del perfil*

B.3.10. Ver el historial de plazas ocupadas

En el apartado B.3.8 se describe como acceder al menú de la aplicación, posteriormente se pulsa sobre *Aparcamientos Recientes* y se accederá a la sección con mismo nombre.

Una vez dentro de la sección se podrá interactuar con las plazas mostradas de la manera que se indica en los apartados B.3.5. y B.3.7.

Anexo B. Figura 10: *Captura de pantalla ver aparcamientos recientes*

B.3.11. Cerrar sesión

Dentro del menú de usuario (en el apartado B.3.8. se explica cómo acceder a él), pulsando sobre el rótulo *Cerrar Sesión*, se cerrará la sesión del usuario volviendo a la pantalla de *Inicio de Sesión*.

B.3.12. Ver Acerca de

Para ver el apartado *Acerca de*, seleccionar dentro del menú principal de la aplicación, el rótulo de mismo nombre.

Anexo B. Figura 11: *Captura de pantalla ver Acerca De*