Sistemas Informáticos
Curso 2012-2013

MAPA DE RECURSOS EN LA CIUDAD DE MADRID

Accesibilidad integral para el ciudadano

ANA ALFARO ORGAZ

SERGIO BALLESTEROS NAVAJAS

LIDIA SESMA SALGADO

Dirigido por:

Dra. Victoria López López

Dra. Matilde Santos Peñas

Facultad de Informática
Universidad Complutense de Madrid

Autorización de Difusión

Ana Alfaro Orgaz, Sergio Ballesteros Navajas y Lidia Sesma Salgado, alumnos matriculados en la asignatura Sistemas Informáticos, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria como el código, la documentación y/o el prototipo desarrollado en el proyecto Mapa de Recursos Ambientales, todo ello realizado durante el curso académico 2012-2013 bajo la dirección de María Victoria López López y Matilde Santos Peñas, ambas profesoras del Departamento de Arquitectura de Computadores y Automática de la Facultad de Informática.

Ana Alfaro Orgaz Sergio Ballesteros Navajas Lidia Sesma Salgado

Resumen

Mapa de Recursos Ambientales es una aplicación que ofrece a los usuarios de dispositivos móviles con sistema operativo Android una gran variedad de información sobre los recursos que proporciona el Ayuntamiento de Madrid a sus ciudadanos.

El objetivo de este proyecto es el desarrollo de una solución para que el Ayuntamiento de Madrid pueda transmitir al ciudadano esta información de forma sencilla y aprovechar mejor los recursos de los que dispone la ciudad.

Los recursos que engloba la aplicación están relacionados con los departamentos de Medio Ambiente y Movilidad de la ciudad de Madrid. Estos recursos han sido divididos en seis categorías: parques, aparcamientos (para coches, motos y bicicletas), puntos limpios (fijos, móviles y contenedores de ropa), puntos de suministro de combustibles ecológicos (bioetanol, GNC, GLP y eléctrico), rutas de bicicleta (incluyendo vías ciclistas y calles seguras), y áreas de prioridad residencial.

Para llevar a cabo el desarrollo de este proyecto se ha tenido en cuenta el reto de cumplir con los estándares que requiere el servicio de Informática del Ayuntamiento de Madrid, consiguiendo que la aplicación pueda ser auditada por el departamento de calidad de esta entidad y estar disponible para el ciudadano en un breve período de tiempo.

En general la aplicación que se ha diseñado reúne información relevante sobre los recursos mencionados mostrándolos al usuario a través de un mapa de manera unificada y sencilla, pudiendo localizar cualquier tipo de recurso en relación a su posición o a una dirección de interés.

De esta forma, se ofrece a cualquier persona que se encuentre en la ciudad de Madrid los datos necesarios para disfrutar de cada una de las infraestructuras disponibles que enriquecen la ciudad y pueden resultarle de gran utilidad en sus necesidades diarias.

Palabras clave

Aplicación móvil, Android, Medio Ambiente, Movilidad, mapa, recursos, Madrid, parques, aparcamientos, puntos limpios, puntos de suministro, combustibles ecológicos, rutas de bicicleta, Áreas de Prioridad Residencial, patrón cliente-servidor, modelo de tres capas, arquitectura SOA, Servicio Web, Spring, Hibernate, MySQL, Axis2, SOAP, WSDL, KSoap2, Google Maps, Quicksort, Maven, SOAPUI.

Abstract

Environmental Resource Map is an application that offers a big variety of information about the resources provided by the Madrid City Council for its citizens through mobile devices with Android operating system.

The goal of this project is to transmit this information from the Madrid City Council to the citizens easily and take advantage of the use of available resources in the city.

The application includes resources related to the departments of Environment and Mobility of Madrid City. These resources have been divided into six categories: parks, parking (for cars, motorcycles and bicycles), recycling points (fixed, mobile and clothes containers), green fuel stations (bioethanol, CNG, LPG and electric), bike routes (including cycle lanes and safe streets), and residential priority areas.

The most important challenge in the developing of the project has been to comply with the Computing department Madrid City Council standards requirements. As soon as possible, the application will be audited by the Quality Department of this entity and will be available for citizens in a short period of time.

In general the application contains important and detailed information about the mentioned resources and shows to the users a unified and simple map where they can locate any type of resource in relation to their position or to an interest address.

Thus, whoever stays in Madrid can obtain the necessary information to enjoy any available infrastructure that enriches the city doing a great use of them also.

Keywords

Mobile Application, Android, Environment, Mobility, map, resources, Madrid, parks, parkings, recycling points, fuel stations, green fuels, bike routes, Residential Priority Areas, client-server pattern, three-layer model, Service Oriented Architecture (SOA), web services, Spring, Hibernate, MySQL, Axis2, SOAP, WSDL, KSoap2, Google Maps, Quicksort, Maven, SOAPUI.

Dedicado a nuestra familia y amigos.

Gracias

Agradecimientos

Con este proyecto cerramos una etapa para dar paso a otra nueva. Nuestra vida laboral nos llama ya desde fuera de los muros de la Facultad de Informática, donde tantas horas y esfuerzo hemos invertido para superarnos cada día y poder alcanzar nuestra meta.

Gracias a nuestros padres, amigos y profesores que nos han apoyado desde que decidimos emprender esta aventura, hemos adquirido los conocimientos necesarios para llevar a cabo este proyecto. Durante los años que ha durado esta etapa, hemos crecido aprendiendo a enfrentar con ilusión cada obstáculo.

También queremos desde aquí, dar las gracias a Victoria López López y a Matilde Santos Peñas, por habernos dado esta oportunidad y al Ayuntamiento de Madrid por su colaboración desde el primer momento, que han hecho posible que pudiéramos lograr nuestro propósito.

"Sólo cabe progresar cuando se piensa en grande,

sólo es posible avanzar cuando se mira lejos."

José Ortega y Gasset

Prólogo

Este proyecto, Mapa de Recursos Ambientales, destaca sobre los demás de su temática por ser un proyecto totalmente finalizado y disponible para el usuario habiéndose atendido los requisitos de estandarización obligados en un cliente muy especial: Informática del Ayuntamiento de Madrid (IAM). Sólo por este motivo, nuestros alumnos Ana, Sergio y Lidia han demostrado estar perfectamente preparados para desarrollar proyectos profesionales como Ingenieros en Informática. El trabajo no ha sido fácil. Además de desarrollar una aplicación que maneja gran cantidad de información procedente de diversas fuentes y con diversos formatos, el proyecto incluye un sistema de comunicación web service específico, desarrollado con herramientas y tecnologías no estudiadas en la carrera que han representado un ambicioso objetivo inicial que se ha cumplido con total satisfacción para las partes interesadas.

Ana, Sergio y Lidia han tenido que trabajar muy duro para sacar adelante este proyecto, pero siempre lo han hecho con buen humor y una seriedad y profesionalidad poco habitual. Probablemente una de las características que ha favorecido el éxito de este grupo de alumnos es el carácter crítico que siempre han mantenido en todas sus fases. Por este motivo el proyecto resultante tiene impresa la personalidad del grupo. Han sido ellos los que han realizado la mayor parte de las propuestas (incluyendo diversos diseños en la interfaz) y los que, sin estar obligados, han decidido desarrollar todos los requisitos impuestos por el IAM con la ilusión de ver su proyecto disponible para los ciudadanos de Madrid.

Como directoras estamos muy orgullosas del resultado obtenido y deseamos un futuro lleno de éxitos para nuestros alumnos Ana, Sergio y Lidia.

Victoria López y Matilde Santos

Índice de contenidos

Capítulo 1. INTRODUCCIÓN

✓ Planteamiento y motivación

Desde la llegada de Internet a nuestras casas, los usuarios disfrutamos del acceso a una gran cantidad de información. Aunque esta ventana al mundo es una gran ventaja, también puede suponer un problema a la hora de clasificar esa información y mostrarla de la forma más ordenada posible para que al usuario le resulte realmente útil. Una forma de mostrar información concreta al usuario es a través de la aparición de las aplicaciones móviles que han cambiado el paradigma de consumo de servicios y contenidos planteando otra forma de interacción online, permitiendo a los usuarios tener una gran cantidad de información en sus bolsillos.

Hoy en día, los usuarios de Smartphones buscan aplicaciones concretas que respondan a sus necesidades diarias, canalizando la información que ofrece la web de forma útil y sencilla. Siguiendo esta filosofía, nace la aplicación "Mapa de recursos ambientales", proporcionando información de uso cotidiano al ciudadano madrileño sobre una serie de temas de interés general para hacer su día a día más fácil.

En Madrid se desplazan más de 2.000.000 de vehículos cada día [1] y el problema con los estacionamientos es cada vez más acuciante, sobre todo si el lugar de desplazamiento no es el habitual. Por eso es importante para los usuarios poder conocer la ubicación de los aparcamientos públicos de los que dispone la ciudad para cada tipo de vehículo, ya sea coche, moto o bicicleta. Además, si el desplazamiento va a realizarse dentro de la zona centro, existe otro tipo de información que debe ser tenida en cuenta y puede resultar útil a la hora de planear una ruta con un vehículo. Las Áreas de Prioridad Residencial (APR), son vías de circulación que tienen prohibido el acceso a los ciudadanos no residentes, con objeto de reducir la alta densidad circulatoria, ordenar la carga y descarga, aumentar las plazas de estacionamiento para residentes, así como disminuir los niveles de contaminación acústica y atmosférica [2]. Este tipo de vías son las grandes desconocidas dentro de la circulación de la ciudad y pueden acarrearnos alguna multa si utilizamos nuestro vehículo en alguna de las calles que se rigen bajo esta normativa.

En relación con los desplazamientos de vehículos, en Madrid se está apostando por los combustibles alternativos menos contaminantes, que poco a poco se están extendiendo gracias a las bonificaciones que se están otorgando a los usuarios de estos vehículos. Por eso, es importante dar a conocer al ciudadano la localización de los puntos de suministro de estos combustibles [3] como el bioetanol, un combustible que puede producirse a partir de un gran número de plantas, que se perfila como un recurso energético potencialmente sostenible que puede ofrecer ventajas medioambientales y económicas a largo plazo en contraposición a los combustibles fósiles, como el Gas Licuado de Petróleo, mezcla de gases licuados presentes en el gas natural o disueltos en el petróleo fáciles de licuar (en la práctica, se puede decir que los GLP son una mezcla de propano y butano), o el Gas Natural Comprimido, un combustible para uso en vehículos que, por ser económico y ambientalmente limpio, se considera una

alternativa sostenible para la sustitución de combustibles líquidos, o los más extendidos en nuestros días, los puntos de suministro eléctrico, lugares que proveen electricidad para la recarga rápida de las baterías de los vehículos eléctricos, incluyendo los vehículos híbridos.

Por otro lado, para seguir contribuyendo con un medio ambiente sostenible, se pretende promocionar el uso de la bicicleta y reducir así las emisiones urbanas de CO_2 como iniciativa beneficiosa para la ciudad. Para ello los ciudadanos que quieran apostar por este tipo de transporte necesitan conocer las infraestructuras de las que dispone la ciudad de Madrid, como son las de vías ciclistas construidas, o las calles de la ciudad que han sido catalogadas como “calles seguras” para el uso de la bicicleta [4] y aprovecharlas al máximo en función de sus necesidades.

Por último, después del gran esfuerzo que se ha hecho por concienciar a la población sobre la importancia del reciclado y de la contaminación que pueden producir algunos residuos si no se tratan adecuadamente, es importante que los ciudadanos dispongan de la información necesaria para desechar estos residuos correctamente conociendo la ubicación de los Puntos limpios más cercanos, ya sean instalaciones de Puntos Fijos para aquellos residuos que no pueden depositarse en los contenedores habituales de la calle y que, por ser peligrosos, nunca deben mezclarse con los residuos domésticos, o Puntos Móviles como camiones municipales especialmente adaptados para que los ciudadanos puedan depositar en ellos ciertos residuos peligrosos, que se sitúan a diario en lugares estratégicos de la capital, según una serie de horarios y ubicaciones fijas [5], o contenedores de ropa que gestiona la empresa Humana destinando el contenido en su gran mayoría a África, a reciclaje o tiendas Humana para que puedan elegir el que más se adapte a sus necesidades [6].

Y después de poner todos los medios para hacer de la ciudad de Madrid un lugar más saludable cada día, es importante también ofrecer al ciudadano información sobre los parques donde poder disfrutar de ella y los servicios que cada uno de ellos ofrece, como por ejemplo si dispone de zona infantil o de mayores, instalaciones deportivas o circuitos biosaludables, carril bici, “pipican” para mantener limpio el entorno cuando se permite la presencia de animales, puntos de información o aseos públicos [7].

✓ Objetivos que se pretenden alcanzar

Tras evaluar la utilidad de los recursos y la información relevante que se debía ofrecer al usuario, se ha decidido abarcar cada uno de los temas mencionados anteriormente de manera conjunta.

De esta forma se consigue ofrecer al ciudadano información a través del uso de las nuevas tecnologías, reuniendo una gran cantidad de datos que puedan resultar útiles en el día a día a cualquier persona evitando el problema de las búsquedas entre la gran cantidad de información que se encuentra en la red.

✓ Resumen de la utilidad de la aplicación

La aplicación "Mapa de recursos ambientales de Madrid", ha sido diseñada para dar acceso a los ciudadanos a una selección de información del Ayuntamiento de Madrid, a través del uso de dispositivos móviles con Sistema Operativo Android.

Permitiendo a la aplicación localizarnos mediante los servicios de ubicación que ofrecen este tipo de dispositivos, podremos obtener la posición geográfica de cualquier tipo de recurso de los mencionados anteriormente, ordenados desde el más cercano al más lejano en función de nuestra ubicación, aunque también es posible hacer una búsqueda de recursos si estamos interesados en otra dirección que no sea en la que nos encontramos actualmente. Así mismo, se permite realizar búsquedas filtradas en función de los intereses o necesidades del usuario para cada uno de los recursos. Además, cada elemento de cada categoría tiene información de interés asociada para que el ciudadano pueda conocer los detalles sobre los servicios que se le ofrecen.

Capítulo 2. ESTADO DEL ARTE

❖ Introducción

Dentro del amplio abanico de aplicaciones móviles del que disponemos en la actualidad, podemos encontrar algunas que presentan ciertas similitudes con el **Mapa de recursos ambientales.** Algunos **e**jemplos de estas aplicaciones se enumeran por categorías a continuación:

• Aplicaciones relacionadas con Puntos Limpios

- **Punto Limpio** es la aplicación que propone a los usuarios la Fundación Ecolec y que se presenta como una buena opción a la hora de localizar el punto limpio (Figura 1) más cercano de tu localidad utilizando geoposicionamiento (Figura 2). Presume de tener más de 1200 puntos limpios localizados dentro de España [8] [9].

Figura 1. Buscador

Figura 2. Mapa

- **Donde Reciclar** es otra aplicación que permite al usuario encontrar el punto de reciclaje más cercano animando al usuario a contribuir con el medio ambiente [10]. Las ventajas que añade frente a otras son principalmente dos:
    - No recoge únicamente Puntos Limpios, sino también zonas para reciclar pilas o ropa.

- Permite al usuario denunciar mediante fotografías localizadas cualquier situación que no permita a los usuarios utilizar estas áreas de reciclaje con normalidad.

- **Puntos Limpios Canarias** es la propuesta que la Fundación Canaria Recicla ofrece a los usuarios Android para localizar el Punto Limpio de una localidad perteneciente a la Comunidad (Figura 4) o el más próximo a la posición del dispositivo móvil (Figura 3)[11].

Figura 3. Opciones

Figura 4. Buscador

- **EcoParques Comunidad Valenciana** permite a los usuarios localizar los Ecoparques de la Comunitat Valenciana en su dispositivo móvil. La App para Android de FUNDACIÓN ECOLEC realizar la búsqueda del Punto Limpio de la localidad elegida dentro de la Comunidad de Valencia o el más próximo a la posición del usuario [11].

- **EcoParques Murcia** es una aplicación similar a la anterior pero orientada a la Región de Murcia que también permite localizar el Ecoparque más cercano a la posición del usuario o bien, realizar una búsqueda indicando la localidad en la que el usuario está interesado [11].

La única diferencia significativa entre las aplicaciones anteriores es el ámbito geográfico que abarcan.

- ## Aplicaciones relacionadas con Parques

- **Parques en Nantes** es una aplicación que permite obtener información sobre los 90 parques de la ciudad de Nantes [12]. Permite al usuario filtrar los parques de acuerdo con los criterios de búsqueda habituales guardados en la memoria del dispositivo móvil (Figura 5).
  A continuación, ofrece al usuario los resultados de la búsqueda de tres formas diferentes: una lista ordenada alfabéticamente, una lista ordenada del más cercano al más lejano o bien un mapa que permite al usuario apreciar la ubicación y la superficie de cada parque (Figura 6).

Figura 5.

Figura 6.

- ## Aplicaciones relacionadas con lugares de Aparcamiento

- **Parkopedia** surge de la combinación de las palabras parking y enciclopedia. Esta aplicación ofrece al usuario la posibilidad de localizar aparcamiento en 25 países distintos en todo el mundo gracias a las contribuciones de los conductores [13].

  Se basa en la geolocalización indicando el más cercano, o bien introduciendo la dirección cerca de la cual desea encontrar aparcamiento (Figura 7). Además, si el usuario lo desea, calculará la ruta más corta hasta el aparcamiento (Figura 8). Los planes de futuro de esta aplicación contemplan incluir la funcionalidad de mostrar la disponibilidad de plazas libres de cada zona aparcamiento en tiempo real, los precios, el horario y la búsqueda filtrada de los aparcamientos en función de las necesidades de los usuarios, pero a día de hoy, es sólo un proyecto.

  Por otro lado, para aumentar su fuente de información, dan la posibilidad al usuario de enviar un e-mail si éste localiza alguna zona sin cubrir o bien, subir una foto con la señal de parking o la lista de precios para añadir un nuevo aparcamiento.

Figura 7.

Figura 8.

- **Aparcamientos España** es una aplicación sólo dedicada a la búsqueda de aparcamiento a través del uso de Google Maps y GPS. Con ella se puede buscar aparcamientos en las proximidades del usuario y automáticamente inicia el sistema de navegación para realizar el aparcamiento. Además incluye un buscador para consultar las localizaciones de los aparcamientos en cualquier otro lugar que no sea cercano a la posición del usuario [14].

- ## Aplicaciones relacionadas con el uso de bicicletas

- **Spotcycle:** aplicación multi-ciudad que, a través del uso del GPS, permite a los usuarios disfrutar de su experiencia en bicicleta gracias al seguimiento de los estados de estaciones de bicicletas que se encuentran a su alrededor y poder planear su ruta hasta su destino [15].

Entre sus funcionalidades principales incluye:

- Muestra la disponibilidad de bicicletas y lugares en las estaciones de bicicletas donde poder estacionarla (Figura 9).
- Para facilitar la información en tiempo real con cierta fiabilidad, muestra la hora de la última actualización de los datos mostrados.
- Automáticamente se enumeran las 10 estaciones más cercanas a la ubicación actual del usuario (Figura 10).
- También dispone de búsquedas filtradas que muestran las 10 estaciones más cercanas a la ubicación del usuario que cumplen los criterios de la búsqueda.
- Dispone de una función por capas donde también muestra los carriles bici junto con las estaciones donde se encuentran pero sólo para ciudades “compatibles”.
- Además permite crear y guardar rutas de bicicleta personalizadas o temáticas permitiendo al usuario incluir puntos de interés y descripciones o comentarios sobre las rutas de bicicleta y compartirlas a través de las redes sociales.
- Por último, la aplicación incluye la posibilidad de personalizar las vistas, servicio de radio y reloj con servicio de alarma para avisar al usuario si lo desea de que el tiempo de alquiler está finalizando.

Esta aplicación sólo se presenta en francés e inglés, en España todavía no podemos disfrutar de ella.

Figura 9.

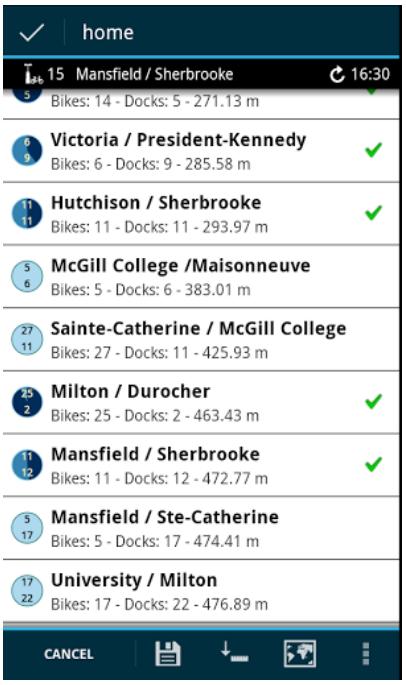


Figura 10.

- **Cycle Hire Widget** es una aplicación que proporciona al usuario las facilidades necesarias para encontrar la estación de alquiler de bicicletas más cercana a través del mapa (Figura 11), o bien mediante una lista (Figura 12), en Barcelona (Bicing), Valencia (ValenbiSi), Santander (Tusbic), Sevilla (Sevici), Zaragoza (Bizi) y otras 35 ciudades en todo el mundo. Tiene varios reconocimientos y ha sido mencionada en artículos a nivel internacional. Dispone de una versión gratuita y otra de pago [16].

Figura 11.

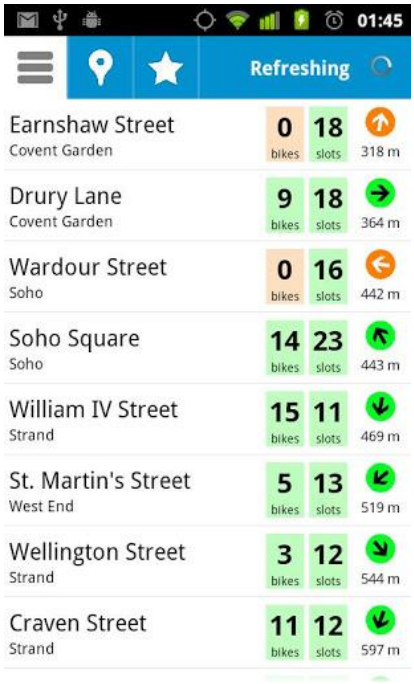


Figura 12.

- **Bici Bike Valencia Lite** es una aplicación para los usuarios de bicicleta dentro de la ciudad de Valencia que incluye los carriles bici y las ciclocalles de la ciudad. Contiene información sobre el servicio de bicicletas y aparcabicis Valenbisi en tiempo real, los aparcabicis públicos y localiza el servicio de alquiler, taller o venta más cercano al usuario. Además ofrece la posibilidad de memorizar dónde aparcó su bici el usuario y guiarlo hasta ella.
  A través de esta aplicación se puede consultar la ordenanza municipal y las recomendaciones para ciclistas. Proporciona mapas online y offline para usuarios que no disponen de tarifa de datos en su dispositivo [17].

- **Carril Bici Barcelona** esta aplicación es muy simple, sólo permite al usuario descargar en su dispositivo un mapa que contiene todos los carriles bici de Barcelona [18].

- ## Aplicaciones relacionadas con Puntos de Recarga

- **Electromaps**: permite ver los puntos de recarga para vehículos eléctricos más próximos a tu ubicación, proporcionando una lista (Figura 14) o mostrándolos en el mapa (Figura 13). Además ofrece información del punto de recarga como la dirección, el precio de la recarga, el precio del parking, fotos asociadas… Permite al usuario consultar los comentarios que hayan dejado otros usuarios o añadir otros nuevos. Incorpora Google Maps Navigation para guiar al usuario a uno de los puntos si así lo desea [19].

Figura 13.

Figura 14.

- **Recargo** proporciona al usuario la posibilidad de encontrar puntos de recarga de vehículos eléctricos más cercanos, planificar viajes y compartir fotos para mantenerse comunicado con la comunidad de la aplicación. Ayuda al usuario obtener las direcciones utilizando sus navegadores favoritos, guardar ubicaciones para la planificación de los viajes y realizar búsquedas de puntos de recarga aplicando un filtro por tipo de enchufe y red de carga, además de consultar si los cargadores están en uso, disponibles o fuera de servicio [20].

- **ElectriCar:** es una aplicación también para la localización de puntos de recarga que se encuentran dentro del territorio español que permite visualizar los puntos en Google Maps y calcular la ruta al punto de recarga elegido [21].

- **ChargeLocator Iberia** es una aplicación para localizar los puntos de carga para un vehículo. Está disponible para España, Andorra y Portugal. Dispone de versión gratuita y otra mucho más completa [22].

  Incluye entre sus funcionalidades la localización del punto de recarga más cercano o en las proximidades de la dirección que se especifique en el buscador (Figura 16), realizar búsquedas filtradas en función de los intereses del usuario (Figura 15) y qué punto de recarga tiene la tarifa más barata proporcionando el teléfono del punto de recarga para consultar el estado.

  Además el usuario puede solicitar que se le calcule la ruta hasta el punto de recarga seleccionado, utilizar StreetView en caso de no identificarlo al llegar y realizar búsquedas aplicando filtros.

  Los desarrolladores de esta aplicación comenzaron gracias a la colaboración de los usuarios y siguen solicitándola para completar sus datos.

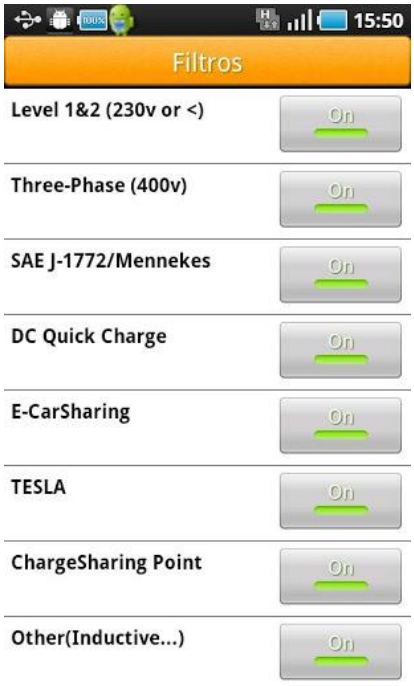


Figura 15.

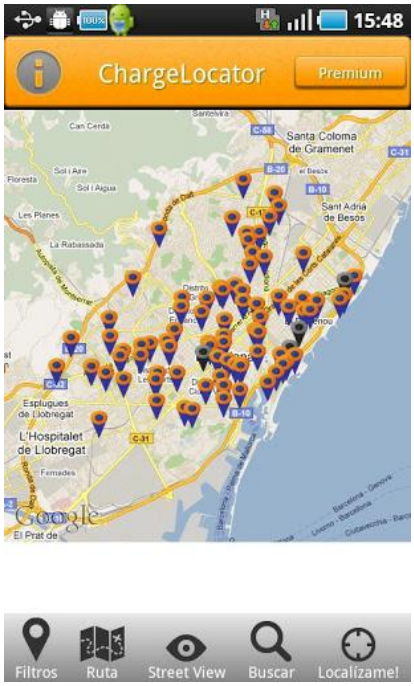


Figura 16.

❖ Mapa de Recursos Ambientales

Cada una de las aplicaciones mencionadas ofrece un servicio concreto con una gama más o menos amplia de funcionalidades interesantes para el usuario.

En este caso, no nos centraremos en ofrecer al usuario un único tipo de servicio. Desde la aplicación del Mapa de Recursos Ambientales se ofrece al usuario la posibilidad de utilizar en la misma aplicación los servicios mencionados anteriormente en aplicaciones independientes, de manera que no sea necesario disponer de una aplicación distinta para cada uno de los recursos que los usuarios quieran localizar en cada momento.

Para ello, desde el grupo de desarrollo hemos seleccionado las funcionalidades que se pueden ofrecer al usuario de manera conjunta para cada tipo de servicio ya que, aunque unas funciones podrían parecer interesantes en una aplicación orientada a un único tipo de servicio, a la hora de englobar todos los recursos en una misma aplicación podrían carecer de interés. De esta forma le otorgamos la uniformidad y sencillez necesaria para que la utilización de la aplicación resulte cómoda y fácil de usar para cualquier usuario.

Capítulo 3. ESPECIFICACIÓN DEL PROYECTO

3.1. REQUISITOS

❖ Especificación de requisitos

• Requisitos funcionales

Los requisitos funcionales definen el comportamiento interno del software. Son las características requeridas del sistema que se encuentran implementadas por medio de la escritura, compilación y ejecución de líneas de código.

Son requisitos que denotan cada funcionalidad del sistema y de qué forma el usuario puede interactuar con ellas.

- **Conexión Wifi o 3G** – La aplicación precisa de conexión a Internet desde el primer momento de su utilización. En caso de no disponer de esta conexión, el usuario será avisado mediante una notificación.

- **Activación de servicios de ubicación mediante redes inalámbricas** – La aplicación exige activar la ubicación por redes inalámbricas dentro de los ajustes del dispositivo móvil, ya que se requiere la localización del usuario para ofrecerle los recursos más cercanos a él. En caso de no estar activada esta función en el terminal, el usuario recibe una notificación solicitando su activación.

- **Sección de Ayuda** – Un breve manual de usuario ofrece información sobre el contenido y el manejo general de la aplicación.

- **Registro del usuario** – Es necesario que el usuario registre sus datos antes de acceder al contenido de la aplicación. Este registro se ha diseñado a petición del Ayuntamiento de Madrid como cliente para la elaboración de futuras estadísticas sobre el uso de la aplicación por los distintos perfiles de usuario. Además se informa al usuario de los términos legales respecto al uso de sus datos personales.

- **Login de acceso** – Tras el registro, el usuario podrá acceder al contenido de la aplicación a través de su correo electrónico y su contraseña.

- **Recordar Contraseña** - Si el usuario olvida la contraseña que utilizó durante el Registro podrá recuperarla recibiendo un email en su dirección de correo electrónico.

- **Modificación de datos del registro** – El usuario podrá modificar sus datos de registro en la aplicación en cualquier momento.

- **Acceso a páginas web de los colaboradores** – A través de la imagen corporativa del Ayuntamiento de Madrid, el usuario podrá acceder desde la aplicación a la página web del Ayuntamiento de Madrid (www.madrid.es), que se abrirá en un navegador, externo a la propia aplicación.

- **Localizar recursos por cercanía** – A través de un mapa se mostrarán todos los recursos de cada categoría ordenados del más cercano al más lejano, tomando como referencia la posición geolocalizada del usuario y permitiéndole visualizarlos en intervalos de 10 en 10.

- **Filtrar búsqueda por preferencias** – En caso de que el usuario quiera hacer una búsqueda concreta en función de sus necesidades, el botón de preferencias le permitirá hacer una selección de opciones para definir sus intereses.

- **Búsqueda por dirección** – Además, de forma análoga a la localización por cercanía, se permite hacer una búsqueda de recursos cercanos a una dirección introducida por el usuario en la aplicación. En este caso se tomará como punto de referencia la dirección introducida y se mostrarán los recursos ordenados por cercanía en intervalos de 10.

- **Ampliación de la información de un recurso** – Cada recurso localizado en el mapa tiene una información más detallada asociada, que el usuario podrá visualizar cuando seleccione un recurso concreto y le informará de los servicios que ofrece el recurso de forma ampliada.

- **Mapas de las redes ciclistas** – Existen dos mapas distinguidos en la ciudad de Madrid para el transporte en bicicleta. El usuario podrá seleccionar uno de ellos para visualizar sobre el mapa la araña ciclista en la que esté interesado.

- **Delimitación de APRs** – Sobre el mapa se ha delimitado la zona que se rige bajo las normas de las “Áreas de Prioridad Residencial”. El usuario podrá visualizar la leyenda sobre el mapa para distinguir los límites de la zona, el área comprendida dentro de esos límites y las calles de libre circulación dentro del área delimitada.

- **Información APRs** – Debido al desconocimiento por parte de los ciudadanos sobre la existencia de estas zonas, se ha incluido en la aplicación un apartado en el que se explica la normativa que rige en estas áreas.

- **Información Puntos de Suministro** – Al hacer referencia a varios tipos de puntos de suministro poco habituales en la actualidad, se proporciona información sobre cada uno de los combustibles alternativos que se ofrecen en la aplicación para todo aquel que los desconozca.

- **Información sobre Puntos Limpios** – Si el usuario decide hacer uso de la opción de puntos limpios y desconoce cuál es su utilidad o en que se diferencian, también podrá acceder a esta información desde la aplicación para hacer uso del que más se adecúe a sus necesidades.

- **Información sobre residuos permitidos** – En el caso de los puntos limpios, se ha añadido información general para cada tipo de punto limpio, explicando cuáles son los residuos y las cantidades que se permiten en cada uno de ellos.

- ## Requisitos no funcionales

Los requisitos no funcionales son características requeridas del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo sin aportar nuevas funcionalidades. Los requisitos no funcionales complementan a los requisitos funcionales anteriores.

- **Tamaño:** Aunque cada día aparecen dispositivos móviles con mayor capacidad de almacenamiento, hay que tratar de reducir al máximo el tamaño de la aplicación para no sobrecargarlos. Para ello se ha tratado de optimizar tanto la carga gráfica de la aplicación reduciéndola al mínimo sin privar al usuario de una buena calidad de imagen, como cada una de las funcionalidades de las que ofrece la aplicación.

- **Eficiencia:** La aplicación se ajusta a un consumo de recursos del dispositivo móvil reducido, ya que no se solicita al usuario la activación de ningún recurso extra que no sea imprescindible para el funcionamiento correcto de la aplicación.

- **Rendimiento:** Se ha tratado de agilizar al máximo todos los tiempos de carga utilizando algoritmos de optimización para la ordenación de datos para que el usuario pueda obtener la información de forma dinámica.

- **Extensibilidad:** La aplicación permite ampliaciones con nuevas funcionalidades como la adaptación de un navegador para trazar la ruta hasta un recurso o marcar recursos como favoritos.

- **Facilidad de uso:** La aplicación se ha diseñado con un entorno visual sencillo que la hace intuitiva y de fácil manejo. Además, se han establecido comportamientos similares en las distintas categorías de recursos para un aprendizaje rápido desde el primer uso.

- **Fiabilidad:** El sistema ha sido probado en innumerables ocasiones para contemplar cada uno de los errores que podría cometerse durante el uso de la aplicación logrando abarcar un gran número de situaciones en las que la aplicación se recupera favorablemente tras un comportamiento inesperado por parte del usuario.

- **Seguridad:** Los datos que se solicitan al usuario son mínimos, evitando que tenga que proporcionar durante el registro datos como su nombre, apellidos, dirección, DNI o cualquier otro dato personal más significativo.

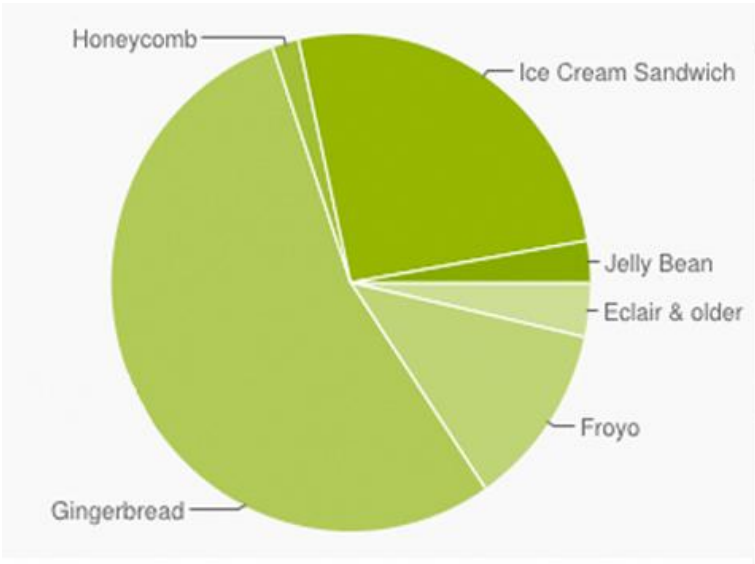
- **Institucionalidad:** El Ayuntamiento de Madrid como cliente solicitó el uso de imágenes corporativas oficiales y representativas de la institución.

3. 2. DISEÑO

Actualmente los nuevos dispositivos móviles llamados smartphones o móviles inteligentes han entrado en la vida de un gran número de personas en la sociedad. Cualquier persona que posea este dispositivo lo puede utilizar como una nueva forma de acceso a internet, una nueva ventana a toda la información almacenada en internet. Además este acceso cada vez se realiza de una forma más sencilla y más llamativa al usuario, gracias a muchos desarrolladores que se esfuerzan para que esto sea posible.

Por este motivo, el Ayuntamiento de Madrid pretendía facilitar al ciudadano la búsqueda de información relacionada con la ciudad, más concretamente información asociada a varios recursos pertenecientes al ámbito medioambiental y de circulación. Así surgió la idea de crear una aplicación móvil para dispositivos con sistema operativo **Android**. Esta aplicación debía ofrecer información relevante para el ciudadano de una forma cercana y fácil de entender.

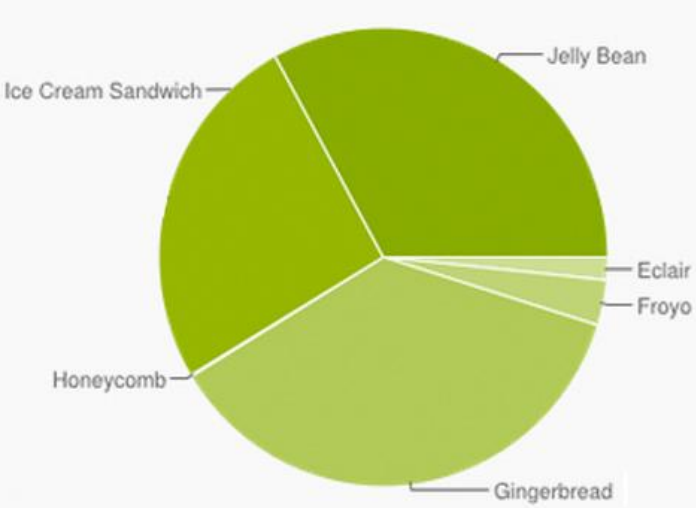
Primero analizamos la versión de Android que usaríamos. Según el informe de Google de noviembre de 2012 [23], sólo un 3.5% de los usuarios usaban la versión Eclair 2.1 o inferiores. La siguiente versión, Froyo 2.2, era usada por un 12%, lo que puede llegar a ser un número de usuarios bastante considerable, y por ello decidimos que la mínima versión en la que nuestra aplicación se puede ejecutar debía ser esta versión que se corresponde con la API 8. La versión más alta que usaríamos sería la versión Android Jelly Bean 4.1 que se corresponde con la API 16. En el siguiente gráfico podemos observar el uso del resto de versiones:



Version	Codename	API	Distribution
1.5	Cupcake	3	0.1%
1.6	Donut	4	0.3%
2.1	Eclair	7	3.1%
2.2	Froyo	8	12%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	53.9%
3.1	Honeycomb	12	0.4%
3.2		13	1.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	25.8%
4.1	Jelly Bean	16	2.7%

Figura 17. Uso de versiones Android (noviembre 2012)

A finales del mes de noviembre del 2012 salió la versión de Android 4.2 que se corresponde con la API 17, así que tuvimos que actualizar a esta nueva versión para poder llegar también a los dispositivos más modernos. En el siguiente gráfico podemos observar cómo ha quedado el uso de las versiones de Android a fecha de mayo de 2013 según un informe de Google [24]:



Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.5%
2.2	Froyo	8	3.2%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	36.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	25.6%
4.1.x	Jelly Bean	16	29.0%
4.2.x		17	4.0%

Figura 18. Uso de versiones Android (junio 2013)

En este gráfico podemos observar que el uso de las versiones de IceCremSandwich (4.0) y de JellyBean (4.1 y 4.2) ha crecido considerablemente en tan sólo 7 meses.

La aplicación Android se llamaría Mapa de Recursos de la Ciudad de Madrid. Un mapa nos permitiría indicar los puntos correspondientes a cada recurso que el usuario quería encontrar. Integrar Android con una API que nos proporcionase un mapa nos llevó inevitablemente a **Google Maps**.

Google Maps es un servicio gratuito de aplicaciones de mapas que ofrece Google. Además, ofrece la posibilidad de que cualquier desarrollador Android pueda integrarlo en su aplicación. Los detalles sobre esta integración se expondrán más adelante. Para el caso de las rutas de bicis se comentarán diversos problemas encontrados y la transición hasta poder incluir esta información dentro de la API de Google Maps [25].

Vista la API que usaríamos para el mapa, no podíamos mostrarle al usuario información sobre todos los recursos a la vez, ya que se incluirían 14 recursos de distinto tipo y a su vez de cada uno se obtendrían cientos o miles puntos en algunos casos. Por ello primeramente decidimos diferenciar entre seis tipos de recursos, agrupados en: parques, aparcamientos, puntos limpios, puntos de suministro, rutas de bici y áreas de prioridad residencial. Además dentro de cada tipo de recurso podríamos diferenciar también otras secciones como, dentro de puntos limpios, los puntos limpios fijos, los móviles y los contenedores de ropa; dentro de aparcamientos, los de coches, de motos y los de bicis; dentro de puntos de suministros, los de bioetanol, los de gnc, glp, y eléctricos; y dentro de las rutas de bicis, las calles seguras y las vías ciclistas. De esta forma el usuario elegiría qué tipo de recurso quería buscar inicialmente, para poderle ofrecer una información más precisa.

Aún así, mostrarle todos los recursos asociados a esa búsqueda en el mapa directamente podía incluir mucho ruido, y que el usuario al final no supiese qué recurso podía ser el más adecuado para sus necesidades. De esta forma, primero debíamos localizar la ubicación del usuario y, con esta información, mostrarle sólo los recursos más cercanos a él, ya que son los que más le podría interesar.

Dos de las formas que nos ofrecía Android para localizar al usuario eran vía gps o vía redes de internet como wifi o 3G. Vía gps sólo funcionaba correctamente en exteriores. Su uso en interiores podría ser nulo o suponía una larga espera que no podíamos permitir en una aplicación móvil. Por ello nos decantamos por localizar al usuario vía redes de internet. De esta forma podía funcionar tanto en interiores como en exteriores, y tanto con conexión wifi como 3G. Además la espera que supone esta localización es mínima.

Cuando el Ayuntamiento de Madrid nos proporcionó toda la información asociada a cada recurso vimos que, por ejemplo, había que incluir 1015 aparcamientos de bicis o 411 aparcamientos de moto, por lo que seguía siendo demasiada información a mostrar al usuario de una sola vez en el mapa. Por ello, y gracias a la disposición de la ubicación del usuario, decidimos calcular en cada búsqueda la distancia existente entre el usuario y cada recurso buscado. De esta forma podíamos aplicar un algoritmo de ordenación como **Quicksort** para obtener los recursos ordenados de menor a mayor distancia con el usuario. Quicksort es un algoritmo basado en la técnica de “divide y vencerás”, que hemos estudiado y aprendido durante la carrera, que permite ordenar *n* elementos en un tiempo promedio de *n log n*.

Una vez ordenados los recursos de menor a mayor cercanía con el usuario, se le muestran solamente los 10 más cercanos a él, aunque el usuario tiene la opción de moverse por todos los recursos de menor a mayor distancia gracias a unos botones en forma de flecha. Así podrá visualizar de 10 en 10 los recursos obtenidos en su búsqueda ordenados por cercanía, para que la información sea clara y pueda encontrar el recurso acorde a sus necesidades.

Se nos ocurrió pensar que el usuario podría no sólo querer buscar los recursos más cercanos a su localización, sino también buscar los recursos más cercanos a una dirección específica. Por ello incorporamos esta nueva funcionalidad gracias a la clase **Geocoder**. La función de esta clase, que proporciona Android, es manejar la geocodificación y la geocodificación inversa, es decir, a partir de una dirección dada obtendremos las coordenadas geográficas decimales de ésta y viceversa. Esto nos simplificó el trabajo puesto que sólo deberíamos pedir al usuario el tipo de vía, el nombre de la vía y el número (opcional), y esta clase nos proporcionaría las coordenadas de esa dirección. Luego ya podíamos volver a calcular la distancia desde ese punto a cada recurso, volver a aplicar el algoritmo Quicksort y mostrarle la información al usuario.

La aplicación necesita un gran número de datos para mostrárselos al usuario. La forma más organizada de almacenar esta información es en una base de datos. Además la aplicación debía ofrecer un registro/login del usuario para que el ayuntamiento pudiese recabar datos sobre el uso de la aplicación. Por lo tanto necesitábamos una base de datos y un servidor donde almacenar esa base de datos.

El servicio de Informática del Ayuntamiento de Madrid (**IAM**) se puso en contacto con nosotros para imponernos una serie de condiciones y estándares que debía cumplir el servidor y la base de datos, ya que la aplicación debía pasar un posterior control de calidad para poderla distribuir a todos los ciudadanos de Madrid.

De esta forma, el IAM nos ahorró bastante trabajo al no tener que valorar qué frameworks nos podrían ofrecer las mayores ventajas o cuáles se adaptarían mejor a nuestro proyecto. Sin embargo, estábamos ante un gran reto tecnológico. Eran demasiados conceptos y frameworks desconocidos para nosotros, y además la primera reunión con la IAM se produjo a mediados del mes de febrero, por lo que sólo disponíamos aproximadamente de unos tres meses y medio para vencer este reto. Pero dicen que quien no arriesga no gana, y si no lo intentábamos, siempre nos quedaría la duda de si hubiésemos sido capaces de conseguir este reto. Así que solo nos quedo decir ¡manos a la obra!

Los estándares a seguir eran primeramente un patrón cliente-servidor, donde el cliente sería la propia aplicación móvil y el servidor contendría la base de datos, que opera en **MySQL**, y toda la lógica de negocio del sistema.

El sistema además debía estar basado en un modelo de tres capas, que eran:

- Capa de persistencia: encargada del acceso a la información de la base de datos, debía estar implementada con el framework **Hibernate**.
- Capa de negocio: recoge toda la lógica del sistema, debía ser implementada con **Spring**.
- Capa de presentación: presentación del proyecto al usuario. Se trata de la propia aplicación móvil **Android**.

Además de estas tres capas, es muy importante la capa de integración ya que es la que permite al cliente realizar una conexión con el servidor, vía internet. Esta capa está implementada con el framework **Axis 2** que implementa **WSDL** (Web Services Description Language) para exponer el código del servidor como un web service. Este servicio web además emplea el **protocolo SOAP**, sobre HTTP. De esta forma se crean servicios para el acceso final a la base de datos (tanto para obtener como almacenar nuevos datos) que el servidor expone al cliente, y éste puede conectarse a este WSDL para consumirlos, determinando qué funciones puede usar y cómo debe hacerlo.

Visto todo lo que debíamos investigar, lo primero que necesitábamos para llevarlo a cabo era paciencia, y dedicamos muchas horas de investigación mirando tutoriales, blogs, libros… tanto en español como en ingles. Hemos de reconocer que internet y la gran cantidad de desarrolladores que aportan sus ejemplos y proyectos nos han facilitado de una forma u otra este reto. Además buscando información acerca de estas tecnologías muchas veces nos topábamos con ofertas de trabajo que requerían Ingenieros con estos conocimientos, lo cual nos motivaba aún más para conseguirlo.

El primer paso fue conocer las tecnologías y frameworks que debíamos usar, conceptos teóricos y ejemplos sencillos. El segundo paso fue comenzar a instalar todo el entorno en nuestros portátiles para así poder trastear con ejemplos y proyectos ya realizados por otros

desarrolladores, lo que nos permitiría ir conociendo un poco más el funcionamiento de todas las tecnologías y la integración entre ellas. De esta forma conocimos el framework **Maven**. Hablaremos sobre él más adelante, pero a grandes rasgos Maven es una herramienta de gestión de proyectos la cual ofrecía muchísimas ventajas en cuanto a la gestión de librerías, y verdaderamente debíamos manejar muchas librerías por lo que la necesidad de usar esta herramienta fue indiscutible.

Pasado un tiempo de investigación de todas las herramientas que la IAM nos demandaba, comenzamos a entender lo que en verdad nos pedían. Nuestra primera meta fue entonces el login de la aplicación. No el login plenamente conectado al cliente (estábamos muy lejos de eso), sino el login como una acción mediante la cual gracias a dos variables de tipo string, que hacían referencia a un email y una contraseña de usuario, pudiese acceder a una tabla de usuarios contenida en la base de datos y consultar si existía algún registro que coincidiese con ese email y la contraseña. Obviamente ese registro había sido previamente insertado "a mano" en la base de datos por nosotros. Luego necesitamos una herramienta para la gestión de la base de datos **MySQL**, lo que nos llevó a usar **SQL Manager for MySQL**. Es una herramienta para manejar bases de datos, sencilla y fácil de usar, pero dándonos todas las ventajas que nos hacían falta. Además para ello necesitábamos una herramienta que nos permitiese levantar el servicio de MySQL de manera local en nuestro ordenador. Elegimos el panel de control **Xampp** [26], de software libre y que con un par de clicks nos daba acceso al servicio MySQL. Esta herramienta también nos proporcionaba un manager llamado PHPMyAdmin, pero preferimos continuar con el uso de SQL Manager ya que nos habíamos hecho a su funcionamiento, y creíamos que podría ofrecernos mayores ventajas a largo plazo.

Creado un proyecto de Eclipse, solucionadas las dependencias con las librerías que necesitábamos, configurados Spring e Hibernate, y realizado el código pertinente para realizar este primer "login", tuvimos que conectar este proyecto con la base de datos. SQL Manager nos ayudó en esta tarea, poniendo a nuestra disposición un usuario y contraseña necesarios para este acceso, y una url *jdbc:mysql://localhost:3306/maparecursos*. Una vez conectados, también necesitábamos un servidor web donde desplegar el proyecto y comprobar su correcto funcionamiento. Elegimos **Apache Tomcat** [27] que es un servidor HTTP de software libre gestionado por la fundación Apache.

De esta forma, Maven nos permitía plegar el proyecto como un **archivo .war** [28] para después desplegarlo en Tomcat. Un archivo .war es un archivo de aplicación web (Web Application Archive) cuya función es combinar varias partes de una aplicación en un solo archivo. Sólo debíamos buscar este archivo, generado por Maven al compilar nuestro proyecto, en la carpeta target contenida en el directorio raíz del proyecto; después copiamos este archivo, y lo debíamos pegar en la carpeta webapps contenida dentro del directorio de instalación de tomcat. A continuación, mediante comandos por consola, desplegábamos este archivo .war. ¿Y ahora como podíamos ver si nuestro "login" funcionaba correctamente o no? Necesitábamos un cliente. Pero no podíamos conectar nuestra aplicación Android desarrollada de forma paralela al servidor, ya que requería otra nueva investigación para ello, y podría añdir errores y no dejarnos cumplir nuestro objetivo en esta fase, que era conectarnos a la base de datos mediante este primer "boceto" de servidor. De esta forma conocimos **SoapUI**.

SoapUI [29] es una aplicación muy versátil que nos permite probar, simular y generar código de servicios web de forma ágil, partiendo del contrato de los mismos en formato WSDL y con vínculo SOAP sobre HTTP. Justo lo que buscábamos y además disponía de una versión freeware. De forma coloquial, SoapUI nos permitía crear un "cliente virtual" para probar nuestro servicio. Una vez instalado, solo debíamos crear un nuevo proyecto. Dentro de él nos requerían un wsdl, una url a la que realizar la conexión, del tipo *localhost/MapaRecursosWS/services/UsuarioWebService?wsdl*.

Introducida la url, SoapUI nos da a conocer la petición que nos surte ese servicio, es decir, las funciones que podremos consumir. De esa forma, podremos crear una nueva petición o request, y ésta nos pedirá los datos necesarios para realizarla. Una vez introducidos esos datos, podemos pulsar en el botón de ejecutar y obtener la respuesta del servicio web [30].

Una vez realizadas todas las pruebas y ver que ese método "login" funcionaba correctamente, podíamos seguir implementando los demás accesos a la tabla de usuarios, como dar de alta un usuario, mirar cuántos usuarios hay en la tabla, modificar datos de un usuario, etc…

Una vez terminado, comenzamos a realizar el mismo proceso para cada recurso, además de ir diseñando la base de datos de la que se hablará más adelante. El acceso más común para las tablas de los recursos era que nos devolviese todos los registros que contuviese, para luego tratarlos dentro de la aplicación Android.

Cuando tuvimos todas las tablas y los servicios implementados y funcionando de manera local, debíamos investigar como subirlo a internet, es decir, necesitábamos un host gratuito donde almacenar esa base de datos y el archivo .war del proyecto.

En internet existen muchos hosting gratuitos, pero no todos ofrecen las funcionalidades que deseábamos, así que finalmente nos decantamos por **Cloudbees** (http://www.cloudbees.com). Cloudbees nos permite subir nuestro archivo .war y desplegarlo en cloud como servicios, además de alojar bases de datos MySQL en sus servidores. De esta forma basta con subir el archivo .war y él solo lo despliega. Para la base de datos simplemente creamos una mediante la web y nos proporcionan la url de almacenamiento de la misma, y se crea un administrador con usuario y contraseña. De nuevo el software SQL Manager para MySQL nos permitía introducir esta url y los datos de acceso a ella para conectarnos a la base de datos alojada en internet. Sólo tuvimos que duplicar las tablas a esta nueva base de datos y listo. Además gracias a la url/wsdl que nos proporciona cloudbees para la conexión a los servicios, también pudimos usar SoapUI para volver a comprobar que todas las operaciones funcionaban correctamente, pero ya alojadas en internet, y no de manera local como hasta ahora.

Hecho esto, teníamos la aplicación Android funcionando por un lado, metiendo recursos "a mano" para poder probar su correcto funcionamiento, y por otro lado el servidor y la base de datos, funcionando correctamente, lo cual nos habíamos asegurado mediante pruebas a través de SoapUI. El siguiente paso era unir estos dos sistemas, es decir, debíamos conectar el cliente Android al servidor para que a través de él pudiese acceder a la información almacenada en la base de datos.

De esta forma comenzamos a investigar cómo consumir servicios web SOAP en Android, y nos encontramos con una librería llamada **kSOAP2-android**. kSOAP2 [31] es una librería eficiente para servicios web basados en el protocolo SOAP. Añadimos esta librería a nuestro proyecto Android y mediante objetos de tipo SoapObject y HttpTransportSE, realizamos la llamada al webservice. Una vez realizada la llamada obteníamos los resultados también mediante un objeto SoapObject. Luego creamos diferentes clases y métodos para acceder a cada servicio asociado a cada tabla de la base de datos, y simplemente teníamos que llamar a estos métodos desde nuestras otras clases que requerían la información de la base de datos. Más adelante se explicará cómo se implementa esta conexión más detalladamente.

Una vez hecho esto ya teníamos el proyecto casi terminado. Nos dedicamos a hacer muchas pruebas para que todo funcionase correctamente y no surgiesen errores imprevistos, así como probar la aplicación en diferentes terminales.

Todas estas tecnologías mencionadas están explicadas en el siguiente capítulo para conocer cómo funcionan y en qué se basan. Además en el capítulo de desarrollo de la aplicación se explica cómo hemos usado nosotros esas tecnologías para el funcionamiento de nuestro proyecto.

3.3. Herramientas Software usadas

- Eclipse JUNO
- SDK Android
- Librerias dentro del cliente:
    - google-play-services (para el uso de la API Google Maps)
    - android support v4 (para que los mapas v2 funcionen desde la versión 2.2 de Android)
    - ksoap2 android assembly 2.4 (para la conexión del cliente con el web service)
    - activation, additionnal y mail (3 librerías para el envío del mail en caso de que el usuario quiera recuperar su contraseña)
- JDK versión 1.6
- Apache Tomcat 7.0.34
- My SQL connecto 5.1.9
- SQL Manager Lite for MySQL
- Spring 2.5.6
- Hibernate 3.3.0
- Maven plugin 2.3.1
- Axis 2 1.4.1
- SoapUI 4.5.1
- JScience 4.3.1
- CloudBees como PaaS

3.4. Herramientas Hardware usadas

Para probar la aplicación, hemos hecho uso de varios dispositivos móviles y tablet de forma que hemos podido comprobar la adaptación de todas las interfaces a distintos tamaños de pantalla. Los modelos usados son: Samsung Galaxy Ace 2, Nexus 4, Samsung Galaxy S3, Sony Ericsson Xperia Arc S, Sony Ericsson Xperia Neo V, Samsung Galaxy Mini, y la tablet Samsung galaxy Tab 2, entre otros.

Figura 19. Prueba de la aplicación en un dispositivo móvil

Figura 20. Prueba de la aplicación en un tablet

Capítulo 4. TECNOLOGÍAS UTILIZADAS

4.1. PATRÓN CLIENTE SERVIDOR

El **modelo arquitectónico cliente-servidor** [32] es un modelo en el que un sistema se organiza como un conjunto de servicios y servidores asociados, más unos clientes que acceden y usan esos servicios. Los principales componentes de este modelo son:

- Un conjunto de **servidores** que ofrecen servicios a otros subsistemas. En este proyecto se ha desarrollado un servidor basado en un entorno empresarial J2EE cuyas características se explicarán más adelante, y que ofrecerá varios servicios.
- Un conjunto de **clientes** que llaman a los servicios ofrecidos por los servidores. En este proyecto los clientes serán clientes Android accediendo al servidor.
- Una **red** que permite a los clientes acceder a estos servicios. Esto no es estrictamente necesario ya que los clientes y los servidores podrían ejecutarse sobre una única máquina. Sin embargo, en este proyecto los clientes y el servidor se ejecutan sobre diferentes máquinas, por ello la red que permite la conexión será internet.

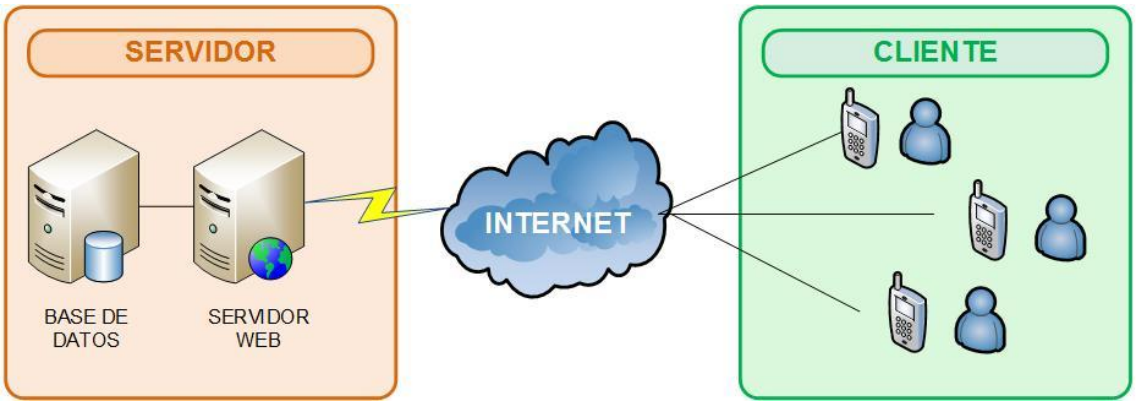


Figura 21. Esquema patrón cliente-servidor

Los clientes pueden conocer los nombres de los servidores disponibles y los servicios que éstos proporcionan. Sin embargo, los servidores no necesitan conocer la identidad de los clientes o cuántos clientes tienen, ni los propios clientes necesitan conocer la existencia de otros clientes. Es decir, los clientes Android acceden a los servicios proporcionados por un servidor comunicándose mediante un modelo de interacción basado en el diálogo a través de **Servicios Web** [33]. De forma más sencilla, un cliente Android realiza una petición a un servidor y espera hasta que recibe una respuesta.

Las **ventajas** del uso del patrón cliente-servidor son:

- Datos almacenados en el lado del servidor, por lo que se consigue un mayor control de seguridad, control de acceso a datos y consistencia de la información.
- Facilidad de mantenimiento y actualización del servidor, de forma que los beneficios los obtienen múltiples clientes generalmente sin necesidad de que éstos actualicen nada.
- Recursos centralizados que pueden ser usados a través de múltiples clientes.

Las **desventajas** del uso del patrón-cliente servidor son:

- Cuando se encuentran muchos clientes que hacen peticiones el servidor se puede congestionar.
- El cliente es muy dependiente del servidor para su correcto funcionamiento.

4.2. MODELO DE TRES CAPAS

La arquitectura del sistema se basa en un modelo de tres capas [34]. El objetivo de este modelo es lograr la independencia y máxima abstracción entre ellas. Estas tres capas son: capa de **persistencia**, capa de **negocio** y capa de **presentación**.

A grandes rasgos, la capa de persistencia es la encargada del acceso a los datos; la capa de negocio es la encargada de la lógica de negocio del sistema, y la capa de presentación es la encargada de presentar la aplicación a través de la interfaz al usuario.

La principal ventaja de este modelo es que cada capa está abstraída de las demás, y en caso de que deba producirse algún cambio en alguna de ellas, esto no afectaría a las demás. Además este tipo de capas son escalables ya que pueden ampliarse con facilidad en caso de que se necesitase.

En la siguiente figura vemos un esquema general de este modelo de tres capas, donde cada una puede comunicarse con la capa inferior. Vamos a ir desarrollando a qué se dedica cada capa y cómo está implementada, usando un framework distinto en cada caso.

Figura 22. Modelo de tres capas

Algunas características principales de este modelo son la centralización de la lógica de negocio en una sola capa, y además la capa de presentación pide o envía información a la capa de negocio, y no directamente al gestor de la base de datos o a la base de datos.

Además de estas 3 capas, cabe mencionar la capa de **integración**. Esta capa es muy importante ya que es la encargada de realizar la conexión entre la capa de presentación y la capa de negocio. Para esta capa en el lado del cliente usaremos la librería ksoap2, y en el lado del servidor usaremos el framework Axis2 que implementa WSDL. Más adelante explicaremos con más detalle esta capa.

4.3. MySQL – Base de Datos

El software MySQL [35] es un sistema de administración de bases de datos SQL (Structured Query Language) muy rápido, robusto y fácil de usar. Se adapta bien a la administración de datos en un entorno de red, especialmente en arquitecturas cliente/servidor. El servidor MySQL está diseñado para entornos de producción críticos, con alta carga de trabajo así como para integrarse en software para ser distribuido.

MySQL es un Sistema Gestor de Bases de Datos (SGBD, o en inglés DBMS, Database Management System) muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales como de entretenimiento, precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo los términos de la licencia GNU General Public License [36] le otorgan como beneficios adicionales contar con un alto grado de estabilidad y un rápido desarrollo. Por ello nos decantamos por este software.

Pero necesitábamos algún manager que nos permitiese hacer uso de todas las funcionalidades de MySQl de una forma fácil y amena. Elegimos el software SQL Maneger for MySQL.

❖ SQL Manager for MySQL

SQL Manager para MySQL [37] es una herramienta de alto rendimiento para la administración de bases de datos MySQL y el desarrollo. Funciona con cualquier versión de MySQL desde 3.23 hasta la más reciente y soporta todas las últimas características de MySQL incluyendo triggers, vistas, procedimientos almacenados y funciones, claves de InnoDB extranjeros, los datos Unicode y así sucesivamente. SQL Manager para MySQL permite crear y editar todos los objetos de base de datos MySQL, el diseño visual de bases de datos MySQL, ejecutar scripts de SQL, importar y exportar datos de bases de datos, administrar usuarios y sus privilegios, y tiene muchas otras funciones útiles para una eficiente administración de MySQL.

4.4. HIBERNATE - Capa de persistencia

La mayoría de las aplicaciones requieren almacenar y recuperar información la cual está almacenada en una base de datos. Esto implica de alguna forma una interacción con la base de datos, y puede presentar un problema a los desarrolladores ya que tanto la base de datos como la programación orientada a objetos se desarrollan en entornos muy diferentes. Lo que llevó a los desarrolladores a intentar construir un puente entre el mundo relacional de la base de datos y el mundo orientado a objetos.

La tarea de persistir objetos Java en una base de datos relacional actualmente está siendo facilitada por un gran número de herramientas que permiten a los desarrolladores dirigir motores de persistencia para convertir objetos Java a columnas/registros de una base de datos y viceversa. Esta tarea implica serializar objetos Java estructurados en forma de árbol a una base de ratos relacional estructurada de forma tabular y viceversa. Para ello, es esencial

mapear los objetos Java a columnas y registros de la base de datos de una manera optimizada en velocidad y eficiencia.

Hibernate [38] es una de estas tecnologías de persistencia de objetos, que intenta simplificar la tarea de conectar las bases de datos con el lenguaje Java. Hibernate funciona persistiendo y restaurando viejos objetos Java (POJOs) utilizando un modelo de programación muy transparente y poco exigente. Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL [36].

Hibernate es un marco de trabajo Java que proporciona mecanismos de mapeo objeto/relacional para definir cómo se almacenan, eliminan, actualizan y recuperan los objetos Java. El término utilizado es ORM (object/relational mapping) y consiste en la técnica de realizar la transición de una representación de los datos de un modelo relacional a un modelo orientado a objetos y viceversa. La vinculación entre objetos y la base de datos relacional se realiza mediante archivos xml. Además, Hibernate ofrece servicios de consulta y recuperación que pueden optimizar los esfuerzos de desarrollo dentro de entornos SQL y JDBC.

Hibernate fue una iniciativa de un grupo de desarrolladores dispersos alrededor del mundo conducidos por Gavin King [39]. Tiempo después, JBoss Inc. (empresa comprada por Red Hat) contrató a los principales desarrolladores de Hibernate y trabajó con ellos en brindar soporte al proyecto.

Una de las características únicas de Hibernate es que no requiere que los desarrolladores implementen interfaces o extiendan clases base para poder persistir las clases. En vez de eso, Hibernate hace uso de la API Java Reflection y el aumento de clases en tiempo de ejecución utilizando una librería de generación de código Java muy poderosa y de alto rendimiento llamada CGLIB. La librería CGLIB se utiliza para extender clases Java e implementar interfaces Java en tiempo de ejecución.

El hecho de que Hibernate use la API Java Reflection es para facilitarle la vida al programador, al permitirle que use simples clases POJO para trabajar con ellas. Otros frameworks menos potentes (o versiones antiguas de Hibernate), obligaban al programador a que sus clases implementaran ciertas interfaces o pertenecieran a complicadas jerarquías de clases, lo cual limitaba la flexibilidad del programador y complicaba la comprensión del código.

El diagrama a continuación brinda una perspectiva a alto nivel de la arquitectura de Hibernate:

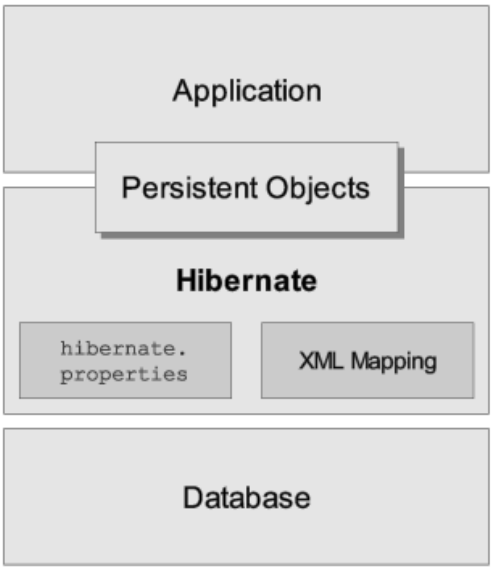


Figura 23. Arquitectura Hibernate

Este diagrama ilustra la manera en que Hibernate utiliza la base de datos y los datos de configuración para proporcionar servicios de persistencia y objetos persistentes a la aplicación.

Los pasos para trabajar con Hibernate son los siguientes, los cuales se ampliarán en el capítulo dedicado a la implementación y haciendo referencia al proyecto:

- Crear la base de datos.
- Crear un Javabean o POJO (Clase Java que implementa la interface Serializable, tiene un constructor sin parámetros y tiene métodos get y set para sus atributos).
- Crear un archivo de mapeo xml que indique la correspondencia entre el bean y una tabla de la base de datos.
- Crear un archivo de propiedades con la configuración JDBC para acceder a la base de datos.
- Incluir en el proyecto las bibliotecas de Hibernate.

El primer paso es la creación de la base de datos. Para más información sobre ello consultar cómo creamos la base de datos de nuestro proyecto y las tablas que la componen.

El segundo paso es crear un JavaBean o POJO, es decir, una clase java simple, por cada tabla de la base de datos. Un javabean o pojo es una clase con métodos get y set para acceder a sus atributos, con un constructor sin parámetros y que implementa la interface Serializable. Cada atributo del javabean representa a un campo de la tabla correspondiente de la base de datos.

El tercer paso es crear un archivo xml de mapeo para establecer la correspondencia entre los beans y las tablas de la base de datos. Habrá también un archivo de mapeo xml por cada tabla de la base de datos, y se les asignará el nombre de la clase seguido de la extensión “hbm.xml”.

El cuarto paso es realizar la configuración para el acceso a la base de datos. Se debe indicar los parámetros de acceso a la base de datos, como el driver utilizado, la url donde está alojada la

base de datos y unos datos de acceso a ella, que son un usuario y una contraseña. También se deben indicar los archivos de mapeo a emplear.

El quinto paso es incluir las librerías necesarias para el funcionamiento de Hibernate en el proyecto. En el capítulo de implementación se comentan las librerías que nosotros hemos tenido que añadir para ello.

Además de estas características, Hibernate posee su propio lenguaje de consultas llamado HQL donde las consultas se realizan sobre objetos java. De este lenguaje se hablará en el siguiente apartado dedicado a Spring. Además Hibernate cuenta con la API Hibernate Criteria. Es una API que nos facilita las consultas (no las inserciones) a la base de datos. La ventaja fundamental de esta API es que nos permite tratar la composición de la consulta de una forma totalmente orientada a objetos. Es decir, no compone la consulta a través de una cadena insertada por nosotros mismos como pasa en HQL, sino que la compone la propia API a través de unas propiedades que nosotros establecemos. Lo único que tenemos que hacer es declarar el objeto Criteria como *DetachedCriteria* y una vez compuesto utilizar el método *findByCriteria* que nos suministra *HibernateTemplate*. Hibernate también incluye lo que se llama Hibernate Annotations, que nos permite realizar los mapeos mediante anotaciones en vez de hacerlo mediante archivos xml.

En este apartado de Hibernate hemos querido comentar qué es, cómo funciona, y qué ventajas puede ofrecer. Además en el siguiente capítulo se ampliará parte de esta información adaptándose al uso que hemos hecho de Hibernate en el proyecto. También hablaremos de él en el siguiente apartado y de su colaboración con Spring. Hibernate es un software que nos proporciona muchas ventajas y funcionalidades pero no es posible mostrarlas todas en este documento. Para más información el usuario puede acudir por ejemplo al manual de su página oficial http://www.hibernate.org/, o a una traducción del mismo alojado en la web http://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/.

4.5. SPRING - Capa de negocio

Spring [40] es un marco de trabajo de código abierto creado por Rod Johnson [41]. Spring se creó para hacer frente a la complejidad del desarrollo de aplicaciones empresariales, haciendo posible el uso de JavaBeans sencillos para conseguir objetivos que antes sólo eran posibles con la EJB (Enterprise JavaBeans).

Un Java Bean (o también conocido simplemente como Bean) es un componente utilizado en Java que permite agrupar funcionalidades para formar parte de una aplicación. Es una clase Java que cumple con ciertas normas con los nombres de sus atributos y operaciones. Un JavaBean tiene declarados sus atributos como privados e implementa para cada uno de ellos un método setter y getter, añadiéndole la palabra "set" o "get" al nombre del atributo y un constructor sin parámetros. Además esta clase java debe implementar la interfaz Serializable.

Un Enterprise Java Bean (EJB) [42] es un componente de negocio J2EE que también agrupa funcionalidades para una aplicación, sin embargo para su ejecución necesita un contenedor EJB, lo cual permite el acceso a los servicios del mismo. Luego un Java Bean requiere ser integrado con otros componentes para que éste sea funcional, mientras un Enterprise Java Bean a través de un EJB Container puede ser activado.

Diciéndolo más coloquialmente, Spring es el "pegamento" que une todos los componentes de la aplicación, maneja su ciclo de vida y la interacción entre ellos.

La utilidad de Spring no se limita al desarrollo en el lado del servidor. Cualquier aplicación de Java puede beneficiarse de Spring, gracias a su simplicidad y capacidad de prueba. En nuestro caso, el uso de este framework sólo se ha desarrollado para el lado del servidor. Además al usar Spring nuestra aplicación quedará dividida en capas bien delimitadas.

❖ Inversión de Control

El núcleo de Spring está basado en un patrón de diseño llamado Inversión de Control (IoC, Inversion of Control) [43]. Las aplicaciones que lo usan basan sus configuraciones en archivos xml para describir las dependencias entre sus componentes, es decir, es Spring el que controla la estructura de la aplicación. De esta forma, si queremos crear una clase que dentro de ella se cree una instancia de otra clase, o incluso de una interface, esa instancia nos la proporcionará el motor de Spring. Spring es el contenedor que inyecta estas dependencias cuando crea el bean. Esta inyección de dependencias en tiempo de ejecución ha hecho que a este tipo de IoC se le dé ese mismo nombre de inyección de dependencias (DI, Dependency Injection), es decir, los componentes no saben la implementación concreta de los otros componentes, sólo ven sus interfaces. Usando DI reducimos casi a cero la cantidad de código a escribir para un diseño basado en interfaces.

En el contexto DI, Spring actúa como un contenedor que además de crear los componentes de la aplicación, contiene y maneja el ciclo de vida de estos componentes. Por ello, podemos declarar cómo debe ser creado cada componente, cómo deben ser configurados y como deben asociarse con los demás. La implementación de DI de Spring se enfoca en el acoplamiento débil, es decir, los componentes deben saber lo menos posible de los demás componentes, y la forma más fácil de conseguirlo es mediante el uso de interfaces.

Además de estas ventajas, el uso de DI reduce el código utilizado para unir los diferentes componentes. El hecho de configurar las dependencias en archivos xml externos permiten realizar una reconfiguración fácilmente, y manejar todas las dependencias desde un solo lugar.

La IoC se basa principalmente en dos paquetes, llamados *org.springframework.beans* y *org.springframework.context*. Estos dos paquetes contienen dos interfaces para la instanciación de objetos, y ambas representan contendores de beans. Una de ellas es *BeanFactory* que proporciona un mecanismo de configuración avanzada capaz de manejar cualquier tipo de objeto, pero existe una sub-interface de ésta llamada ApplicationContext, la cual agrega una integración más fácil, además de un módulo de Programación Orientada a Aspectos (AOP, Aspect-Oriented Programming), para el manejo de recursos de mensajes, publicación de eventos, y contextos específicos para ciertas capas de aplicaciones (como aplicaciones web), entre otras. Spring proporciona muchas implementaciones de *ApplicationContext*. En particular en las aplicaciones que no son web usamos *ClassPathXmlApplicationContext*, que busca los archivos de configuración dentro del classpath, o *FileSystemXmlApplicationContext*, que los busca en el sistema de archivos de la máquina donde se ejecute la aplicación.

Estos objetos Bean son los objetos instanciados por Spring, y las dependencias entre ellos se deben declarar en los archivos xml de mapeo y configuración de Spring. Aunque esta configuración también puede realizarse mediante anotaciones en java, nosotros nos centraremos en la primera ya que es la que usamos en nuestro proyecto.

Dentro del archivo de configuración xml se incluyen las definiciones de beans, es decir, las definiciones de los objetos que componen nuestra aplicación, y se realiza mediante un elemento *<beans>* que dentro contiene todos los elementos *<bean>* que necesitemos. Cuando el proyecto crece y necesitamos muchos bean para el funcionamiento, y en el caso particular de nuestro proyecto, es mejor usar varios xml para agrupar los bean. En nuestro caso tendremos un archivo xml de este tipo para cada tabla de la base de datos, para incluir los bean relacionados con cada una de forma independiente. Luego esto deberá incluirse también en un archivo de configuración importando todos los archivos xml creados que incluyan los bean mediante el elemento *<import>*.

Volviendo a la configuración de un bean, dentro de cada elemento *<bean>* se debe dar un identificador dentro de la aplicación mediante el atributo *id*; de esta forma el bean puede ser referenciado con ese nombre. A continuación se debe indicar a Spring de qué tipo es el bean, es decir, la clase de la que queremos que cree los objetos, mediante el atributo *class*.

Debido a que *ApplicationContext* proporciona más funcionalidades, es preferible su uso frente a *BeanFactory*. Hay muchas implementaciones de ApplicationContext, aunque hay tres que son las más usadas:

- *ClassPathXmlApplicationContext*: carga la configuración desde un archivo XML ubicado en el classpath.
- *FileSystemXmlApplicationContext*: carga la configuración desde un archivo XML ubicado en cualquier parte del sistema de archivos de la computadora.
- *XmlWebApplicationContext*: carga la configuración desde un archivo XML ubicado en el contexto de una aplicación web.

Por ello, nosotros usaremos la implementación *ClassPathXmlApplicationContext*, y a partir de ella obtendremos el bean mediante el uso del método *getBean()*. De esta forma, el bean

recibido y que Spring ha creado nos permite hacer uso de todos los métodos que hayamos definido. Este uso de los bean puede verse dentro del proyecto en el capítulo de implementación.

Por defecto, los beans en Spring son *Singleton* [44]. *Singleton* es un patrón de diseño que asegura que solo hay una instancia de un bean en la aplicación. Es decir, no importa cuántas veces llamemos al método *getBean* de *ApplicationContext* para el mismo bean, que Spring siempre nos devolverá la misma instancia.

En el archivo de configuración de los bean en Spring se permite modificar el scope asociado a cada bean. Por defecto es *singleton,* como acabamos de ver, pero existen otros cuatro valores:

- *prototype*: se creará una nueva instancia del bean por cada llamada a *getBean*, es decir, cada vez que el bean vaya a ser usado.
- *request:* existirá una sola instancia del bean por cada petición HTTP, es decir, cada petición HTTP tiene su propia instancia de un bean. Este scope sólo es válido cuándo se usa un contenedor de Spring con capacidades web como Spring MVC.
- *session*: existirá una instancia del bean por cada sesión HTTP. Este scope sólo es válido cuando se usa un contenedor de Spring con capacidades web como Spring MVC.
- *globalSession*: existirá una instancia del bean por cada sesión global HTTP. Este scope sólo es válido cuando se usa un contenedor de Spring con capacidades web como Spring MVC.

Por lo tanto en nuestro proyecto no hemos modificado este atributo, y lo hemos dejado por defecto a *singleton*, de forma que esa instancia única es almacenada en una caché especial para estos beans *singleton*, y todas las llamadas subsecuentes para obtener ese bean recibirán el objeto que se encuentra en la caché.

❖ Patrón DAO

Por otra parte, Spring emplea otro patrón llamado Spring DAO que permite la implementación del Patrón DAO a partir de clases específicas que facilitan la inversión del control IoC.

Este patrón de Objeto de Acceso a Datos (DAO, Data Access Object) [45] se usa para para abstraer y encapsular el acceso a los datos. Un DAO maneja la conexión con la fuente de datos para obtener y guardar los datos.

Existen clases para el acceso a la información dependiendo del software ORM usado. En nuestro caso, para Hibernate existe la clase *HibernateDaoSupport*, que requiere la asignación de un atributo *sessionFactory* y mediante el método *getHibernateTemplate* proporciona un objeto sobre el que realizar las consultas a la base de datos. Luego las clases que queramos que usen este patrón DAO deberán extender a esta clase *HibernateDaoSupport*.

La clase *HibernateTemplate*, permite realizar consultas con lenguaje HQL y diferentes operaciones. Algunas de estas operaciones, que además hemos usado en nuestro proyecto son:

- List find(String hql, Object[] args): devuelve un listado de objetos con el resultado de ejecutar una consulta HQL con los parámetros suministrados en args.
- void delete(Object entidad): borra de la base de datos el registro asociado al objeto suministrado.
- void update(Object entidad): actualiza en la base de datos el registro asociado al objeto suministrado.
- void save(Object entidad): inserta en la base de datos un registro con los valores del objeto suministrado.

❖ Hibernate Query Languaje

HQL (Hibernate Query Languaje) es un lenguaje de consulta potente que se parece a SQL. Sin embargo, comparado con SQL, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. Su principal particularidad es que las consultas se realizan sobre los objetos java que forman nuestro modelo de negocio, es decir, las entidades que se persisten en Hibernate. Esto hace que algunas características de HQL sean:

- Los tipos de datos son los de Java.
- Las consultas son independientes del lenguaje de SQL específico de la base de datos
- Las consultas son independientes del modelo de tablas de la base de datos.
- Es posible tratar con las colecciones de Java.
- Es posible navegar entre los distintos objetos en la propia consulta.

El lenguaje HQL es sensible a mayúsculas y minúsculas para el nombre de las clases y los atributos de ellas, pero no lo es para las palabras reservadas y para los valores a comparar en los atributos.

Dentro del manual oficial de Hibernate en la web antes nombrada podemos encontrar un capítulo completo sobre este lenguaje HQL [46].

4.6. ANDROID - Capa de presentación

Android [47] es un paquete de software basado en código abierto para teléfonos móviles, creado por Google y la Open handset Alliance [48]. Se encuentra dentro de millones de teléfonos móviles, algo que convierte a Android en una de las plataformas principales para los desarrolladores de aplicaciones.

Android es el primer entorno que combina las siguientes características:

- Una plataforma de desarrollo completamente abierta y gratuita basada en Linux y en código abierto: Esta plataforma les gusta a los desarrolladores porque pueden utilizarla y personalizarla porque saben que no está limitada a ningún vendedor.

- Una arquitectura tipo componentes: Las partes de una aplicación pueden utilizarse en otra distinta de un modo inimaginable por el desarrollador. Incluso es posible sustituir componentes integrados con sus propias versiones mejoradas. Esto liberará una nueva corriente de creatividad en el espacio móvil.

- Toneladas de servicios integrados: Los servicios de localización utilizan GPS o triangulación de celdas para permitirle personalizar la experiencia al usuario en función de su ubicación. Una completa base de datos SQL le permite aprovecharse de la firmeza del almacenamiento local durante tareas de sincronización. Las vistas de navegación y mapas pueden integrarse directamente en sus aplicaciones. Todas estas capacidades de integración le ayudarán a incrementar las funcionalidades a la vez que reduce los costes de desarrollo. La mayoría de estos servicios nombrados están integrados dentro de nuestro proyecto.

- Gestión automática del ciclo de vida de la aplicación: Los programas están aislados los unos de los otros, mediante la inclusión de varias capas de seguridad, las cuales proporcionarán un nivel de estabilidad del sistema no visto antes en ningún teléfono inteligente. El usuario final no tendrá que preocuparse de qué aplicaciones están activas o de cerrar algunos programas para abrir otros. Android está optimizado para dispositivos de poca memoria y poca batería, de una forma nunca abarcada por plataformas anteriores.

- Gráficos y sonido de alta calidad: Gráficos vectoriales suavizados en dos dimensiones y animaciones Flash, se unen a los gráficos 3D acelerados OpenGL para permitir nuevos tipos de juegos y aplicaciones empresariales. Los códec para los formatos de audio y vídeo más comunes están integrados, como es el caso de H.264 (AVC) y AAC

- Portabilidad a través de un amplio rango de hardware actual y futuro: Todos sus programas están escritos en Java y ejecutados por la máquina virtual Dalvik de Android, de modo que su código se podrá portar a arquitecturas como ARM, x86 y otras. Se incluye compatibilidad con otros métodos de entrada como teclados, ratones e interfaces táctiles. Las interfaces de usuario permiten muchas funcionalidades para que puedan ser adaptadas a diferentes resoluciones de pantalla y orientación.

Android ofrece un soplo de aire fresco en el modo en que las aplicaciones móviles interactúan con los usuarios, junto con los elementos técnicos para hacerlo posible.

❖ Arquitectura Android

Comenzamos analizando la arquitectura del sistema global: las capas y componentes principales que conforman la pila de software de código abierto Android. En la siguiente imagen se puede observar un esquema de la estructura global de Android. Cada capa utiliza los servicios proporcionados por las capas inferiores a ella. Se explicará a continuación esta estructura comenzando por abajo.

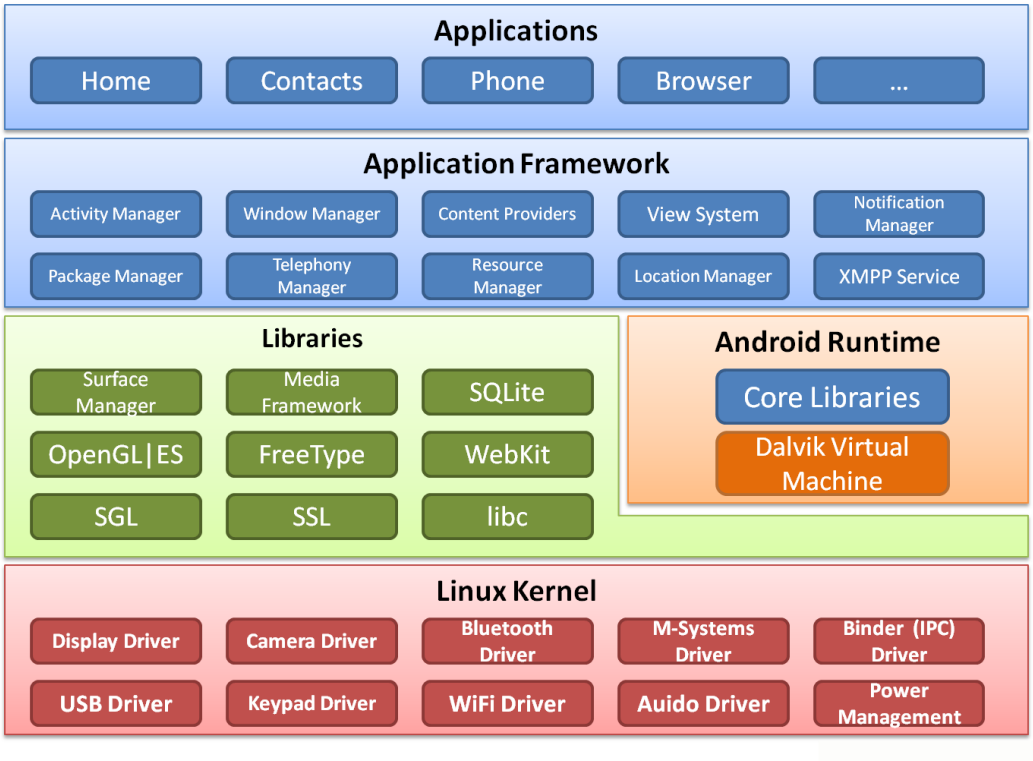


Figura 24. Arquitectura del sistema Android

- **Kernel de Linux**

Android está construido sobre el kernel de Linux. Creado por Linus Torvalds en 1991, Linux puede encontrarse hoy en día en todo tipo de ordenadores, desde los más reducidos hasta superordenadores con grandes capacidades de procesamiento. Linux proporciona a Android la capa de abstracción de hardware, permitiendo a Android que pueda portarse a una amplia variedad de plataformas.

Internamente, Android utiliza Linux para la gestión de su memoria, de los procesos, para las operaciones de red y para otros servicios del sistema operativo. El usuario del teléfono Android nunca verá Linux y sus programas no realizarán llamadas a Linux directamente.

- **Bibliotecas nativas**

La siguiente capa situada encima del kernel contiene las bibliotecas nativas de Android. Estas bibliotecas compartidas están escritas en C o C++, compiladas para la arquitectura de hardware concreta utilizada por el teléfono y preinstalada por el vendedor del teléfono. Algunas de las bibliotecas nativas más importantes son:

- Gestor de superficie:

Android utiliza un gestor de ventana de composición similar a Vista o Compiz pero mucho más sencillo. En lugar de dibujar directamente en el buffer de pantalla, los comandos de dibujo se colocan en mapas de bit fuera de la pantalla en combinación con otros mapas de bit para formar la pantalla que ve el usuario. Esto permite al sistema crear todo tipo de interesantes efectos como ventanas transparentes y divertidas transiciones.

- Gráficos 3D y 2D:

En Android es posible combinar elementos de dos y tres dimensiones en una única interfaz de usuario. La biblioteca utilizará hardware 3D si el dispositivo dispone de él o un rederizador de software en caso de no tenerlo.

- Códecs multimedia:

Android puede reproducir vídeo y grabar/reproducir audio en diferentes formatos como AAC, AVC (H.264), H.263, MP3 y MPEG-4.

- Base de datos SQL:

Android incluye el motor de base de datos SQLite, la misma base de datos utilizada en Firefox y el iPhone de Apple. Puede ser utilizada en la aplicación para el almacenamiento persistente.

- Tecnología de navegador:

Para la muestra rápida de contenido HTML, Android utiliza la biblioteca WebKit. Esta es la misma tecnología empleada en los navegadores Chrome de Google, Safari de Apple, el iPhone de Apple y la plataforma S60 de Nokia.

Estas bibliotecas no son aplicaciones que se mantienen por sí solas. Sólo existen si son llamadas por programas de un nivel más alto. Desde Android 1.5, es posible escribir y desarrollar bibliotecas nativas utilizando el Native Development Kit (NDT).

- **Tiempo de ejecución Android**

También sobre el kernel se encuentra el tiempo de ejecución Android, que incluye la máquina virtual Dalvik y las bibliotecas core de Java.

Dalvik es una máquina virtual (VM) diseñada y escrita por Dan Bornstein en Google. El código se compila en instrucciones independientes de la máquina llamados bytecodes, las cuales son ejecutadas por la VM Dalvik en el dispositivo móvil. Aunque los formatos de bytecode son

ligeramente diferentes, Dalvik es simplemente una máquina virtual Java optimizada para requerimientos de poca memoria. Permite que se ejecuten varias instancias de VM al mismo tiempo y se aprovecha del sistema operativo interno (Linux) para aspectos de seguridad y aislamiento de procesos.

La VM de Dalvik es la implementación de Java por parte de Google, optimizada para dispositivos móviles. Todo el código escrito para Android estará escrito en Java y se ejecutará en la VM. Dalvik se diferencia de la máquina tradicional Java en dos aspectos importantes:

- La VM Dalvik ejecuta archivos .dex, los cuales son convertidos en el momento de la compilación a partir de archivos estándar .class y .jar. Los archivos .dex son más compactos y eficientes que los archivos .class, una consideración importante teniendo en cuenta que el objetivo de Android son dispositivos de memoria limitada y alimentados con batería.
- Las bibliotecas core de java que vienen con Android son diferentes de las que ofrecen Java Standard Edition (Java SE) y Java Mobile Edition (Java ME). Sin embargo, son bastantes las coincidencias que se dan.

- **Estructura de las aplicaciones**

Encima de las bibliotecas nativas y el tiempo de ejecución se encuentra la capa de estructura de aplicaciones. Esta capa proporciona los bloques de construcción de alto nivel que serán utilizados al crear aplicaciones. La estructura viene preinstalada con Android pero puede ser ampliada con componentes propios según lo requiera.

Las partes más importantes de la estructura son las siguientes:

· Gestor de actividad: Este gestor controla el ciclo de vida de las aplicaciones y mantiene un orden de aplicaciones para que el usuario pueda navegar por ellas.

· Proveedores de contenido: Estos objetos encapsulan datos que necesitan ser compartidos entre aplicaciones, como contactos.

· Gestor de recursos: Los recursos son cualquier elemento incluido en el programa que no sea código.

· Gestor de ubicación: Un teléfono Android siempre sabe dónde está.

· Gestor de notificaciones: Eventos como los mensajes entrantes, citas, alertas de proximidad, etc… pueden ser mostrados al usuario de forma atractiva.

- **Aplicaciones**

La capa más alta en el diagrama de la arquitectura Android es la de aplicaciones y widgets. Es como la punta del iceberg de Android. Los usuarios finales, verán únicamente estos programas, sin saber qué está sucediendo debajo de ese nivel. Las aplicaciones son programas que pueden ocupar toda la pantalla e interactuar con el usuario.

❖ Aplicaciones y actividades

En el escritorio de su Linux o Windows se pueden tener varias aplicaciones en ejecución y visibles en diferentes ventanas al mismo tiempo. En Android existe una aplicación en primer plano que normalmente ocupa toda la pantalla excepto la barra de estado. Cuando el usuario enciende el teléfono la primera aplicación que ve es la de Inicio.

Cuando el usuario ejecuta una aplicación, Android la inicia y la pone en primer plano. Desde esa aplicación el usuario puede ejecutar otras aplicaciones o abrir otras pantallas de la misma aplicación y así sucesivamente. Todos estos programas y pantallas son grabados en la pila de la aplicación por el Gestor de actividad del sistema. En cualquier momento, el usuario puede utilizar el botón Volver para retornar a la pantalla anterior de la pila. Desde el punto de vista del usuario, el funcionamiento es muy parecido al del historial en un navegador Web. Al pulsar Volver, volvemos a la página anterior.

Internamente, cada pantalla de la interfaz de usuario está representada por una clase Activity. Cada actividad tiene su propio ciclo de vida. Una aplicación es una o varias actividades más un proceso Linux que las contiene.

En Android una aplicación puede estar “viva” incluso si un proceso ha sido “matado”. Dicho de otro modo, el ciclo de vida de una actividad no está ligado al ciclo de vida de un proceso. Los procesos no son más que contenedores desechables de actividades. Este comportamiento probablemente sea diferente al de cualquier otro sistema.

Una actividad de un programa Android puede estar en alguno de los estados de la siguiente imagen durante su ciclo de vida. Como el desarrollador no tiene control sobre el estado en que se encuentra su programa es un procedimiento controlado por el sistema. Sin embargo recibe notificaciones cuando el estado va a cambiar gracias a las llamadas al método *onXX()*.

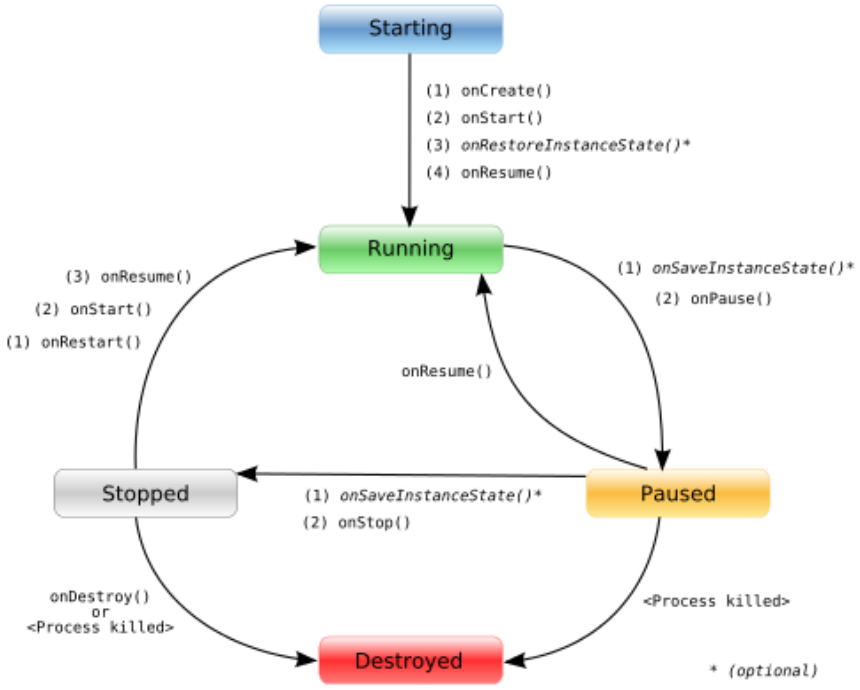


Figura 25. Ciclo de vida de una actividad Android

Algunos de estos métodos son:

- → onCreate(Bundle): A este método se le llama cuando se inicia la actividad por primera vez. Puede ser utilizado para llevar a cabo inicializaciones individuales como la que se realiza al crear la interfaz de usuario. onCreate() toma un parámetro que puede ser null o alguna información de estado guardada previamente por el método onSaveInstanceState().
- → onStart(): Éste indica que la actividad va a ser mostrada al usuario.
- → onResume(): A éste se le llama cuando la actividad puede comenzar a interactuar con el usuario.
- → onPause(): Se ejecuta cuando la actividad se dispone a pasar a segundo plano, normalmente porque otra actividad ha sido ejecutada en primer plano. Aquí es donde debe guardar el estado persistente de su programa, como en el caso del registro de una base de datos que está siendo editado.
- → onStop(): Se llama cuando la actividad ya no es visible por el usuario y no será necesaria durante un tiempo. Si la memoria es limitada, puede que no se llame nunca a onStop() (el sistema termina directamente el proceso).
- → onRestart(): Si se llama a este método, indica que se está volviendo a mostrar la actividad al usuario desde un estado detenido.
- → onDestroy(): Se llama a este método justo antes de que la actividad sea destruída. Si la memoria es limitada, puede que no se llame nunca a onDestroy()(el sistema termina directamente el proceso).
- → onSaveInstanceState(Bundle): Android llamará a este método para permitir a la actividad guardar el estado de la instancia, como puede ser la posición del cursor en un campo de texto. Normalmente no será necesario omitirla ya que la implementación

predeterminada guarda el estado de todos los controles de la interfaz de usuario de forma automática.

Las actividades que no se estén ejecutando en primer plano pueden ser detenidas o los procesos Linux que las albergan pueden ser finalizados en cualquier momento, con el objetivo de dejar espacio para nuevas actividades. Este planteamiento es muy habitual, de modo que es importante que una aplicación esté diseñada desde el comienzo siguiendo esta idea. Por ello desde un primer momento en la aplicación hemos intentado gestionar los ciclos de vida de cada actividad de forma que no se produzcan errores inesperados. E incluso implementando la funcionalidad del botón atrás para que vuelva a la actividad correspondiente en cada caso.

4.7. AXIS2 - Capa de integración

Axis2 [49] es la versión mejorada del contenedor de Web Services AXIS. El proyecto ha evolucionado independientemente de la primera versión debido a que implementa especificaciones diferentes.

AXIS originalmente se convirtió en uno de los contenedores más extendidos para implantar soluciones basadas en Web Services [50] o que proveían acceso a sistemas preexistentes mediante Web Services SOAP [51].

Con la aparición de AXIS2 y sobre todo con el aumento de la documentación y mejora de estabilidad de las versiones iniciales AXIS2 se ha convertido en la solución de referencia para construir Web Services y desplegarlos bajo los requisitos más exigentes.

Las mejoras han sido numerosas, pero cabe destacar los conceptos de Ingeniería que se han aplicado a su diseño interno. En primer lugar, todas las funcionalidades de AXIS2 se encuentran divididas en módulos de manera que se distinguen módulos básicos y módulos opcionales.

Cada modulo tiene asociados una serie de flujos de entrada y de salida sobre los que podemos configurar manejadores (handlers). Estos manejadores habitualmente serán clases que el contenedor notificará cuando se produzca un evento (la llegada de un mensaje SOAP por ejemplo).

Uno de los aspectos fundamentales de AXIS2 es AXIOM (Axis Object Model). AXIOM [52] es el modelo de objetos que AXIS2 emplea para serializar y deserializar mensajes SOAP aprovechando para ello API's de la familia JAX (Java Api for XML). AXIOM aporta su propia capa de abstracción encima de las API's de JAX pero en general el funcionamiento es similar al empleado en DOM de JAXP.

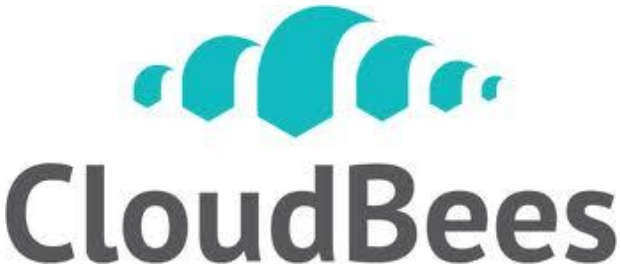
Con AXIOM obviamente estamos expuestos a la implementación aportada por el parser de XML, pero AXIS2 incluye su propia implementación de referencia, con la que debería ser suficiente para la mayoría de aplicaciones estándar (en caso contrario siempre se puede contar con algún experto que solucione las limitaciones de una implementación dada).

Gracias a esta tecnología podremos presentar unos servicios los cuales el cliente podrá consumir, de forma que la integración entre el servidor y el cliente pueda combinarse de una forma fácil, segura e independiente de la tecnología del cliente.

Axis 2 nos ayuda a exponer el código del servidor como un web service. Además Axis2 soporta WSDL (Web Services Description Language) [53] lo que facilita la construcción del acceso al servicio web. WSDL describe los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios. De esta forma, un cliente se conectará a un servicio web a través del WSDL determinando qué funciones puede consumir y cómo debe hacerlo.

❖ CloudBees

CloudBees [54] es una **plataforma como servicio PaaS** (Platform as a Service) [55] para aplicaciones Java/JEE que cubre tanto el despliegue en Cloud como servicios para el desarrollo en el nube. La computación en la nube, del inglés cloud computing, es un paradigma que permite ofrecer servicios de computación a través de Internet. PaaS es un modelo en el que se ofrece todo lo necesario para soportar el ciclo de vida completo de construcción y puesta en marcha de aplicaciones y servicios web completamente disponibles en Internet. Otra característica importante es que no hay descarga de software que instalar en los equipos de los desarrolladores. PasS ofrece mútliples servicios pero todos como una solución integral en la web.



Es decir, CloudBees es un servicio que te permite desplegar aplicaciones Java en un servidor en producción. Esto se conoce como el RUN@Cloud. En la cuenta gratuita permiten hasta 5 aplicaciones diferentes. Además de las aplicaciones, CloudBees también ofrece la posibilidad de utilizar sus servidores de base de datos MySQL, también hasta 5 como máximo y con un tamaño máximo de 5MB cada una.

Por otro lado, la integración continua (continuous integration) es un modelo informático propuesto inicialmente por Martin Fowler [56] que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible, para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de tests de todo un proyecto. Las ventajas que ofrece el uso de este modelo son:

- Los desarrolladores pueden detectar y solucionar problemas de integración de forma continua, evitando el caos de última hora cuando se acercan las fechas de entrega.

- Disponibilidad constante de una build para pruebas, demos o lanzamientos anticipados.
- Ejecución inmediata de las pruebas unitarias.
- Monitorización continua de las métricas de calidad del proyecto.

Jenkins [57] es un software de integración continua open source escrito en Java. Está basado en el proyecto Hudson y es, dependiendo de la visión, un fork del proyecto o simplemente un cambio de nombre. Jenkins proporciona integración continua para el desarrollo de software. Es un sistema corriendo en un servidor que es un contenedor de servlets, como Apache Tomcat. Soporta herramientas de control de versiones y puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows. El desarrollador principal es Kohsuke Kawaguchi [58]. Liberado bajo licencia MIT [59], Jenkins es software libre.

CloudBees también incluye la integración continua. Esta integración se basa principalmente en una instancia de Jenkins a la que se puede agregar tantas tareas como queramos. Es lo que se conoce en CloudBees como el DEV@Cloud. Nosotros podemos crear una tarea de tal forma que, cada vez que se detecte un cambio en nuestro repositorio, se genere un nuevo archivo WAR y automáticamente se despliegue en nuestro servidor en producción.

Nosotros en el proyecto sólo usamos los servicios de RUN@Cloud y de almacenamiento de base de datos MySQL que nos proporciona CloudBees. Trabajar con esta plataforma ha sido realmente fácil y es una herramienta que facilita de manera casi total el despliegue de servicios en la nube. Además la disposición de estos servicios es gratuita. La única pega que encontramos es que duerme por las noches y hay que activar el servicio cada día accediendo a cualquier url wsdl del proyecto.

4.8. MAVEN – Gestión del proyecto

Maven [60] es una herramienta para la gestión de proyectos de software que se basa en el concepto de POM (Proyect Object Model) [61] [62]. Es decir, con Maven vamos a poder compilar, empaquetar, generar documentación, ejecutar test, entre otras funcionalidades.

Existe otra herramienta llamada Ant [63] cuyo objetivo es similiar a Maven, pero no debemos confundirlos ya que Maven y Ant son totalmente diferentes. Algunas de las principales ventajas de Maven frente a Ant son:

Maven se basa en patrones y en estándares. Esto permite a los desarrolladores moverse entre proyectos y no necesitan aprender cómo compilar o empaquetar. Esto mejora el mantenimiento y la reusabilidad.

Mientras que con Ant escribimos una serie de tareas, y unas dependencias entre estas tareas, con el POM hacemos una descripción del proyecto. Es decir, decimos de qué se compone nuestro proyecto (nombre, versión, librerías de las que depende, ...), y Maven se encargará de hacer todas las tareas por nosotros.

Maven hace la gestión de librerías, incluso teniendo en cuenta las dependencias transitivas. Es decir, si A depende de B y B depende de C, A depende de C. Esto quiere decir que cuando empaquetemos A, Maven se encargará de añadir tanto B como C en el paquete.

4.9. ARQUITECTURA SOA

La arquitectura cliente/servidor se caracteriza por descomponer el trabajo en dos partes: el servidor, que centraliza el servicio, y el cliente, que controla la interacción del usuario. El servidor ha de ofrecer sus servicios a través de una dirección conocida. Las pautas de comportamiento que siguen este tipo de arquitecturas pueden verse en la siguiente imagen.

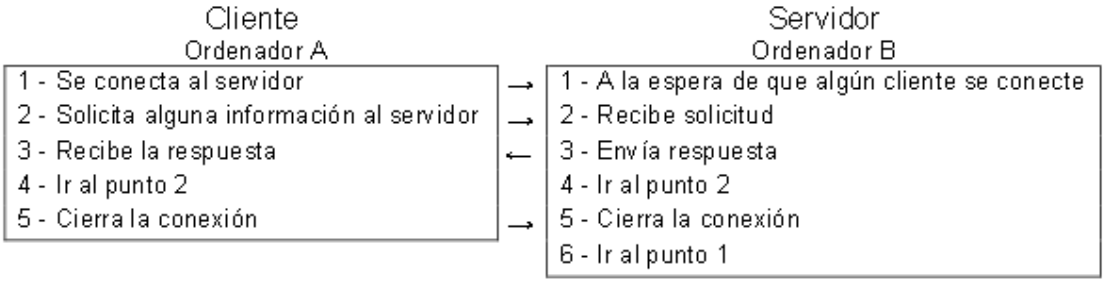


Figura 26. Comportamiento Arquitectura Cliente/Servidor

Un servicio web es un servicio disponible como API remota y accesible por internet. Los mensajes enviados están basados en el formato estándar XML. Los servicios web más usados, y que nosotros hemos usado en el proyecto, son los servicios SOAP+HTTP, con servicios definidos en el estándar WSDL.

Los servicios son una unidad básica de funcionalidad en la Arquitectura SOA (Service Oriented Architecture) [64]. Un servicio es un conjunto coherente de funcionalidad, autocontenido, sin estado e independiente. Los servicios no son dependientes de la condición de ningún otro servicio.

El estándar WSDL (Web Services Definition Language) es un lenguaje XML para describir los servicios en la arquitectura SOA. A grandes rasgos, permite caracterizar cuatro aspectos fundamentales de todo servicio:

- El conjunto de operaciones públicas que se ofrece
- Información sobre los tipos de datos empleados
- Información sobre los protocolos de comunicación para acceso

- Localización física del servicio

Y además, añade una total independencia con el cliente que puede conectarse a él, de forma que no necesita conocer la plataforma donde está desplegado ni de la implementación empleada.

El estándar SOAP (Simple Object Access Protocol) es un protocolo de mensajería para el intercambio de información entre sistemas. Aunque está concebido para utilizarlo sobre el protocolo de comunicaciones HTTP, admite otros muchos protocolos como FTP, JMS… A diferencia de otros estándares de comunicación por mensajes (como CORBA o RMI) se basa en documentos XML, por lo que resulta independiente de la tecnología y la plataforma subyacente.

De esta forma, SOA es un paradigma de arquitectura para sistemas de información que busca el mínimo acoplamiento entre sus componentes y que promueve su reutilización.

La forma en que el cliente consume un servicio web puede observarse en la figura 27.

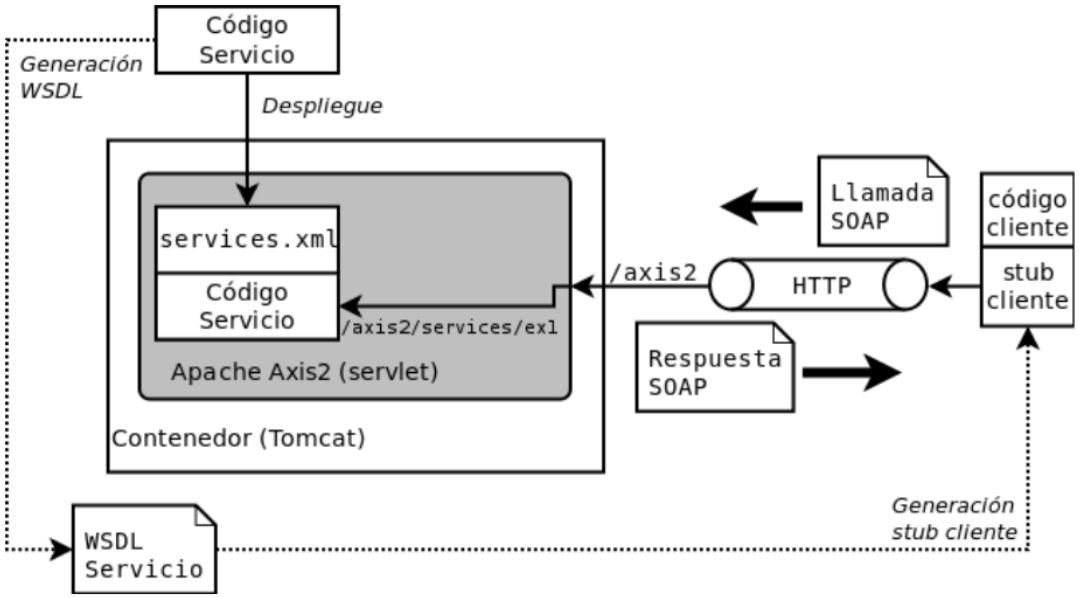


Figura 27. Conexión cliente/servidor

En el servidor se implementa el código del servicio necesario que se ejecutará al realizar la llamada. Además se crea un archivo XML llamado services.xml donde se especifica la creación del servicio. De esta forma la tecnología Axis2 es capaz de crear el WSDL que proporcionará el servicio.

Por otro lado el cliente debe hacer la llamada. Leyendo el WSDL que le expone Axis2, conoce qué funciones puede consumir y cómo debe hacerlo. De esta forma el cliente crea un objeto SOAP, y necesita definir la petición, crear una envoltura y definir un canal de transporte que será el protocolo HTTP. Ahora ya puede realizar la llamada al servicio que ha elegido, y obtener la respuesta del mismo.

Esta forma de conexión es la usada en nuestro proyecto para que el cliente Android pueda consumir los servicios que el servidor le ofrece. Aunque nuestro cliente es de tipo Android, el servidor está creado para que cualquier cliente, independientemente de su tecnología, pueda conectarse a él y consumir sus servicios, por ejemplo una aplicación web.

Para terminar este capítulo, y a modo de conclusión, podemos ver que la arquitectura del proyecto sigue el esquema de la figura 28.

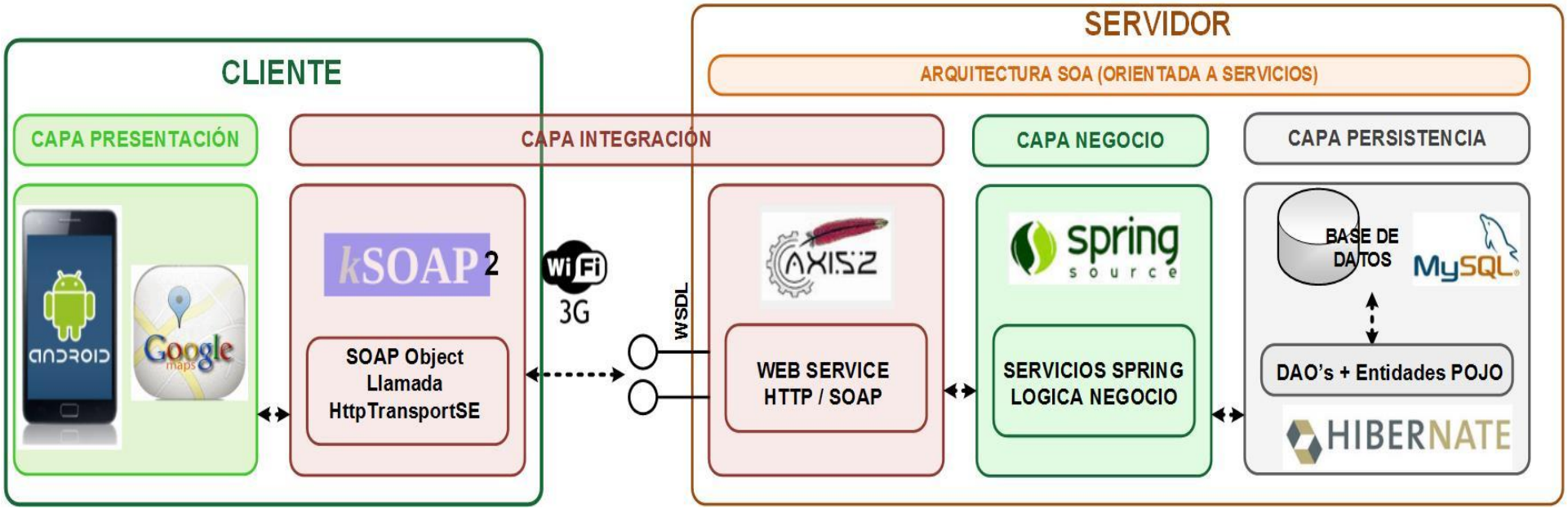


Figura 28.
Esquema de la Arquitectura del proyecto Mapa de Recursos. Arquitectura cliente/servidor que emplea el modelo de 3 capas.
- Servidor: se trata de una Arquitectura Orientada a Servicios (SOA). Incluye la base de datos y el servidor web. Implementa las capas de persistencia y negocio. El servidor expone unos servicios web, que cualquier cliente podrá consumir, con independencia de la tecnología que emplee o de la plataforma donde esté desplegado.
- Cliente: se trata de una aplicación móvil Android, la cual se identifica con la capa de presentación.
Cabe destacar la capa de integración entre el cliente y servidor, de forma que el cliente Android, podrá conectarse al WSDL del servidor y consumir los servicios que éste le ofrece.

Capítulo 5. DESARROLLO DE LA APLICACIÓN

5.1. Base de datos

La base de datos es un pilar muy importante en el proyecto. El objetivo principal de las bases de datos es el de unificar los datos y almacenarlos, de forma que los clientes o programas pueden requerir esos datos o querer introducir nuevos datos.

El modelo relacional es la forma de diseño de bases de datos más usada y extendida. La representación gráfica de este modelo es la tabla. Una tabla se compone de filas y columnas. Las filas se corresponden con los registros y las columnas se corresponden con los campos. Un campo será la unidad mínima de información.

El núcleo de cualquier base de datos son sus tablas. Normalmente se crea una tabla para cada tipo de datos de los que se compone la base de datos, y dentro de ella se almacena información relacionada con distintos conceptos de ese tipo de datos.

Una base de datos puede tener muchas tablas y aunque se crean como elementos independientes, pueden crearse relaciones entre las distintas tablas para recuperar datos de ellas mediante consultas o formularios.

Una vez que el Ayuntamiento de Madrid nos proporcionó la información relacionada con todos los recursos a incluir en la aplicación, tuvimos que organizarla por cada tipo de recurso y observar detenidamente la variedad de campos que incluían, la variedad de tipos de datos e incluso la variedad de formatos en los que se recibió la información. Además debíamos hacernos varias preguntas para poder diseñar la que sería la base de datos del sistema.

Primeramente debíamos evitar la repetición de los datos. Si creábamos una tabla por cada tipo de recurso, por ejemplo cada registro haría referencia a un recurso, luego no repetiríamos información. En segundo lugar, debíamos ver los campos que podía contener cada tabla. Nos dimos cuenta que en recursos como los puntos limpios o en los aparcamientos había mucha variedad en los campos dependiendo de si el aparcamiento era de coches, de motos o de bicis. Es decir, si queríamos unificar estos tres recursos en una misma tabla, era probable que en muchos casos se quedasen muchos campos vacíos o en blanco, por lo que debíamos evitarlo. Además sólo en aparcamientos de bicis necesitaríamos 1015 registros, lo que sumado a los aparcamientos de motos y de coches daban un gran número de registros para almacenar en una misma tabla, lo que podría añadirnos lentitud a las consultas. Debido a estos dos factores nos decidimos a separar un mismo tipo de recurso en varias tablas. En caso de los aparcamientos tendremos tres tablas: para coches, motos y bicis, y en el caso de los puntos limpios también tendremos tres tablas, para fijos, móviles y contenedores de ropa. Para el caso de los puntos de suministro, debido a que no iban a contener un gran número de

registros y además podíamos adaptar los campos para los distintos tipos de combustible y sus características, finalmente los dejamos en una misma tabla.

En último lugar teníamos que ver si debía existir relación entre las tablas. Un usuario se registra en la base de datos, y realiza consultas sobre los recursos almacenados en otras tablas, pero no necesitábamos almacenar los recursos que ha buscado ni tener ninguna relación entre los diferentes recursos. Por ello decidimos que las tablas no tendrían ninguna relación entre ellas ya que no era necesario.

La base de datos contiene 9 tablas, y salvo la tabla *usuario*, las demás tablas contienen unos campos cuya función es común. Por ello vamos a explicar primero qué almacena la tabla de usuarios, a continuación los campos comunes al resto de las 8 tablas, y después trataremos cada una de estas tablas por separado para comentar sus campos.

Si de algún campo no se tiene información, se rellenará siempre con un valor de -1 para indicar que esa información no debe mostrarse al usuario, ya que no se dispone de ella, y además evitamos el uso de null que puede traer varios errores en el funcionamiento de la aplicación.

❖ Tabla *usuario*

La tabla *usuario* almacena información sobre todos los usuarios que acceden a la aplicación. Cada usuario accede a la aplicación previo registro. Los datos que se piden al usuario se muestran en la figura 29, y se explican a continuación:

Field Name	Field Type	Size	Precision	Not Null	Unsigned	Default
idUsuario	INTEGER	11	0	☑	☐	Null
email	VARCHAR	100	0	☑	☐	Null
password	VARCHAR	50	0	☑	☐	Null
sexo	VARCHAR	20	0	☐	☐	Null
añonacimiento	INTEGER	11	0	☐	☐	Null

Figura 29. Tabla usuario

- *email*: se trata de un campo de texto de tipo varchar que permite hasta 100 caracteres. En él se almacena el email de cada usuario. Además este campo es único (propiedad unique) indicado por el rombo de color azul en la figura 29, es decir, en la tabla no pueden aparecer dos registros con el mismo campo email. Esto es debido a que el email se usa para que el usuario pueda entrar en la aplicación, y un mismo usuario solo podrá registrarse en la aplicación una vez con el mismo email. Este campo no podrá ser null ya que es usado para el login del usuario.

- *password*: se trata de un campo de texto de tipo varchar que permite hasta 50 caracteres. En él se almacena la contraseña del usuario para entrar en la aplicación. Además este campo no podrá ser null ya que es usado para el login del usuario.

- *sexo*: se trata de un campo de texto de tipo varchar que permite hasta 20 caracteres. En él se almacena el sexo del usuario según sea “Hombre” o “Mujer”.

- *añonacimiento*: se trata de un campo de texto de tipo integer. En él se almacena el año de nacimiento del usuario.

Además de estos campos tenemos el campo *idUsuario*. Este campo se rellena solo cuando el usuario se registra en el sistema. Es simplemente un número identificativo de tipo integer, y es la clave primaria de la tabla para permitirnos identificar a cada usuario de forma única, es decir, no habrá dos usuarios con el mismo id. Este campo no podrá ser null nunca.

Un ejemplo de varios registros incluidos en la tabla *usuarios* puede verse en la figura 30.

idUsuario	email	password	añonacimiento	sexo
1	anaalfaroorgaz@hotmail.com	[redacted]	1.988	Mujer
2	sergioballes0@gmail.com	[redacted]	1.988	Hombre
3	lydwig@gmail.com	[redacted]	1.988	Mujer

Figura 30. Ejemplo de registros incluidos en la tabla usuario

❖ Campos comunes

Las 8 tablas restantes de la base de datos contienen información acerca de diferentes recursos usados en la aplicación. Cada tabla contendrá los campos propios y necesarios para cada recurso, pero todas las tablas poseen unos campos específicos para el funcionamiento de la aplicación que son:

- *id*: este campo es simplemente un número identificativo para cada registro, de tipo integer y es la clave primaria de cada tabla para permitirnos identificar a cada registro de forma única, es decir, no habrá dos registros con el mismo id en una misma tabla. Este campo no podrá ser null nunca.

- *longitud* y *latitud*: estos dos campos se refieren a las coordenadas geográficas de cada recurso para poder mostrarlo en el mapa. Estos dos campos son muy importantes ya que es la forma de ofrecer y visualizar los recursos en el cliente. Son de tipo decimal, puesto que es necesario preservar la precisión exacta. Si perdiésemos precisión, entonces la ubicación de cada recurso en el mapa sería muy poco exacta o incluso errónea. Este tipo de datos requiere dos atributos relacionados con la dimensión: el tamaño y la precisión. Estos campos tendrán un tamaño de 65 y una precisión de 30. El tamaño representa el número de dígitos que pueden almacenarse a continuación del punto decimal, y la precisión representa el número de dígitos decimales significativos que se almacenan para los valores. Estos dos atributos son la capacidad máxima que nos permite MySQL, y decidimos su uso de esta forma para asegurarnos que todos los

decimales especificados en las coordenadas se almacenaban correctamente y no se realizaba ningún tipo de truncamiento del número o redondeo. El campo latitud siempre comenzará por 40 y el campo longitud por -3, ya que son las medidas que delimitan la ciudad de Madrid.

- *ndp*: este campo se refiere a unas direcciones para cada recurso, denominadas "direcciones normalizadas", y siguen un formato determinado, identificándolas mediante un código llamado NDP. El Ayuntamiento de Madrid requirió este campo ya que el uso de las direcciones debía realizarse a partir del Callejero Municipal que mantiene el Servicio de Cartografía e Información Urbanística del Área de Urbanismo y Vivienda. Este campo es de tipo texto varchar, de tamaño 20 ya que los ndp son códigos de 8 caracteres. Es leído y tratado de forma semejante a los demás campos, pero no se usa dentro de la aplicación y en todos los registros de todas las tablas de la base de datos se han introducido con un valor por defecto 0 (para evitar el uso de null). Este campo se ha incluido ya que en un futuro el Ayuntamiento de Madrid y la Informática del Ayuntamiento de Madrid pueden precisar de este registro para aportar mayor precisión a la ubicación de los recursos.

❖ Tabla *parque*

La tabla *parque* almacena información sobre los distintos parques de la ciudad de Madrid.

Field Name	Field Type	Size	Precis...	Not Null	Unsig...	Default
idParque	INTEGER	11	0	☑	☐	Null
nombre	VARCHAR	100	0	☑	☐	Null
superficie	VARCHAR	20	0	☑	☐	Null
longitud	DECIMAL	65	30	☑	☐	Null
latitud	DECIMAL	65	30	☑	☐	Null
zona_infantil	TINYINT	1	0	☐	☐	Null
zona_mayores	TINYINT	1	0	☐	☐	Null
zona_deportiva	TINYINT	1	0	☐	☐	Null
carril_bici	TINYINT	1	0	☐	☐	Null
circuito_biosaludable	TINYINT	1	0	☐	☐	Null
pipican	TINYINT	1	0	☐	☐	Null
punto_informacion	TINYINT	1	0	☐	☐	Null
cierre	TINYINT	1	0	☐	☐	Null
horario_cierre	VARCHAR	300	0	☐	☐	Null
aseo_publico	TINYINT	1	0	☐	☐	Null
ndp_parque	VARCHAR	20	0	☐	☐	Null

Figura 31. Tabla parque

Los campos que se incluyen en esta tabla se muestran en la figura 31 y se explican a continuación:

- *nombre*: para indicar el nombre del parque. Se trata de un campo de texto varchar que permite hasta 100 caracteres. No puede ser null.

- *superficie*: para indicar la superficie que ocupa el parque. Se trata de un campo de texto varchar de tamaño máximo 20 caracteres, y que no puede ser null. Hemos preferido que este campo sea de tipo texto y no de tipo numérico para añadir la unidad de medida en que está expresada la superficie, ya que esta unidad podría variar de un parque a otro según sus dimensiones.

- *zona_infantil*: se utiliza para indicar si el parque dispone de una zona infantil. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el campo no disponga de este servicio y 1 en caso afirmativo.

- *zona_mayores*: se utiliza para indicar si el parque dispone de una zona de mayores. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el campo no disponga de este servicio y 1 en caso afirmativo.

- *zona_deportiva*: se utiliza para indicar si el parque dispone de una zona deportiva. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el campo no disponga de este servicio y 1 en caso afirmativo.

- *carril_bici*: se utiliza para indicar si el parque dispone de carril bici. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el campo no disponga de este servicio y 1 en caso afirmativo.

- *circuito_biosaludable*: se utiliza para indicar si el parque dispone de un circuito biosaludable. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el campo no disponga de este servicio y 1 en caso afirmativo.

- *pipican*: se utiliza para indicar si el parque dispone de un pipican. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el campo no disponga de este servicio y 1 en caso afirmativo.

- *punto_información*: se utiliza para indicar si el parque dispone de al menos un punto de información. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el campo no disponga de este servicio y 1 en caso afirmativo.

- *aseo_publico*: se utiliza para indicar si el parque dispone de aseos públicos. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el campo no disponga de este servicio y 1 en caso afirmativo.

- *cierre*: se utiliza para indicar si el parque dispone de un horario de apertura al público. Este campo es de tipo tinyint de tamaño 1, y almacenamos 0 en caso de que el parque no cierre, es decir, su apertura sea de 24 horas y 1 en caso de que tenga un horario específico de apertura al público.

- *horario_cierre*: se utiliza para indicar el horario del parque. Es de tipo texto varchar de tamaño 300. Este tamaño lo ha dado el horario especificado más largo. Este campo almacenará "Horario 24H" en caso de que el campo anterior *cierre* sea 0, y en caso de que sea 1 almacenará el horario específico asociado a cada época del año.

Actualmente hay almacenados 189 parques en la base de datos.

❖ Tabla *parkcoche*

La tabla *parkcoche* almacena información sobre los distintos aparcamientos de coches de la ciudad de Madrid.

Field Name	Field Type	Size	Precision	Not Null	Unsign...	Default
idCoche	INTEGER	11	0	☑	☐	Null
distrito	VARCHAR	30	0	☐	☐	Null
tipo_uso	VARCHAR	20	0	☐	☐	Null
nombre	VARCHAR	40	0	☐	☐	Null
plaza_residente	INTEGER	11	0	☐	☐	Null
plaza_rotacion	INTEGER	11	0	☐	☐	Null
situacion	VARCHAR	50	0	☐	☐	Null
latitud	DECIMAL	65	30	☑	☐	Null
longitud	DECIMAL	65	30	☑	☐	Null
ndp_coche	VARCHAR	20	0	☐	☐	Null

Figura 32. Tabla parkcoche

Los campos que se incluyen en esta tabla se muestran en la figura 32 y se explican a continuación:

- *nombre*: para indicar el nombre del aparcamiento de coches. Se trata de un campo de texto varchar que permite hasta 40 caracteres.

- *distrito*: para indicar el distrito en el que está ubicado el aparcamiento de coches. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

- *situacion*: para indicar la dirección exacta en la que está ubicado el aparcamiento de coches. Se trata de un campo de texto varchar que permite hasta 50 caracteres.

- *tipo_uso*: para indicar si el tipo de uso del aparcamiento es de rotación o mixto; este último caso indica que el uso del aparcamiento está destinado a rotación y a residentes. Se trata de un campo de texto varchar que permite hasta 20 caracteres.

- *plaza_residente*: para indicar el número de plazas reservadas a residentes. Se trata de un campo numérico de tipo integer.

- *plaza_rotacion*: para indicar el número de plazas reservadas a rotación. Se trata de un campo numérico de tipo integer.

❖ Tabla *parkmoto*

La tabla *parkmoto* almacena información sobre los distintos aparcamientos de motos de la ciudad de Madrid.

Field Name	Field Type	Size	Precisi...	Not Null	Unsig...	Default
idMoto	INTEGER	11	0	☑	☐	Null
ubicacion	VARCHAR	60	0	☐	☐	Null
metros	DOUBLE	15	3	☐	☐	Null
capacidad	INTEGER	11	0	☐	☐	Null
latitud	DECIMAL	65	30	☑	☐	Null
longitud	DECIMAL	65	30	☑	☐	Null
ndp_moto	VARCHAR	20	0	☐	☐	Null

Figura 33. Tabla parkmoto

Los campos que se incluyen en esta tabla se muestran en la figura 33 y se explican a continuación:

- *ubicación*: para indicar la dirección donde se encuentra el aparcamiento. Se trata de un campo de texto tipo varchar que permite hasta 60 caracteres.

- *metros*: para indicar los metros que mide el aparcamiento. Se trata de un campo numérico doublé de tamaño 15 y precisión 3, es decir, permite hasta 3 decimales.

- *capacidad*: para indicar el número de motos total que pueden aparcarse. Se trata de un campo numérico de tipo integer.

Actualmente hay almacenados 411 aparcamientos de motos en la base de datos.

❖ Tabla *parkbici*

La tabla *parkbici* almacena información sobre los distintos aparcamientos de bicicletas de la ciudad de Madrid.

Field Name	Field Type	Size	Precisi...	Not Null	Unsig...	Default
idBici	INTEGER	11	0	☑	☐	Null
distrito	VARCHAR	30	0	☐	☐	Null
barrio	VARCHAR	30	0	☐	☐	Null
latitud	DECIMAL	65	30	☑	☐	Null
longitud	DECIMAL	65	30	☑	☐	Null
ndp_bici	VARCHAR	20	0	☐	☐	Null

Figura 34. Tabla parkbici

Los campos que se incluyen en esta tabla se muestran en la figura 34 y se explican a continuación:

- *distrito*: para indicar el distrito en el que está ubicado el aparcamiento de bicis. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

- *barrio*: para indicar el barrio en el que está ubicado el aparcamiento de coches. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

Actualmente hay almacenados 1015 aparcamientos de bicis en la base de datos.

❖ Tabla *ptolimpiofijo*

La tabla *ptolimpiofijo* almacena información sobre los distintos puntos limpios fijos de la ciudad de Madrid. Los puntos limpios fijos son instalaciones adecuadas para aquellos residuos que no pueden depositarse en los contenedores habituales de la calle, y que por ser peligrosos nunca deben mezclarse con los residuos domésticos.

Field Name	Field Type	Size	Precisi...	Not Null	Unsig...	Default
idPtoFijo	INTEGER	11	0	☑	☐	Null
distrito	VARCHAR	30	0	☐	☐	Null
direccion	VARCHAR	80	0	☐	☐	Null
observaciones	VARCHAR	80	0	☐	☐	Null
cod_postal	VARCHAR	10	0	☐	☐	Null
barrio	VARCHAR	30	0	☐	☐	Null
llegar_metro	VARCHAR	30	0	☐	☐	Null
llegar_bus	VARCHAR	30	0	☐	☐	Null
latitud	DECIMAL	65	30	☑	☐	Null
longitud	DECIMAL	65	30	☑	☐	Null
ndp_ptofijo	VARCHAR	20	0	☐	☐	Null

Figura 35. Tabla ptolimpiofijo

Los campos que se incluyen en esta tabla se muestran en la figura 35 y se explican a continuación:

- *distrito*: para indicar el distrito en el que está ubicado el punto limpio fijo. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

- *direccion*: para indicar la dirección en la que está ubicado el punto limpio fijo. Se trata de un campo de texto varchar que permite hasta 80 caracteres.

- *cod_postal*: para indicar el código postal de la dirección en que está ubicado el punto limpio fijo. Se trata de un campo de texto varchar que permite hasta 10 caracteres.

- *barrio*: para indicar el barrio en el que está ubicado el punto limpio fijo. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

- *observaciones*: para algunas indicaciones importantes sobre el acceso al punto limpio fijo. Se trata de un campo de texto varchar que permite hasta 80 caracteres.

- *llegar_metro*: para indicar las líneas de metro que el ciudadano puede usar para llegar al punto limpio fijo. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

- *llegar_bus*: para indicar las líneas de autobús que el ciudadano puede usar para llegar al punto limpio fijo. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

Además de estos campos hay otros 5 atributos comunes a todos los puntos limpios fijos, por lo que se han incluido directamente en la aplicación y no en la base de datos. Estos atributos son:

- El horario de los puntos limpios fijos de lunes a sábado: 08:00-20:00.
- El horario de los puntos limpios fijos los domingos y festivos: 09:00-14:00.
- El horario de los puntos limpios fijos los días 24 y 31 de diciembre: 08:00 – 14:00.
- Los días de cierre: 25 de diciembre, y 1 y 6 de enero.
- Más información, para indicar que el traslado de los residuos corre a cargo del ciudadano con sus propios medios de transporte.

Actualmente hay almacenados 16 puntos limpios fijos en la base de datos.

❖ Tabla *ptolimpiomovil*

La tabla *ptolimpiomovil* almacena información sobre los distintos puntos limpios móviles de la ciudad de Madrid. Los puntos limpios móviles son camiones municipales especialmente adaptados para que los ciudadanos puedan depositar en ellos ciertos residuos peligrosos. Se sitúan a diario en lugares estratégicos de la capital, según una serie de horarios y ubicaciones fijas.

Field Name	Field Type	Size	Precisi...	Not Null	Unsig...	Default
idPtoMovil	INTEGER	11	0	☑	☐	Null
distrito	VARCHAR	30	0	☐	☐	Null
direccion	VARCHAR	50	0	☐	☐	Null
dias_parada	VARCHAR	30	0	☐	☐	Null
horario	VARCHAR	50	0	☐	☐	Null
dias_norecorrido	VARCHAR	80	0	☐	☐	Null
latitud	DECIMAL	65	30	☑	☐	Null
longitud	DECIMAL	65	30	☑	☐	Null
ndp_ptomovil	VARCHAR	20	0	☐	☐	Null

Figura 36. Tabla ptolimpiomovil

Los campos que se incluyen en esta tabla se muestran en la figura 36 y se explican a continuación:

- *distrito*: para indicar el distrito en el el punto limpio móvil realiza la parada. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

- *dirección*: para indicar la dirección en la que el punto limpio móvil realiza la parada. Se trata de un campo de texto varchar que permite hasta 50 caracteres.

- *días_parada*: para indicar los días en los que el punto limpio móvil realiza una parada. Se trata de un campo de texto varchar que permite hasta 30 caracteres.

- *horario*: para indicar el horario en el que el punto limpio móvil va a estar parado en la dirección antes especificada. Se trata de un campo de texto varchar que permite hasta 50 caracteres.

- *días_norecorrido*: para indicar los días específicos en los que el punto limpio móvil no realiza parada. Se trata de un campo de texto varchar que permite hasta 80 caracteres.

Actualmente hay almacenados 58 puntos limpios móviles en la base de datos.

❖ Tabla *contropa*

La tabla *contropa* almacena información sobre los distintos contenedores de ropa de la ciudad de Madrid. Estos contenedores de ropa se ubican en distintas zonas de la ciudad para que los ciudadanos puedan donar la ropa que ya no necesitan, de forma que esta ropa recogida se clasifica y se destina a tiendas de segunda mano, a pequeños comercios en África, a su reciclaje o a vertederos apropiados.

Field Name	Field Type	Size	Precisi...	Not Null	Unsig...	Default
idCont	INTEGER	11	0	☑	☐	Null
distrito	VARCHAR	40	0	☐	☐	Null
direccion	VARCHAR	80	0	☐	☐	Null
latitud	DECIMAL	65	30	☑	☐	Null
longitud	DECIMAL	65	30	☑	☐	Null
ndp_contropa	VARCHAR	20	0	☐	☐	Null

Figura 37. Tabla contropa

Los campos que se incluyen en esta tabla se muestran en la figura 37 y se explican a continuación:

- *distrito*: para indicar el distrito en el que está ubicado el contendor de ropa. Se trata de un campo de texto varchar que permite hasta 40 caracteres.

- *direccion*: para indicar la dirección en la que está ubicado el contendor de ropa. Se trata de un campo de texto varchar que permite hasta 80 caracteres.

Actualmente hay almacenados 72 contenedores de ropa en la base de datos.

❖ Tabla *ptosuministro*

La tabla *ptosuministro* almacena información sobre los distintos puntos de suministro de la ciudad de Madrid. Estos puntos están destinados a suministro de combustibles ecológicos o

alternativos, entre los cuales están: gas licuado de petróleo (GLP), gas natural comprimido (GNC), bioetanol y puntos de carga eléctrica.

Field Name	Field Type	Size	Precisi...	Not Null	Unsig...	Default
id_ptosum	INTEGER	11	0	☑	☐	Null
tipo_combustible	VARCHAR	20	0	☑	☐	Null
nombre_estacion	VARCHAR	80	0	☑	☐	Null
latitud	DECIMAL	65	30	☑	☐	Null
longitud	DECIMAL	65	30	☑	☐	Null
distrito	VARCHAR	40	0	☐	☐	Null
ubicacion	VARCHAR	200	0	☐	☐	Null
comercializador	VARCHAR	40	0	☐	☐	Null
ndp_ptosum	VARCHAR	20	0	☐	☐	Null
observaciones	VARCHAR	300	0	☐	☐	Null

Figura 38. Tabla ptossuministro

Los campos que se incluyen en esta tabla se muestran en la figura 38 y se explican a continuación:

- *nombre_estacion*: para indicar el nombre del punto de suministro. Se trata de un campo de texto varchar que permite hasta 80 caracteres. Este campo no puede ser null.

- *tipo_combustible*: para indicar el tipo de combustible que suministra cada punto. Se trata de un campo de texto varchar que permite hasta 20 caracteres. Este campo será “gnc”, “glp”, “bioetanol”, o “eléctrico” para distinguir el combustible en cada caso. No puede ser null.

- *distrito*: para indicar el distrito en el que está ubicado el punto de suministro. Se trata de un campo de texto varchar que permite hasta 40 caracteres.

- *ubicacion*: para indicar la dirección en la que está ubicado el punto de suministro. Se trata de un campo de texto varchar que permite hasta 200 caracteres.

- *comercializador*: para indicar la empresa que surte el combustible. Se trata de un campo de texto varchar que permite hasta 40 caracteres.

- *observaciones*: para indicar cierta información de interés en el caso de los puntos de suministro eléctricos. Se trata de un campo de texto varchar que permite hasta 300 caracteres.

Actualmente hay almacenados 35 puntos de suministro en la base de datos, 1 de bioetanol, 1 de gnc, 9 de glp y 24 eléctricos.

5.2. Arquitectura del Servidor

El servidor se compone de dos proyectos Maven Project con naturaleza Spring (Spring Project Nature):

- MapaRecursosServer: incluye la implementación asociada a las capas de persistencia y negocio.
- MapaRecursosWS: incluye la implementación asociada a la capa de integración por parte del servidor.

❖ Mapa Recursos Server

• Dependencias con librerías

Para este proyecto son necesarias varias librerías. Gracias al framework Maven basta con especificar las librerías que necesitamos en el archivo pom.xml, y el solo se encarga de descargar estas dependencias de su repositorio. Este archivo se guarda en el directorio raíz del proyecto.

Dentro de este archivo pom.xml debemos indicar cómo queremos plegar este proyecto.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
       http://maven.apache.org/maven-v4_0_0.xsd">
         <modelVersion>4.0.0</modelVersion>
         <groupId>com.maparecursos</groupId>
         <artifactId>MapaRecursosServer</artifactId>
         <packaging>jar</packaging>
         <version>1.0</version>
         <name>MapaRecursosServer</name>
       <!—Espacio para todas las dependencias -->
</project>
```

Gracias a este código el proyecto MapaRecursosServer se plegará en un .jar, de forma que lo trataremos como una “nueva librería” y lo incluiremos en el proyecto MapaRecursosWS como explicaremos más adelante.

Las dependencias con librerías utilizadas y descritas en el archivo pom.xml siguen la siguiente estructura, como se muestra en el ejemplo para la librería del conector de la base de datos MySQL.

```
       <!-- MySQL database driver -->
       <dependency>
              <groupId>mysql</groupId>
              <artifactId>mysql-connector-java</artifactId>
              <version>5.1.9</version>
       </dependency>
```

Debemos indicar el nombre bajo el que se agrupan una serie de objetos, la identificación de la unidad administrada por el repositorio y la versión de la librería.

Las librerías necesarias en este proyecto MapaRecursosServer son:

Nombre de la librería y versión	Descripción
antlr-2.7.7	ANTLR: ANother Tool for Language Recognition (Otra herramienta para el Reconocimiento del Idioma). Proporciona un framework para construir reconocedores, compiladores, y traductores de las descripciones gramaticales que contienen Java, C + +, o C#. Librería necesaria para Hibernate.
asm-3.1	Librería para la persistencia de Hibernate.
cglib-2.2	CGLIB es una librería de generación de código de alto rendimiento y calidad. Es usada para extender clases Java e implementa interfaces en tiempo de ejecución. Es necesaria para Hibernate.
commons-collections-3.2.1	Librería de Apache que extiende o aumenta la librería de Colecciones de Java.
commons-logging-1.1.1	Librería de Apache para la generación de logs.
dom4j-1.6.1	Librería que ofrece soporte para API's de procesamiento de XML.
hibernate-annotations-3.3.0	Librería de Hibernate.
jdk-1.6	Librería java para el funcionamiento de Spring.
junit-3.8.1	Librería para realizar test con JUnit.
mysql-connector-java-5.1.9	Librería del conector de la base de datos MySQL usada.
spring-2.5.6	Librería de Spring.
xml-apis-1.0.b2	Contiene las clases java que necesita JDOM para dar formato al XML.

- ## Organización del proyecto

El proyecto contiene 3 directorios diferenciados para contener las clases y archivos necesarios. Estos directorios son:

- *src/main/resources*: contiene varios paquetes con los archivos xml que incluyen todas las configuraciones de Spring e Hibernate, además de un archivo de tipo *.properties* para las propiedades de la base de datos.
- *src/main/java*: contiene todas las clases java necesarias para el funcionamiento de la capa de persistencia y negocio.
- *src/test/java*: contiene2 paquetes que incluyen las clases usadas para los test sobre el funcionamiento de las capas de persistencia y negocio, y las clases para la carga de información a la base de datos.

✓ Directorio src/main/resources

Dentro de este directorio se incluyen 5 paquetes:

1. **Paquete *hibernate*:**

El paquete hibernate incluye un archivo xml por cada tabla de la base de datos.

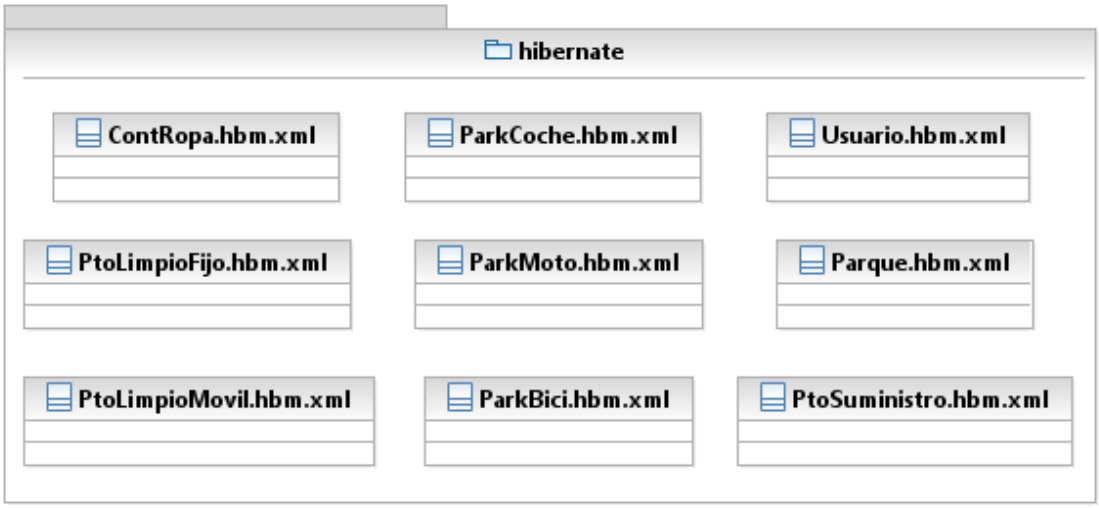


Figura 39. Contenido del paquete hibernate

Estos archivos xml realizan el mapeo de la base de datos; de esta forma en cada archivo xml se especifica el nombre de la tabla, sus correspondientes nombres de columna en la base de datos asociándolos con su tipo de datos y sus posibles propiedades, así como las relaciones entre las tablas. Además cada archivo xml se debe relacionar con la clase java correspondiente contenida en la carpeta *src/main/java*, dentro del paquete *com.maparecursos.dominio*.

Nuestra base de datos, como se ha explicado anteriormente, no posee relaciones entre las 9 tablas, es decir, cada tabla se trata de forma independiente y no poseen ninguna conexión entre ellas. Un ejemplo del mapeo de la tabla Usuario sería el siguiente:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.maparecursos.dominio.Usuario" table="usuario"
catalog="maparecursos">
        <id name="idUsuario" type="java.lang.Long">
            <column name="idUsuario" />
        </id>
        <property name="password" type="string">
            <column name="password" length="50" not-null="true" />
        </property>
        <property name="email" type="string">
            <column name="email" length="100" not-null="true"
unique="true"/>
```

```
        </property>
        <property name="sexo" type="string">
            <column name="sexo" length="20" />
        </property>
        <property name="añonacimiento" type="int">
            <column name="añonacimiento"/>
        </property>
    </class>
</hibernate-mapping>
```

Como observamos en el ejemplo, se indican como atributos dentro del elemento *<class>* la clase java dentro del paquete *com.maparecursos.dominio* (dentro de la carpeta *src/main/java*) llamada *Usuario* (.java) con la que se hará el mapeo para poder tratar la tabla de la base de datos como un POJO, es decir, como una clase java simple, y su posterior instancia. A continuación se describe el nombre de la tabla, en este caso la tabla usuario, y luego el nombre de la base de datos, *maparecursos*.

Dentro de este elemento *<class>* se incluye un elemento *<property>* por cada campo de la tabla, salvo para el campo id, que se indica dentro del elemento *<id>.* Dentro del elemento *<property>* o *<id>* se indican como atributos el nombre (*name*) correspondiente al atributo de la clase java simple (POJO) y su tipo de datos. Además dentro de cada elemento *<property>* o *<id>* se incluye el elemento *<column>* donde se indica el nombre exacto de la columna de ese campo en la tabla de la base de datos. También se incluye como atributo dentro de este elemento las posibles propiedades de cada campo.

Con un ejemplo del código anterior queda más claro. Por ejemplo, incluimos la propiedad con nombre *email*, cuyo tipo es String, dentro de la clase Usuario.java con la que se hará el mapeo. Este atributo de la clase java se debe relacionar con la columna de la tabla de la base de datos de nombre *email*, cuya longitud es de 100 caracteres máximo, no puede ser null y además esta columna es única (propiedad *unique*), lo que significa que dentro de la tabla no podrá haber dos registros con este campo idéntico.

Este proceso de mapeo se repite por cada tabla de la base de datos, y se obtienen 9 archivos nombrados de la forma *nombredelatabla.hbm.xml*, según se muestran en la figura anterior.

Los tipos declarados en estos archivos de mapeo no son tipos de datos Java ni tipos de base de datos MySQL. Estos tipos se llaman tipos de mapeo Hibernate, que pueden traducir tipos de datos de Java a MySQL y viceversa. En algunos casos puede evitarse la especificación del tipo de cada atributo, y Hibernate tratará de determinar el tipo correcto de conversión y de mapeo por sí mismo. Si el atributo type no se encuentra presente en el mapeo, hay que tener especial cuidado ya que en algunos casos esta detección automática (utilizando Java Reflection [65]) puede que no devuelva lo que esperamos, por ejemplo para una propiedad llamada date, puede llevar a confusión a Hibernate ya que existe un tipo específico de Java con ese nombre. Nosotros, para asegurarnos el buen funcionamiento, hemos preferido incluir todos los tipos.

2. **Paquete *spring.database*:**

Dentro de este paquete tenemos dos archivos xml. El primero de ellos se trata del archivo *Hibernate.xml* donde se debe indicar a Hibernate el archivo en el que está almacenada la

configuración para el conector de la base de datos, y cuál es ese conector. Además se debe indicar dónde se encuentran almacenados los archivos xml de mapeo, los comentados antes incluídos en el paquete *hibernate*.

Este archivo quedaría:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

	<!-- Hibernate session factory -->
<bean id="sessionFactory"
	class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
	<property name="dataSource">
		<ref bean="dataSource"/>
	</property>
	<property name="hibernateProperties">
		<props>
			<prop key="hibernate.dialect">
				org.hibernate.dialect.MySQLDialect</prop>
			<prop key="hibernate.show_sql">true</prop>
		</props>
	</property>
		<property name="mappingResources">
			<list>
				<value>/hibernate/Usuario.hbm.xml</value>
				<value>/hibernate/Parque.hbm.xml</value>
				<value>/hibernate/PtoSuministro.hbm.xml</value>
				<value>/hibernate/ParkBici.hbm.xml</value>
				<value>/hibernate/ParkMoto.hbm.xml</value>
				<value>/hibernate/ParkCoche.hbm.xml</value>
				<value>/hibernate/ContRopa.hbm.xml</value>
				<value>/hibernate/PtoLimpioFijo.hbm.xml</value>
				<value>/hibernate/PtoLimpioMovil.hbm.xml</value>
			</list>
		</property>
    </bean>
</beans>
```

Como observamos en el documento, se indica la creación de un nuevo bean donde se indica mediante el elemento *<property>* el nombre del archivo donde está almacenada la configuración del conector de la base de datos. Además indica el tipo de conector, que es MySQL. Y como se ha dicho, también incluye una lista con las rutas donde Hibernate va a encontrar los archivos xml para el mapeo de las tablas de la base de datos.

El otro archivo incluido en este paquete se llama DataSource.xml, y su código es el siguiente:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

	<bean class="org.springframework.beans.factory.
		config.PropertyPlaceholderConfigurer">
```

```
            <property name="location">
                  <value>properties/database.properties</value>
            </property>
      </bean>

      <bean                                                    id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property name="driverClassName"
                        value="${jdbc.driverClassName}" />
            <property name="url" value="${jdbc.url}" />
            <property name="username" value="${jdbc.username}" />
            <property name="password" value="${jdbc.password}" />
      </bean>
</beans>
```

Este archivo se usa para configurar el conector MySQL de la base de datos. Los datos requeridos para la conexión son driverClassName, url, username y password, los cuales se explicarán en el siguiente paquete. Estos datos son privados y necesarios para el uso y acceso de la base de datos. Los valores de cada uno podrían incluirse directamente en este archivo, pero por seguridad suelen almacenarse en un archivo aparte de extensión *.properties*. Como se indica al principio de este archivo xml, la localización de ese archivo de nombre *database.properties* se encuentra dentro del paquete *properties*. Además para cada dato se indica como valor (*value*) *el* nombre de la variable usada en ese documento *database.properties*, para que hibernate sepa exactamente qué valor corresponde con cada dato.

3. Paquete *properties*:

Dentro de este paquete se incluye el archivo *database.properties* antes mencionado. En este archivo se introducen los valores para la configuración del conector MySQL de la base de datos. Los datos requeridos son:

- *driverClassName*: es el nombre del controlador usado para la conexión con la base de datos.

- *url*: se trata de la url donde está almacenada la base de datos, para poder realizar la conexión. El formato general de esta url es:

  jdbc:mysql://[serverName[\instanceName][:portNumber]][;property=value[;property=value]]

  donde,

  - *jdbc:mysql://* (obligatorio) es el subprotocolo y es constante.

  - *serverName* (opcional) es la dirección del servidor con el que se establece la conexión. Puede ser un DNS o una dirección IP o bien un host local o 127.0.0.1 para el equipo local. Si no se especifica en la URL de conexión, es necesario especificar el nombre del servidor en la colección de propiedades.

- *instanceName* (opcional) es la instancia para establecer la conexión con *serverName*. Si no se especifica se establece una conexión con la instancia predeterminada.

- *portNumber* (opcional) es el puerto para establecer la conexión con *serverName*. El valor predeterminado es 1433. Si usa el valor predeterminado no es necesario especificar el puerto ni el signo ":" precedente en la dirección URL.
  Para obtener un rendimiento óptimo de la conexión se debe establecer *portNumber* al conectar con una instancia con nombre. De este modo se evita un ciclo de ida y vuelta en el servidor para determinar el número de puerto. Si se usan *portNumber* e *instanceName*, *portNumber* tiene prioridad e *instanceName* se omite.

- *property* (opcional) es una o más propiedades de conexión. Se puede especificar cualquier propiedad de una larga lista. Las propiedades sólo se pueden delimitar mediante punto y coma (";") y no se pueden duplicar. No se va a comentar ningún tipo de propiedades, ya que son numerosas y nosotros no hemos hecho uso de ellas. Los siguientes datos que faltan para la conexión, el nombre de usuario y la contraseña, pueden incluirse también en la url como propiedades separadas por ";" como se ha explicado.

- *username*: el nombre de usuario correspondiente para acceder a la base de datos.

- *password*: la contraseña necesaria para acceder a la base de datos.

Por lo tanto, el contenido de este archivo *database.properties* es el siguiente:

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://ec2-50-19-213-178.compute-1.amazonaws.com:3306/maparecursos
jdbc.username=proyssii
jdbc.password=****
```

La contraseña de la base de datos se ha ocultado por seguridad de la misma.

La url se corresponde con la proporcionada por Cloudbees, donde se ha almacenado la base de datos como se comentó en el apartado dedicado a esta web.

4. Paquete *spring.beans*:

Dentro de este paquete encontramos un archivo xml por cada archivo xml de mapeo que realizamos anteriormente dentro del paquete *hibernate*, es decir, un archivo xml por cada tabla de la base de datos. Estos archivos xml son para crear un bean de Spring. Todos estos bean podrían declararse en un mismo archivo xml, pero hemos preferido separarlos asociados a cada tabla de la base de datos, por comodidad y sencillez. Por cada servicio que se quiere proporcionar para cada tabla se creará un bean, es decir, un objeto de esa clase que presta el

servicio que creará Spring. Y además por cada Objeto de Acceso a Datos necesario en cada servicio también se creará un bean, es decir, una instancia de la clase que proporciona ese acceso a los datos de cada tabla.

Este archivo se llama en cada caso *NombreTabla.xml*. Y por ejemplo, el archivo *Usuario.xml* para la tabla usuario, sería el siguiente:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

      <!-- Stock business object -->
   <bean id="servicioUsuario"
        class="com.maparecursos.negocio.servicio.impl.ServicioUsuarioImpl" >
            <property name="usuarioDao" ref="usuarioDao" />
   </bean>

   <!-- Stock Data Access Object -->
   <bean id="usuarioDao"
        class="com.maparecursos.persistencia.dao.impl.UsuarioDaoImpl" >
            <property
                  name="sessionFactory" ref="sessionFactory">
            </property>
   </bean>
</beans>
```

Como se observa, la clase que accede a los datos de la base de datos se llama *UsuarioDaoImpl*, almacenada en el paquete *com.maparecursos.persistencia.dao.impl*, y creamos una instancia de ella llamada *usuarioDao*. Y la clase que presta el servicio se llama *ServicioUsuarioImpl*, almacenada en el paquete *com.maparecursos.negocio.servicio.impl*, y creamos una instancia de ella llamada *servicioUsuario*. Además, este bean usará el servicio anterior de *usuarioDao*. El contenido y uso de estas clases java se comentará más adelante. La propiedad de *sessionFactory* maneja la sesión del ciclo de vida del bean. El atributo *id* del elemento *<bean>* nos permite en cada caso anotar un identificador para poder referenciar cada bean y obtenerlo a través de la interfaz de *ApplicationContext* de Spring como veremos más adelante.

5. Paquete *spring.config*:

Por último, este paquete incluye un archivo xml llamado BenLocations.xml donde se debe indicar a Spring donde está almacenada la configuración para el conector de la base de datos, y donde están almacenados los bean para la creación de las instancias de todos los servicios.

El contenido de este archivo sería:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

      <!-- Database Configuration -->
      <import resource="../database/DataSource.xml"/>
      <import resource="../database/Hibernate.xml"/>
```

```
        <!-- Beans Declaration -->
        <import resource="../beans/Usuario.xml"/>
        <import resource="../beans/Parque.xml"/>
        <import resource="../beans/PtoSuministro.xml"/>
        <import resource="../beans/ParkBici.xml"/>
        <import resource="../beans/ParkMoto.xml"/>
        <import resource="../beans/ParkCoche.xml"/>
        <import resource="../beans/ContRopa.xml"/>
        <import resource="../beans/PtoLimpioFijo.xml"/>
        <import resource="../beans/PtoLimpioMovil.xml"/>
</beans>
```

✓ Directorio src/main/java

Dentro de este directorio se incluyen 8 paquetes, los cuales comienzan por *com.maparecursos.**,* lo que a partir de ahora se obviará para sencillez en la explicación.

En figura 40 podemos observar la agrupación lógica que se ha realizado.

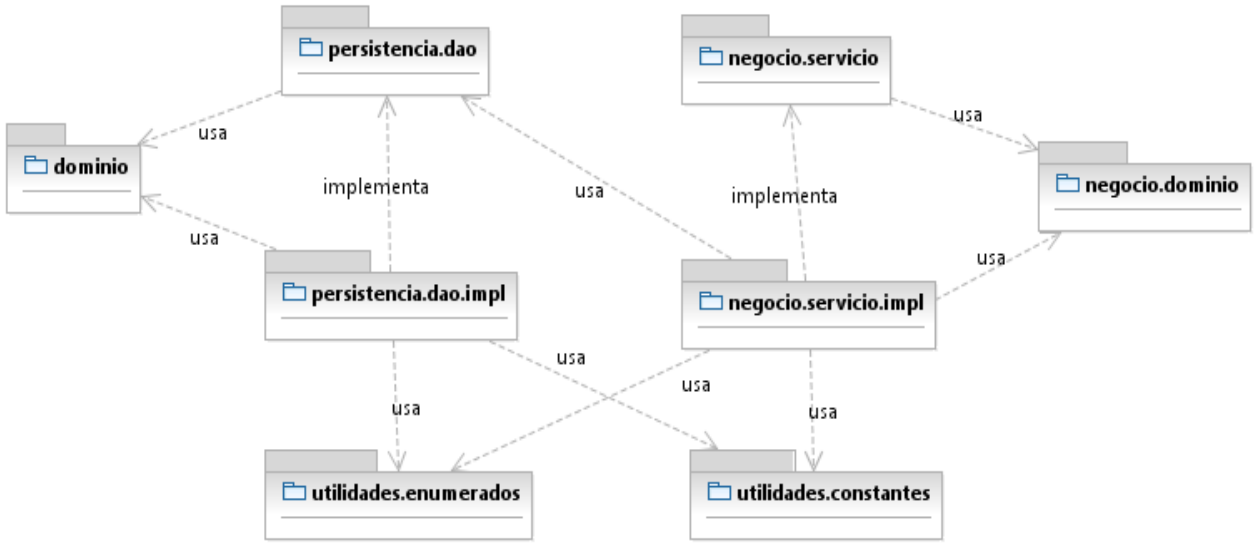


Figura 40. Diagrama paquetes del directorio src/main/java

Los paquetes de dominio, persistencia.dao y persistencia.dao.impl, pertenecen a la capa de persistencia. En ellos se aplicará el patrón DAO (Data Access Object), es decir, se ocuparán del acceso a los datos de la base de datos como se explicará más adelante. Los paquetes de negocio.dominio, negocio.servicio y negocio.servicio.impl pertenecen a la capa de negocio. En ellos se aplicará el patrón DTO (Data Transfer Object), cuya función es sólo almacenar los datos que se requieren en la base de datos, de forma que conseguimos separar la parte de persistencia y la parte de negocio, es decir, la parte del modelo y la del controlador, teniendo la ventaja de que si debe producirse algún cambio en alguna de estas capas no afectará a la otra.

Vamos a ir hablando de cada paquete de forma independiente, y a su vez de cómo se relacionan entre ellos.

1. Paquete *dominio*

Dentro de este paquete residen las clases java encargadas de realizar el mapeo con las tablas de la base de datos. Es decir, en él residen las clases java POJO relacionadas con los xml dentro de la carpeta src/main/resources y dentro del paquete hibernate. Tendremos entonces 9 clases java, una por cada tabla de la base de datos, y estas clases java simples deben incluir un atributo por cada campo de la tabla, de forma que el nombre de cada atributo se corresponda con el que ya dimos en el archivo de mapeo xml correspondiente. Estos atributos deben tener visibilidad privada. Además la clase java debe incluir un método get y set para cada atributo.

Para continuar con el ejemplo para la tabla de nombre *usuario*, vamos a ver de nuevo el archivo xml y la clase java simple relacionadas en este mapeo, en concreto el mapeo realizado para el atributo email, para que la explicación quede lo más clara posible.

Dentro del archivo Usuario.hbm.xml teníamos:

```
< hibernate-mapping >
    <class name="com.maparecursos.dominio.Usuario"
           table="usuario"
           catalog="maparecursos">
      …
      <property name="email" type="string">
            <column name="email" length="100"
                  not-null="true" unique="true"/>
      </property>
      …
   </class>
</hibernate-mapping>
```

Y dentro de la clase Usuario.java tendremos:

```
public class Usuario implements Serializable {

      private Long idUsuario;
      private String password;
      private String email;
      private String sexo;
      private int añonacimiento;

      public Usuario() {}

      public Usuario(Long idUsuario, String password, String email,
                        String sexo, int añonacimiento) {
            super();
            this.idUsuario = idUsuario;
            this.password = password;
            this.email = email;
            this.sexo = sexo;
            this.añonacimiento = añonacimiento;
      }
      public String getEmail() {
            return email;
      }
      public void setEmail(String email) {
            this.email = email;
      }
```

```
//...
}
```

En esta clase se han eliminado el resto de métodos getter y setter, por simplicidad.

Para realizar el mapeo, en el arhivo xml le decimos a Hibernate que la columna de la tabla de la base de datos *email* se corresponde con la propiedad *email*, es decir, le estamos diciendo a Hibernate qué métodos get y set debe usar cuando cree una instancia de esta clase Usuario.java, los llamados get*Email* y set*Email*. Lo mismo para los demás atributos de esta clase, y para las demás clases con sus respectivos atributos.

Incluimos una figura con las clases java POJO contenidas en este paquete con sus atributos correspondientes. Estas clases no se relacionan entre sí pero todas tienen algo en común, implementan a la interfaz *Serializable*. Normalmente existen varias diferencias entre un JavaBean y un POJO, y un POJO no debería implementar la interfaz *Serializable*, pero en este caso es válido entender un JavaBean como un POJO y, por ello, es factible implementar esta interfaz *Serializable*.

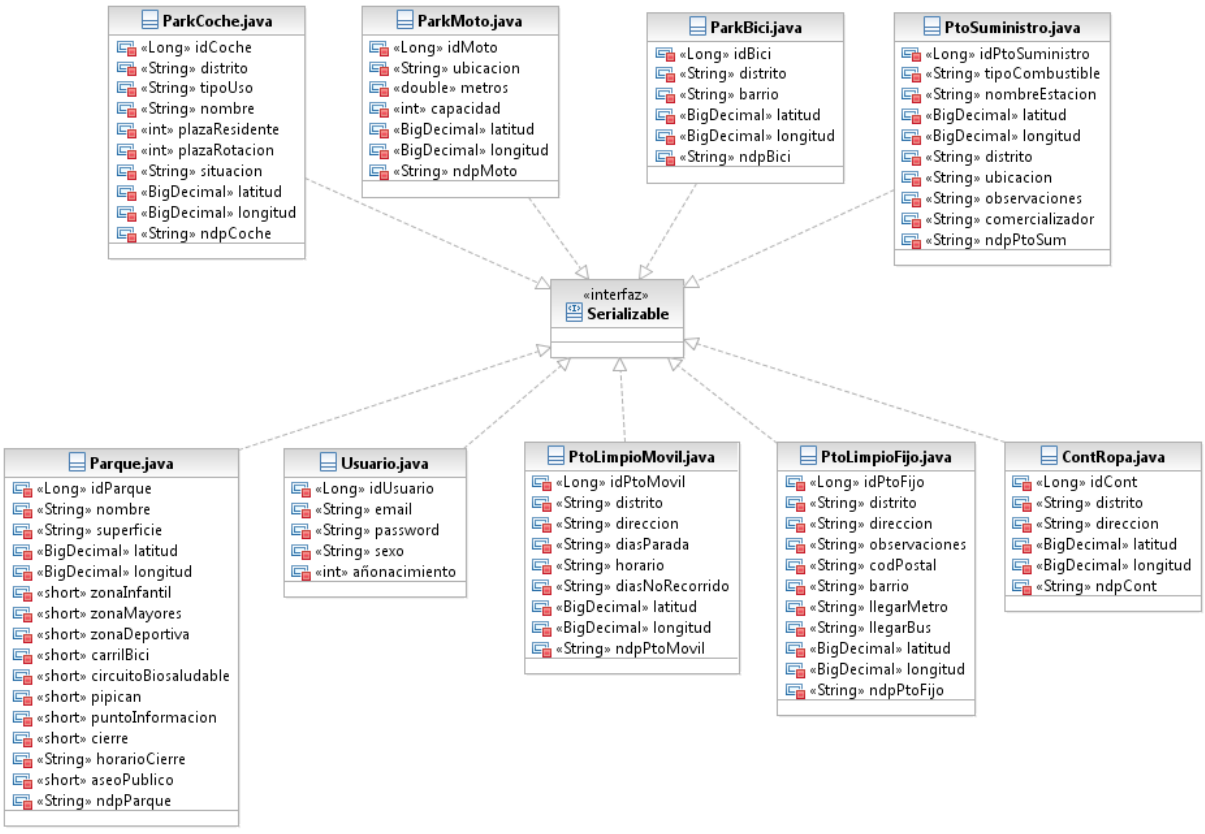


Figura 41. Entidades POJO

Como vemos en la figura 42 tenemos 9 clases POJO, una por cada tabla de la base de datos para poder realizar el mapeo. Cada clase contendrá un atributo por cada campo de la tabla a la que se refiere. La equivalencia del tipo varchar de la base de datos se corresponde con el tipo string de java, el tipo integer (11) con el tipo Long, el tipo decimal (65,30) con el tipo de java BigDecimal (para mayor precisión), y el tipo tinyint con el tipo short. Además otros tipos

integer lo hacemos corresponder con el tipo int de java, y el tipo double(15,3) lo hacemos corresponder con el tipo doublé.

2. Paquetes *persistencia.dao* y *persistencia.dao.impl*

Estos paquetes son los que se encargan de la persistencia de los datos, es decir, las clases que contienen son las encargadas de acceder a la base de datos. Para ellos usamos el **patrón DAO**. El patrón DAO indica que por cada clase de dominio que usamos (las clases java POJO anteriores) necesitamos crear un DAO o Objeto de Acceso a Datos (Data Access Object), que se encargue de persistir o buscar la información en la base de datos.

Creamos una interfaz DAO genérica con los principales métodos que necesitamos para manipular cada una de nuestras entidades en la base de datos. Estas operaciones serán del tipo añadir nuevos datos a la base de datos, eliminarlos, modificarlos o realizar búsquedas. Estas interfaces irán dentro del paquete persistencia.dao.

A partir de aquí tendremos otras clases java dentro del paquete persistencia.dao.impl que implementarán a cada una de las interfaces anteriores, y por ello incluirán la funcionalidad a realizar en cada método de acceso a la base de datos. Estas clases deben extender la clase HibernateDaoSupport. Spring nos provee esta clase HibernateDaoSupport para brindarle a nuestros DAO soporte para Hibernate. Esta clase ofrece varias utilidades para manipular la sesión de Hibernate, y se encarga de manejar automáticamente las transacciones. En particular, uno de los métodos más útiles que provee es getHibernateTemplate(). Este método devuelve un template con varios métodos útiles, que simplifican el uso de Hibernate. Los métodos de este template usados son save (para guardar datos en la base de datos), merge (para modificar datos), delete (para borrar registros de la base de datos) y find (para ejecutar consultas a la base de datos). Además estos métodos suelen encapsular varias excepciones java propias de acceso a datos de Hibernate, por lo que deben ir dentro de un try/catch para capturar esas posibles excepciones.

Vamos a pasar a comentar cada interfaz con la clase que la implementa por separado, ya que es importante conocer qué métodos de acceso a la base de datos se realizan en cada caso y para cada tabla de la base de datos.

En algunos casos los métodos devolverán un tipo entero que indica si la acción se ha realizado correctamente, con un 0, y en otro caso el número entero indicará cuál ha sido el error correspondiente. La asociación de estos números con el significado de cada error se explicará en los paquetes de *utilidades*.

Hay tres métodos básicos que son comunes a todas las interfaces. La única diferencia es el tipo de objeto que se pasa por parámetro, que variará según la tabla con la que estemos trabajando. Estos métodos son:

- **save**: coge un objeto del tipo específico, entidad, y lo almacena en la tabla usuarios en la base de datos gracias al método getHibernateTemplate().save(entidad). Devuelve un número entero para indicar el resultado de la acción.

- **update**: coge un objeto del tipo específico, entidad, y actualiza sus nuevos atributos gracias al método getHibernateTemplate().merge(entidad). Devuelve un número entero para indicar el resultado de la acción.

- **delete**: coge un objeto del tipo específico, entidad, y lo elimina de la base de datos gracias al método getHibernateTemplate().delete(entidad). Devuelve un número entero para indicar el resultado de la acción.

❒ Clases UsuarioDao.java y UsuarioDaoImpl.java

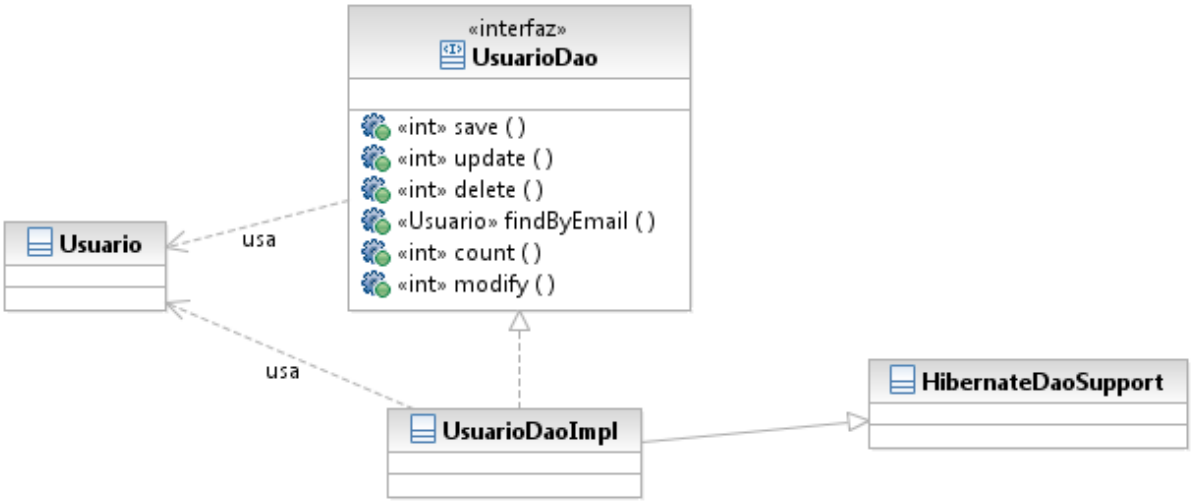


Figura 42. Persistencia Usuario

La interfaz *UsuarioDao* y la clase *UsuarioDaoImpl* usan la clase *Usuario.*Además la clase *UsuarioDaoImpl* implementa la interfaz *UsuarioDao*, o lo que es lo mismo, los métodos definidos en la interfaz. Estos métodos son, además de los 3 comunes citados antes:

- **findByEmail**: coge un string con el email del usuario que se quiere recuperar, y lo busca en la base de datos gracias al método ,

```
List<Usuario> listado =
   getHibernateTemplate().find("from Usuario where email = ?", email);
```

donde se ordena recuperar de la tabla Usuario los registros cuyo campo email sea igual al email especificado. Devuelve un número entero para indicar el resultado de la acción.

- **count**: coge un string con el email del usuario que se quiere recuperar, y lo busca en la base de datos gracias al método ,

```
int resultado = DataAccessUtils.intResult(
      getHibernateTemplate().find("select count(*) from Usuario")
      );
```

donde se ordena recuperar de la tabla Usuario el número de registros que haya. Devuelve un número entero para indicar el resultado de la acción.

- **modify**: hace lo mismo que el método update, es más, este método solo contiene una llamada a this.update(entidad), para no llamar directamente desde los servicios al método update.

❒ Clases ParqueDao.java y ParqueDaoImpl.java

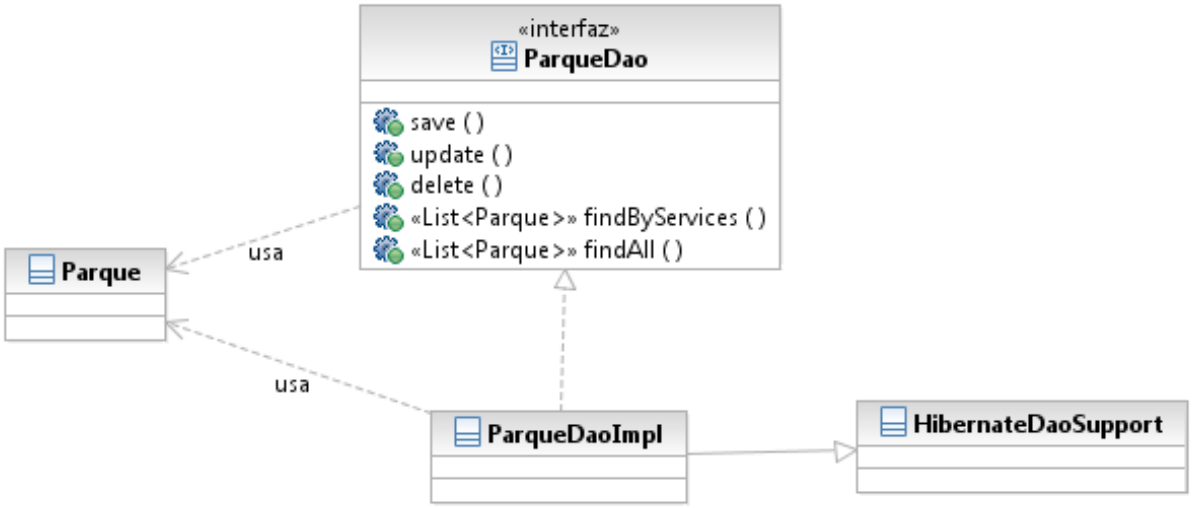


Figura 43. Persistencia Parque

La interfaz *ParqueDao* y la clase *ParqueDaoImpl* usan la clase *Parque*. Además la clase *ParqueDaoImpl* implementa la interfaz *ParqueDao* que implementa los métodos definidos en la interfaz. Estos métodos son, además de los 3 comunes citados antes:

- **findByServices**: coge un string con todos los servicios de los parques que queremos filtrar y busca en la base de datos los parques que cumplan todos esos servicios gracias al método ,

```
List<Parque> listaResultados
      = getHibernateTemplate().find("from Parque where "+ services);
```

  donde services es del tipo “zonaInfantil=1 and zonaMayores=1”. Devuelve una lista con todos los parques que cumplan esos servicios.

- **findAll**: gracias al método ,

```
List<Parque> listaResultados
      = getHibernateTemplate().find("from Parque");
```

  se ordena recuperar de la tabla Parque de la base de datos todos sus registros. Devuelve una lista con todos ellos.

- **Clases ParkCocheDao.java y ParkCocheDaoImpl.java,**
  Clases ParkMotoDao.java y ParkMotoDaoImpl.java,
  Clases ParkBiciDao.java y ParkBiciDaoImpl.java,
  Clases PtoLimpioFijoDao.java y PtoLimpioFijoDaoImpl.java,
  Clases PtoLimpioMovilDao.java y PtoLimpioMovilDaoImpl.java, y
  Clases ContRopaDao.java y ContRopaDaoImpl.java

El grupo de estos 6 recursos se comportan igual. La interfaz *"Recurso"Dao* y la clase *"Recurso"DaoImpl* usan la clase *"Recurso"* correspondiente. Además la clase *"Recurso"DaoImpl* implementa la interfaz *"Recurso"Dao* y los métodos definidos en ella. Junto con los 3 comunes citados antes, estas interfaces añaden un método para recuperar todos los registros incluidos en cada una de las tablas de la base de datos:

- **findAll**: gracias al método ,

```
List<"Recurso"> listaResultados
      = getHibernateTemplate().find("from TablaRecurso");
```

se ordena recuperar de la tabla correspondiente a cada recurso de la base de datos todos sus registros. Devuelve una lista con todos ellos.

Mostramos el diagrama común correspondiente a todos ellos. Habría que sustituir "Recurso" por cada tipo correspondiente en cada caso.

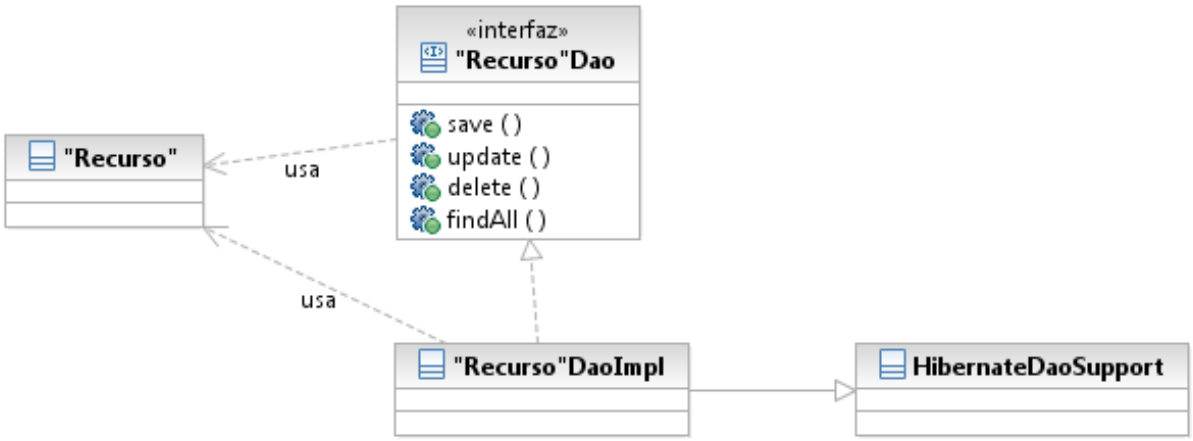


Figura 44. Persistencia "Recurso"

❒ **Clases PtoSuministroDao.java y PtoSuministroDaoImpl.java**

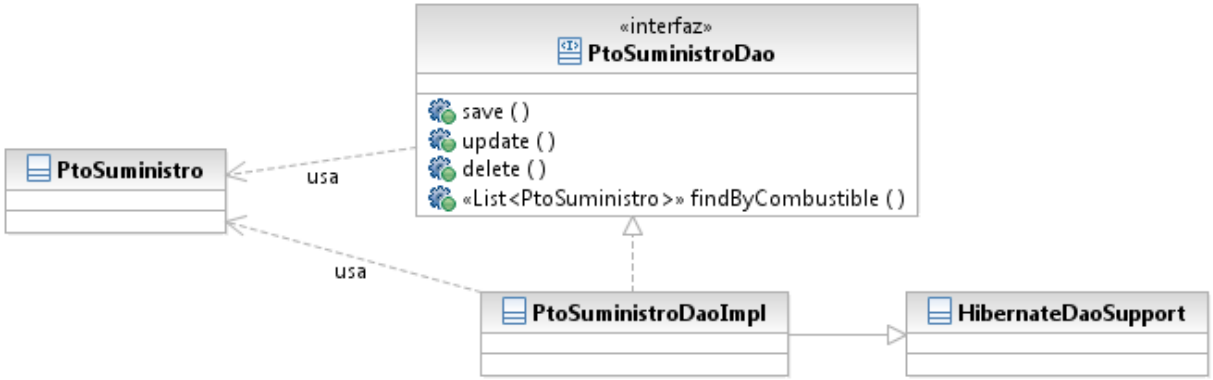


Figura 45. Persistencia PtoSumnistro

La interfaz *PtoSuministroDao* y la clase *PtoSuministroDaoImpl* usan la clase *PtoSuministro*. Además la clase *PtoSuministroDaoImpl* implementa la interfaz *PtoSuministroDao* y los métodos definidos en ella. Junto con los 3 comunes citados antes, incluye el método:

- **findByCombustible**: coge un string con el tipo de combustible de los registros que queremos recuperar, y gracias al método ,

```
List<PtoSuministro> listaResultados =
      getHibernateTemplate().find("from PtoSuministro where
                                  tipo_combustible = ?", combustible);
```

  los recupera de la base de datos. Devuelve una lista con todos los puntos de suministro que sean de ese tipo de combustible.

3. Paquete *negocio.dominio*

En este paquete incluimos las clases DTO. Estas clases son similares a las del paquete de dominio. Esto se hace para poder tener separadas la capa de persistencia de la de negocio.

Son 9 clases, una por cada tabla de la base de datos, y su nombre es “Tabla”DTO.java, donde “Tabla” lo sustituiremos por cada tabla diferente. Estas clases incluyen los mismos atributos que incluían en las clases similares del paquete dominio, y sus correspondientes métodos getter y setter. Además estas clases también implementan la interfaz Serializable como sus clases similares.

4. Paquete *negocio.servicio* y Paquete *negocio.servicio.impl*

Dentro del paquete negocio.servicio tendremos una interfaz con los métodos asociados a cada tabla de la base de datos, pero estos métodos ya no son los métodos de acceso a la base de datos sino que son los necesarios para el funcionamiento de la aplicación, es decir, los métodos específicos para la demanda del cliente. Y dentro del paquete negocio.servicio.impl,

tenemos las clases que implementan estas interfaces. Estas clases ya no deben extender ninguna clase de Hibernate, ahora deben tener un atributo del tipo de la interfaz Dao que corresponda, y así a partir de este objeto se hacen las llamadas a los métodos que implementamos antes de acceso a la base de datos.

Dentro de cada clase de implementación necesitaremos un método privado de conversión de objetos de tipo de la base de datos a tipo DTO. Esto es debido a que los objetos recuperados serán del tipo de la base de datos (ya que los conseguimos a través de los métodos de la capa de persistencia), pero los métodos que estamos implementando pertenecen a la capa de negocio, por lo que debemos devolver objetos DTO. Luego cuando conseguimos la información que viene de la base de datos la convertimos a objetos de tipo DTO. El código para esta conversión es el siguiente, y será igual en cada clase modificando el tipo "Recurso" por el conveniente en cada caso. Además debemos realizar el set"Atributo" con el get"Atributo" para cada atributo contenido en la clase.

```
private "Recurso"DTO convertBDToDTO ("Recurso" bd) {

            "Recurso"DTO dto = new "Recurso"DTO();
            dto.set"Atributo1"(bd.get"Atributo1"());
            dto.set"Atributo2"(bd.get"Atributo2"());
            return dto;
      }
```

Y el proceso contrario, es decir, si el cliente nos proporciona un objeto de tipo DTO y tenemos que llamar a un método de la capa de persistencia que requiere un objeto de tipo de la base de datos, entonces necesitamos el método inverso para convertir el objeto.

```
public "Recurso" convertDTOToBD ("Recurso"DTO dto) {
            "Recurso" bd = new "Recurso" ();
            bd.set"Atributo1"(dto.get"Atributo1"());
            bd.set"Atributo1"(dto.get"Atributo1"());
            return bd;
      }
```

Vamos a ver clase a clase dentro de los paquetes negocio.servicio y negocio.servicio.impl para explicar los nuevos métodos de la interfaz.

❒ **Clases ServicioUsuario.java y ServicioUsuarioImpl.java**

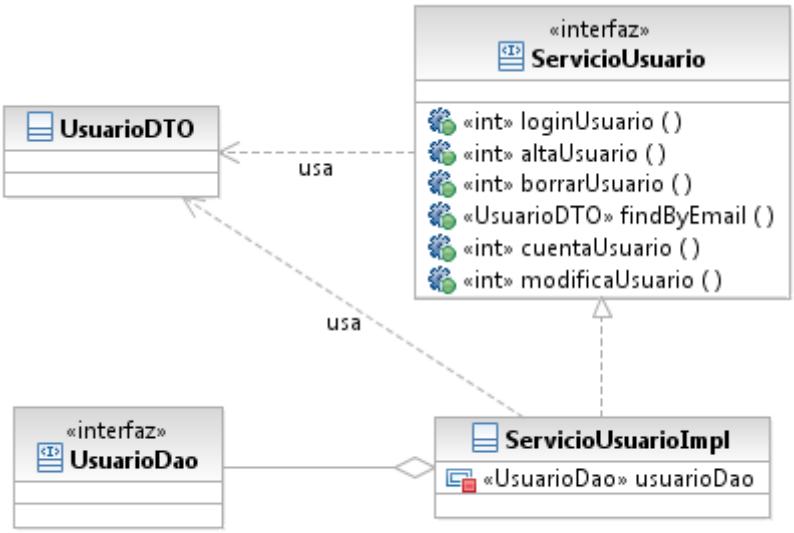


Figura 46. Servicio Usuario

La interfaz ServicioUsuario contiene los métodos incluidos en la lógica de negocio para la tabla usuario de la base de datos. La clase ServicioUsuarioImpl implementa esta interfaz, luego debe implementar todos estos métodos. Además esta clase contiene un atributo de tipo UsuarioDao para servirnos de todos los métodos de la capa de persistencia para acceder a la base de datos.

Vamos a ver cada método por separado y cómo se relacionan con la capa de persistencia:

- **findByEmail**: este método recibe un string email por parámetro y simplemente hace una llamada al método findByEmail de la capa de persistencia de esta forma:

```
Usuario resultadoBD = usuarioDao.findByEmail(email);
```

  Obtenemos el Usuario de la base de datos que contiene el email dado por el parámetro, pero este método debe devolver un UsuarioDTO, luego después de la llamada hacemos uso del método de conversión convertBDToDTO y devolveremos null si no hay ningún usuario con ese email, o el objeto de tipo UsuarioDTO en caso de que si exista.

- **loginUsuario**: este método contiene la funcionalidad a ejecutar cuando el usuario quiera realizar el login, es decir, cuando quiera acceder a la aplicación. Para ello este método recibe como parámetros dos String que son el email y la contraseña. Primero se comprueban que estos dos parámetros no sean null, y en caso afirmativo se devolverá el error correspondiente en forma de número entero.

  Si no son null, se procede a recuperar el usuario almacenado en la base de datos con ese email, llamando al mismo método de esta clase findByEmail de la forma:

```
UsuarioDTO usuarioRecuperado = this.findByEmail(email);
```

Si obtenemos null es que no existe ningún usuario en el sistema registrado con ese email, por lo que se devuelve el correspondiente error. Si es distinto de null entonces comparamos que la contraseña del usuario recuperado sea igual a la contraseña recibida por parámetro. Si son iguales entonces se devuelve 0 informando que el login se ha realizado correctamente. Y en otro caso se devuelve el error asociado a contraseña incorrecta.

- **altaUsuario**: este método contiene la funcionalidad para dar de alta a un usuario nuevo en el sistema. Recibe por parámetro un objeto del tipo UsuarioDTO con los datos que el usuario ha rellenado desde el cliente. Lo primero que se comprueba es que este objeto y todos sus atributos no sean null. La siguiente comprobación es ver si ya existe un usuario con ese mismo email en la base de datos usando el método findByEmail de esta misma clase:

```
UsuarioDTO usuarioExistente = this.findByEmail(entidad.getEmail());
```

donde entidad es el objeto de tipo UsuarioDTO que hemos recibido por parámetro. Si este método devuelve algún usuario, es decir, usuarioExistente es distinto de null, significa que ya existe un usuario con ese mismo email y devolverá el error correspondiente. En caso de que sea null no existe ningún usuario con ese email, luego podemos registrarlo en el sistema. Para añadirlo a la base de datos llamamos al método save mediante la variable *usuarioDao,* pero como éste recibe como parámetro un objeto de tipo Usuario (y no UsuarioDTO) tendremos que convertir primero el objeto que nos llegó del cliente, de la forma:

```
Usuario entidadBD = this.convertDTOToBD(entidad);
// realizamos el alta en BD
int resultado = usuarioDao.save(entidadBD);
```

- **borrarUsuario**: este método contiene la funcionalidad para borrar un usuario de la base de datos. Recibe por parámetros un objeto del tipo UsuarioDTO, y lo primero que comprobamos es que este objeto no sea null, y validamos que sus atributos tampoco lo sean. A continuación convertimos este objeto a tipo Usuario para poder llamar al método *delete* de la capa de persistencia mediante el atributo *usuarioDao*. Se devuelve un entero con el resultado de la operación.

```
Usuario entidadBD = this.convertDTOToBD(entidad);
int resultado = usuarioDao.delete(entidadBD);
```

- **cuentaUsuario**: este método cuenta los usuarios que hay registrados en la base de datos. Lo usa el cliente para darle un id a un nuevo usuario ya que si hay n usuarios en la base de datos, el nuevo usuario tendrá id n+1. La implementación de este método es simplemente llamar al método *count* de la capa de persistencia mediante el atributo *usuarioDao*. Se devuelve entonces el entero con el número de usuarios.

```
        int resultado = usuarioDao.count();
```

- **modificaUsuario**: este método contiene la funcionalidad para modificar un usuario, esto es, por si el cliente decide cambiar sus datos. Recibe por parámetros un objeto del tipo UsuarioDTO, y lo primero que comprobamos es que este objeto no sea null, y validamos que sus atributos tampoco lo sean. A continuación convertimos este objeto a tipo Usuario para poder llamar al método *modify* de la capa de persistencia mediante el atributo *usuarioDao*. Se devuelve un entero con el resultado de la operación.

```
        Usuario entidadBD = this.convertDTOToBD(entidad);
        int resultado = usuarioDao.modify(entidadBD);
```

❒ Clases ServicioParque.java y ServicioParqueImpl.java

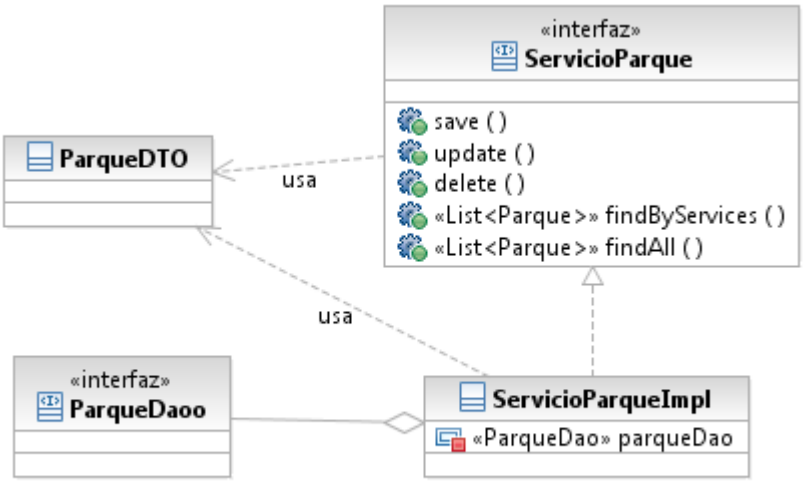


Figura 47. Servicio Parque

La interfaz *ServicioParque* contiene los métodos incluidos en la lógica de negocio para la tabla parque de la base de datos. La clase *ServicioParqueImpl* implementa esta interfaz y todos sus métodos. Además esta clase contiene un atributo de tipo *ParqueDao* para servirnos de todos los métodos de la capa de persistencia para acceder a la base de datos.

Vamos a ver cada método por separado y cómo se relacionan con la capa de persistencia:

- **save:** este método se usa para guardar un nuevo parque en la base de datos. Recibe por parámetro un objeto de tipo *ParqueDTO*. Primero se comprueba que este objeto no sea null, ni tampoco sus atributos internos. A continuación convertimos este objeto a un objeto del tipo de la base de datos Parque, y ya podemos llamar al método *save* de la capa de persistencia mediante el atributo *parqueDao*. Este método se usará sólo para la carga de información en la tabla de la base de datos.

```
        Parque entidadBD = this.convertDTOToBD(entidad);
        parqueDao.save(entidadBD);
```

- **update:** este método se usa para actualizar información sobre un parque de la base de datos. Recibe por parámetro un objeto de tipo *ParqueDTO*. Primero se comprueba que este objeto no sea null, ni tampoco sus atributos internos. A continuación convertimos este objeto a un objeto del tipo de la base de datos Parque, y ya podemos llamar al método *update* de la capa de persistencia mediante el atributo parqueDao. Este método se usará sólo para la actualización de información en la tabla de la base de datos.

```
	Parque entidadBD = this.convertDTOToBD(entidad);
	parqueDao.update(entidadBD);
```

- **delete:** este método se usa para borrar un parque de la base de datos. Recibe por parámetro un objeto de tipo *ParqueDTO*. Convertimos este objeto a uno del tipo de la base de datos Parque, y ya podemos llamar al método *delete* de la capa de persistencia mediante el atributo *parqueDao*. Este método se usará sólo para el borrado de información en la tabla de la base de datos.

```
	Parque entidadBD = this.convertDTOToBD(entidad);
	parqueDao.delete(entidadBD);
```

- **findByServices**: este método se usa para obtener de la base de datos aquellos parques que contienen una serie de servicios. Recibe 8 variables de tipo short, cada una para indicar cada uno de los servicios que puede contener un parque. Estos 8 servicios son: *zonaInfantil, zonaMayores, zonaDeportiva, carrilBici, pipican, circuitoBiosaludable, puntoInformación* y *aseos.* Estas variables valen 1 si queremos que contenga ese servicio, y 0 si nos da igual si el parque contiene ese servicio o no. Con los servicios que valgan 1 construimos un string *servicios* de la forma "servicio1=1 and servicio2=1" que será el parámetro que usemos para llamar al método *findByServices* de la capa de persistencia que es el que accede a la base de datos. Si todos los servicios recibidos valiesen 0, es decir, el string creado servicios no contiene nada, entonces la llamada será con el método findAll ya que quiere decir que no queremos realizar ningún filtrado.

```
	List<Parque> listaResultado;
	if(servicios.length()==0){
		//quiero todos los parques sin filtrar
		listaResultado = parqueDao.findAll();
	}else{
		listaResultado =parqueDao.findByServices(servicios);
}
```

Devolveremos la lista de objetos *Parque* que hemos obtenido de la base de datos. En estos casos devolvemos objetos de tipo *Parque* y no de tipo *ParqueDTO* ya que la capa de persistencia nos devuelve una lista de objetos Parque. Si quisiésemos transformarlo a una lista de objetos *ParqueDTO* tendríamos que recorrer la lista y convertir cada objeto, lo que supondría bastante complejidad.

- **findAll:** este método se usa para obtener todos los parques de la base de datos. Se llama al método *findAll* de la capa de persistencia que es el que accede a la base de datos. Igual que antes devolvemos tal cual la lista de objetos Parque obtenida de la capa de persistencia para disminuir la complejidad.

```
List<Parque> listaResultado = parqueDao.findAll();
```

❒ **Clases ServicioParkCoche.java y ServicioParkCocheImpl.java,**
Clases ServicioParkMoto.java y ServicioParkMotoImpl.java,
Clases ServicioParkBici.java y ServicioParkBiciImpl.java,
Clases ServicioPtoLimpioFijo.java y ServicioPtoLimpioFijoImpl.java,
Clases ServicioPtoLimpioMovil.java y ServicioPtoLimpioMovilImpl.java, y
Clases ServicioContRopa.java y ServicioContRopaImpl.java

El grupo de estos 6 recursos se comporta igual. La interfaz *Servicio"Recurso"* contiene los métodos incluidos en la lógica de negocio para cada tabla de estos recursos de la base de datos. La clase *Servicio"Recurso"Impl* implementa esta interfaz, y debe implementar todos estos métodos. Además esta clase contiene un atributo de tipo *"Recurso"Dao* para servirnos de todos los métodos de la capa de persistencia para acceder a la base de datos.

Los 6 recursos usan 4 métodos cuya funcionalidad es idéntica. Un método **save** para almacenar un nuevo recurso en la base de datos, que llamará al método save de la capa de persistencia. Un método **update** para actualizar un recurso en la base de datos, que llamará al método update de la capa de persistencia. Un método **delete** para borrar un recurso en la base de datos, que llamará al método delete de la capa de persistencia. Y por último un método **findAll** para obtener todos los recursos de la tabla de la base de datos, que devolverá una lista con objetos de tipo del recurso específico. Como antes será una lista de objetos del tipo del recurso de la capa de persistencia (es decir, no DTO), para ahorrarnos complejidad a la hora de tener que convertir todos los objetos de la lista.

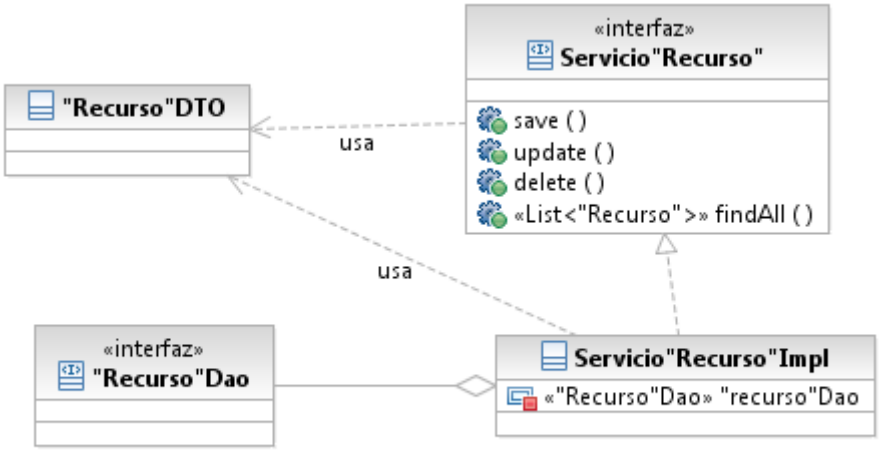


Figura 48. Servicio "Recurso"

❒ **Clases ServicioPtoSuministro.java y ServicioPtoSuministroImpl.java**

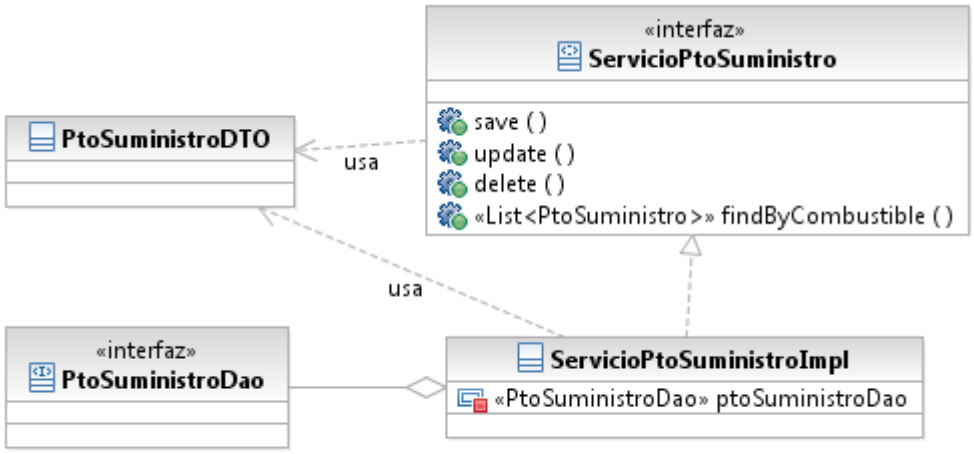


Figura 49. Servicio PtoSuministro

La interfaz *ServicioPtoSuministro* contiene los métodos incluidos en la lógica de negocio para la tabla *ptossuministros* de la base de datos. La clase *ServicioPtoSuministroImpl* implementa esta interfaz, e implementa todos estos métodos. Además esta clase contiene un atributo de tipo *PtoSuministroDao* para servirnos de todos los métodos de la capa de persistencia para acceder a la base de datos.

Los métodos usados son: un método **save** para almacenar un nuevo *PtoSuministro* en la base de datos, que llamará al método save de la capa de persistencia; un método **update** para actualizar un *PtoSuministro* en la base de datos, que llamará al método update de la capa de persistencia, y un método **delete** para borrar un *PtoSuministro* en la base de datos, que llamará al método delete de la capa de persistencia. Por último un método **findByCombustible** para obtener todos los puntos de suministro de la tabla de la base de datos cuyo combustible sea el combustible pasado por parámetro en forma de string. Este método devolverá una lista con objetos de tipo *PtoSuministro*. Como antes será una lista de objetos del tipo de la capa de persistencia (es decir, no DTO), para ahorrarnos complejidad a la hora de tener que convertir todos los objetos de la lista.

5. Paquetes *utilidades.constantes* y *utilidades.enumerados*

Dentro del paquete *utilidades.constantes* encontramos una clase llamada *Constantes.java* donde incluimos 3 constantes que se usan para devolver un número entero en caso de que en la acción realizada se haya obtenido una ejecución correcta, una validación correcta, o un login correcto. El código de estas tres constantes es el siguiente:

```
public static final int EJECUCION_CORRECTA = 0;
public static final int VALIDACION_CORRECTA = 0;
public static final int LOGIN_CORRECTO = 0;
```

De esta forma si queremos cambiar el número entero que se usa para alguna de estas tres situaciones, bastará con cambiarlo en esta clase *Constantes.java*.

Dentro del paquete *utilidades.enumerados* encontramos una clase llamada *ErroresUsuario.java* donde incluimos un enumerado llamado ErroresUsuario. Este enumerado contiene varios posibles errores que pueden darse al ejecutar varias de las acciones comentadas anteriormente, para de esta forma poder capturarlos y saber qué error especifico ha pasado. Cada tipo de error contiene un número entero propio y una descripción del error.

El código de este enumerado es el siguiente:

```
public enum ErroresUsuario {
        ErrorEntidadNula (-1, "La entidad es nula"),
        ErrorEjecucionDAO (-2, "La invocación al DAO ha devuelto un error"),

        ErrorAñoNacNulo (-11, "El año de nacimiento es nulo"),
        ErrorPasswordNulo (-12, "La contraseña es nula"),
        ErrorEmailNulo (-13, "El email del usuario es nulo"),
        ErrorSexoNulo (-14, "El sexo es nulo"),

        ErrorEmailEnUso (-21, "El email ya existe"),
        ErrorEmailNoExiste (-22, "No hay ninguna cuenta asociada a ese
                                                        email"),
        ErrorPasswordIncorrecta (-23, "La contraseña es incorrecta");
```

Igual que antes, si quisiésemos modificar el número entero que se devuelve en cada caso, bastaría con modificarlo dentro de este enumerado.

- ✓ **Directorio src/test/java**

Una vez llegados a este punto del proyecto teníamos que comprobar de alguna manera que la capa de persistencia y la capa de negocio funcionaban perfectamente, y que la conexión con la base de datos se realizaba con normalidad. Además necesitábamos almacenar en la base de datos toda la información proporcionada por el Ayuntamiento de Madrid sobre cada recurso, ayudándonos de todos los métodos incluidos en las capas anteriores.

Por ello creamos dos paquetes dentro de este directorio src/test/java, cuyos nombres son com.maparecursos.test y com.maparecursos.carga

1. **Paquete *com.maparecursos.test***

Una vez terminadas la capa de persistencia y de negocio, debemos asegurarnos que funcionan bien para poder continuar con la capa de integración y la capa de presentación, ya que si no corremos el riesgo de que falle más adelante y sea mucho más complejo depurar el error.

jUnit [66] es un framework java que permite la realización de la ejecución de clases de manera controlada, para poder comprobar que los métodos realizan su cometido de forma correcta. Su integración con Eclipse es muy sencilla pero somos nosotros los que debemos lanzar los test manualmente. Sin embargo los proyectos de Maven poseen una estructura con un directorio llamado src/test/java, donde Maven detecta automáticamente todos los test que dejemos en

dicha ruta y los ejecutará cada vez que le solicitemos una compilación. Gracias a esta funcionalidad podemos detectar los fallos que pudiese haber.

Para compilar un proyecto Maven desde Eclipse hacemos click en el proyecto con el botón derecho y elegimos la opción Run As y a continuación Maven build. Aparece un cuadro para editar la configuración. Hay muchos tipos de configuraciones pero a nosotros sólo nos interesa una. Dentro de goals escribimos "clean install" para que nos elimine las posibles compilaciones anteriores y lo compile nuevamente, y de esta forma además ejecutará los test que tengamos almacenados en el directorio src/test/java.

Tendremos una clase test por cada tabla de la base de datos para comprobar que los servicios ofrecidos para cada una funcionan correctamente. Cada clase test deberá extender la clase TestCase, y en cada clase habrá un método que de nombre comienza por "test" donde se realizarán las comprobaciones necesarias, y además debe ser void, es decir, no devuelve nada. Estas clases deben contener un método suite() propio de jUnit para la creación del test.

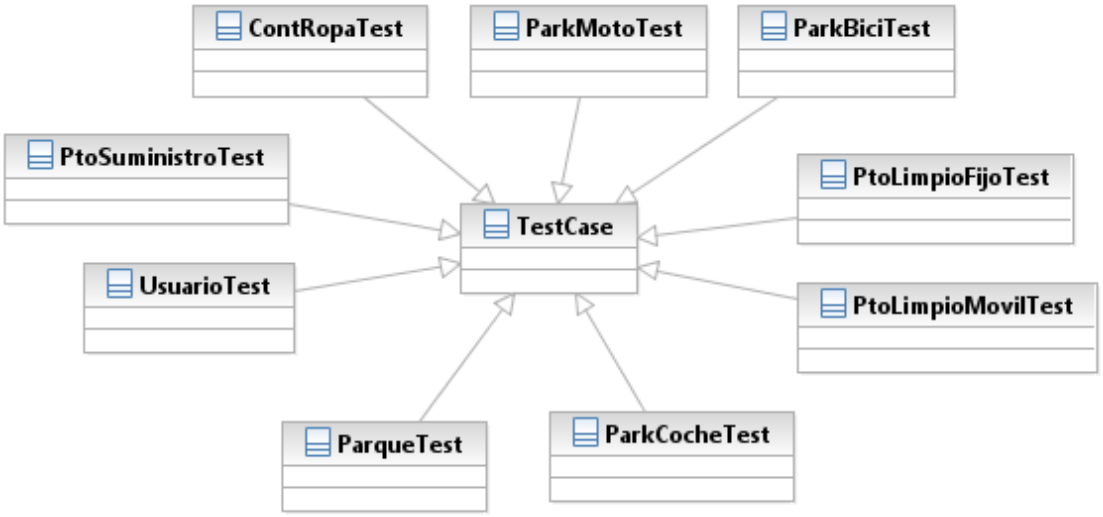


Figura 50. Test Servicios

Dentro de cada clase test obtenemos una instancia de *ApplicationContext* y le indicamos la ubicación del archivo que contiene la configuración de Spring; en nuestro caso ese archivo está dentro de spring/config/ y el nombre del archivo es BeanLocations.xml. Recordamos que este archivo xml no contiene la declaración de los bean como tal sino que contiene los import de otros archivos xml donde están almacenados los bean, agrupándolos según la tabla de la base de datos a la que se accederá.

```
ApplicationContext appContext =
      new ClassPathXmlApplicationContext("spring/config/BeanLocations.xml");
```

Así si quisiésemos crear un bean asociado a la clase *ServicioParqueImpl* creamos un nuevo objeto de esta clase, y mediante el objeto anterior *appContext* obtenemos el bean correspondiente a ese servicio mediante el método *getBean().*

```
ServicioParqueImpl servicioParque =
      (ServicioParqueImpl)appContext.getBean("servicioParque");
```

Con este nuevo objeto *servicioParque* disponemos de todos los métodos que incluye, como son *save*(), *update*(), *delete*(), *findAll*() o *findByServices*().

Por ejemplo, un test para ver cómo registrar un usuario en el sistema sería el siguiente código. Obtenemos una instancia de *ApplicationContext* y a continuación obtenemos el bean con el *id servicioUsuario*, que es el que nos interesa.

```
ApplicationContext appContext =
      new ClassPathXmlApplicationContext("spring/config/BeanLocations.xml");

ServicioUsuarioImpl servicioUsuario =
      (ServicioUsuarioImpl)appContext.getBean("servicioUsuario");

// insert
UsuarioDTO user = new UsuarioDTO();
user.setEmail("miemail@email.com");
user.setIdUsuario(18L);
user.setPassword("pwd");
user.setAñonacimiento(1990);
user.setSexo("Hombre");

int resultado = servicioUsuario.altaUsuario(user);
System.out.println(resultado);
```

Una vez creados los test, compilamos el proyecto de la forma especificada arriba y por consola obtendremos los resultados. El resultado de la ejecución de los 9 test tiene el siguiente aspecto, indicándonos que tanto los test como la compilación se han realizado con éxito.

```
Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 27.409s
[INFO] Finished at: Thu May 30 18:49:58 CEST 2013
[INFO] Final Memory: 5M/76M
[INFO] ------------------------------------------------------------------------
```

En particular, el test de inserción de un nuevo usuario de prueba mostraría:

```
Running com.maparecursos.tests.UsuarioTest
03-jun-2013 11:30:01 org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3c6e4ba6: displ
03-jun-2013 11:30:01 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefiniti
Hibernate: insert into maparecursos.usuario (password, email, sexo, añonacimiento, idUsuario) values (?, ?, ?, ?, ?)
0
```

Vemos que devuelve un 0 como resultado de la operación alta usuario lo que significa que la operación se ha realizado correctamente

2. Paquete *com.maparecursos.carga*

Dentro de este paquete se incluyen todas las clases usadas para la carga de los datos en la base de datos. Habrá 9 clases para la carga de información asociada a los siguientes recursos: parques, puntos limpios fijos, puntos limpios móviles, contenedores de ropa, aparcamientos de coche, aparcamientos de moto, aparcamientos de bici, puntos de suministro de gnc, glp y bioetanol, y por último puntos de recarga eléctricos.

Toda la información proporcionada por el Ayuntamiento la clasificamos por recursos, y dentro de ellos por los campos incluidos en cada tabla de la base de datos. Con ello creamos varios documentos de texto que incluyen esta información. De esta forma era fácil leer estos archivos .txt desde Eclipse. Estos archivos están almacenados en una carpeta de nombre infoCargaBD situada en el directorio raíz del proyecto. Dentro de ella se encuentra la información de cada recurso organizada en distintas carpetas según el tipo de recurso asociado en cada caso.

Para poder abrir un fichero de texto para leerlo desde java usamos la clase *FileReader* de java. Esta clase tiene métodos que permiten leer caracteres pero nosotros queríamos leer líneas enteras. La clase *BufferedReader* nos permite hacer esta lectura. De esta forma, podemos construir un *BufferedReader* a partir de un *FileReader*. Al objeto FileReader debemos pasarle la ruta del archivo a leer. A modo de ejemplo sería:

```
BufferedReader br =
      new BufferedReader(
            new FileReader("infoCargaBD/Recurso/miArchivo.txt"));
```

La apertura del fichero y su posterior lectura pueden lanzar excepciones que debemos capturar. Por ello la apertura del fichero y la lectura debemos meterla en un bloque *try-catch*.

Una vez creados estos objetos podemos leer líneas del archivo de texto gracias al método readLine(). Luego con un bucle while se puede ir leyendo el archivo hasta el final. Una vez leído el archivo debemos cerrar el archivo de texto con el método close().

```
String linea = null;
while((linea=br.readLine())!=null){
      //tratar la linea leida
}
br.close();
```

Así vamos leyendo toda la información asociada a cada recurso, y vamos creando un objeto de tipo DTO específico en cada caso para insertar un nuevo registro en la tabla correspondiente. Gracias a sus métodos *set()* vamos rellenando sus atributos, que serán los campos de la tabla.

Para poder insertar cada objeto DTO necesitamos un servicio que permita llamar al método save() específico en cada caso, para que guarde cada objeto como un nuevo registro en la base de datos. Para ello realizamos la misma metodología que usamos en los test. Obtenemos una instancia de *ApplicationContext* y le indicamos la ubicación del archivo que contiene la

configuración de Spring. A continuación creamos una instancia del servicio que queramos usar, y obtenemos el bean asociado al id de este servicio mediante el método *getBean* que proporcina *ApplicationContext*. A partir de este objeto ya podremos acceder a los métodos de ese servicio y hacer uso de ellos.

Vamos a ver un ejemplo para insertar un nuevo registro en la tabla asociada a los contenedores de ropa:

```
ApplicationContext appContext =
       new ClassPathXmlApplicationContext("spring/config/BeanLocations.xml");

ServicioContRopaImpl servicioContRopa =
       (ServicioContRopaImpl)appContext.getBean("servicioContRopa");

ContRopaDTO objeto = new ContRopaDTO();

p.setIdCont(id);
p.setDireccion(direccion);
p.setDistrito(distrito);
p.setLatitud(lat);
p.setLongitud(lon);
p.setNdpCont(ndp);

servicioContRopa.save(p);
```

Los atributos de id, dirección, distrito, lat, lon y ndp se refieren a la información leída en cada caso de los archivos txt correspondientes.

Hasta aquí tenemos la explicación de la implementación de las capas de persistencia y de negocio del proyecto. Continuamos con la explicación de la capa de integración por el lado del servidor.

❖ Mapa Recursos WS

Este proyecto se encarga de la capa de integración en el lado del servidor para exponer los servicios como web services al cliente.

• Dependencias con librerías

Para este proyecto también son necesarias varias librerías. De nuevo gracias al framework Maven basta con especificar las librerías que necesitamos en el archivo pom.xml, y el solo se encarga de descargar estas dependencias de su repositorio. Este archivo se guarda en el directorio raíz del proyecto.

Primero dentro de este archivo pom.xml debemos indicar cómo queremos plegar este proyecto.

```
<project                              xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.maparecursos.webservice</groupId>
    <artifactId>MapaRecursosWS</artifactId>
    <packaging>war</packaging>
    <version>0.1</version>
    <name>MapaRecursosWS</name>
      <!—Espacio para todas las dependencias -->
</project>
```

Gracias a este trozo de código el proyecto MapaRecursosWS se plegará en un archivo .war. Además debemos incluir una dependencia con la librería que se creó del proyecto anterior MapaRecursosServer de la siguiente forma:

```
<dependencies>
      <dependency>
                    <groupId>com.maparecursos</groupId>
                    <artifactId>MapaRecursosServer</artifactId>
                    <version>1.0</version>
      </dependency>
      <!—Espacio para las demás dependencias -->
</dependencies>
```

De forma que esta librería se añada a nuestro nuevo proyecto MapaRecursosWS. El archivo plegado .war que obtendremos al final es que usaremos para plegarlo en la plataforma CloudBees como se especificó anteriormente.

Las librerías necesarias en este proyecto MapaRecursosWS son:

Nombre de la librería y versión	Descripción
antlr-2.7.7	ANTLR: ANother Tool for Language Recognition (Otra herramienta para el Reconocimiento del Idioma). Proporciona un framework para construir reconocedores, compiladores, y traductores de las descripciones gramaticales que contienen Java, C + +, o C#. Librería necesaria para Hibernate.
asm-attrs-1.5.3	Librería de Hibernate.
axis2-kernel-1.4.1, annogen-0.1.0, axiom-api-1.2.7, axiom-dom-1.2.7, axiom-impl-1.2.7, axis2-codegen-1.4.1, axis2-adb-1.4.1, axis2-spring-1.4.1	Librerías de Axis2.
cglib-2.2	CGLIB es una librería de generación de código de alto rendimiento y calidad. Es usada para extender clases Java e implementa interfaces en tiempo de ejecución. Es necesaria para Hibernate.
commons-collections-3.2.1	Librería de Apache que extiende o aumenta la librería de Colecciones de Java.
commons-httpclient-3.1	Librería de Apache para peticiones http.
commons-logging-1.1.1	Librería de Apache para la generación de logs.
dom4j-1.6.1	Librería que ofrece soporte para API's de procesamiento de XML.
ehcache-1.2.3	Librería con estándares para la caché de java. Usado en proyectos con Hibernate y Spring.
hibernate-3.2.1.ga	Librería de Hibernate.
jaxen-1.1.1	Librería para compilar JavaBean como xml.
jdk-1.6	Librería java para el funcionamiento de Spring.
jta-1.0.1B	Librería de la API Java Transaction para resolver las dependencias de Maven.
junit-4.7	Librería para realizar test con JUnit.
log4j-1.2.14, slf4j-api-1.5.8	Librerías para logs
mysql-connector-java-5.1.9	Librería del conector de la base de datos MySQL usada.
neethi-2.0.4	Librería para configuración wsdl.
persistance-api-1.0	Librería para la persistencia de objetos java.
servlet-api-2.3	Librería de Servlets.
spring-2.5.6, spring-beans-3.0.3, spring-core-3.0.3, spring-context-3.0.3, spring-aop-3.0.3, spring-expression-3.0.3, spring-asm-3.0.3, spring-web-2.5.1	Librerías de Spring.
wooden-api-1.0M8, wooden-impl-dom-1.0M8	Librerías para crear y escribir wsdl.
wsdl4j-1.6.2	Librería para Web Services.
wstx-asl-3.2.4	Librería Apache para parseo de xml.
xalan-2.7.0	Librería para transformar xml en html.
xml-apis-1.0.b2	Contiene las clases java que necesita JDOM para dar formato

	al XML.
xmlschema-1.4.2, xmlParserAPIs-2.0.6, xercesImpl-2.8.1	Librerías para esquemas y parseos xml.

- **Organización del proyecto**

El proyecto contiene 3 directorios diferenciados para contener clases y archivos necesarios. Estos directorios son:

- *src/main/java*: contiene las clases necesarias para indicar los servicios que se deben ofrecer como web service en forma de interfaces y clases java que las implementan.
- *src/main/resources*: contiene dos archivos de configuración con los beans a instanciar por parte de Spring.
- *src/main/webapp*: contiene el archivo de configuración web.xml y los archivos para la configuración de los web service y sus operaciones.

✓ Directorio src/main/java

Dentro de este directorio se incluyen 2 paquetes, *webservice* y *webservice.impl*:

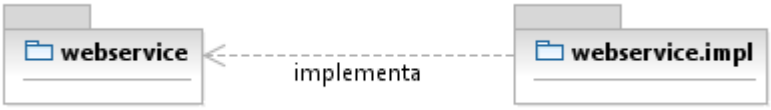


Figura 51. Paquetes del directorio src/main/java

- **Paquetes *com.maparecursos.webservice* y *com.maparecursos.webservice.impl***

El paquete webservice incluye las interfaces donde definimos los métodos con las operaciones que queremos exponer en los servicios. Tendremos una interface por cada tabla de la base de datos, como siempre, para tener las operaciones agrupadas.

El paquete webservice.impl incluye las clases que implementan estas interfaces, donde se dará la funcionalidad necesaria para que las operaciones que queremos exponer en los servicios llamen a los servicios que a su vez creamos dentro de la capa de negocio, dentro del proyecto MapaRecursosServer. Tendremos de nuevo una clase por cada tabla de la base de datos para tener las operaciones agrupadas.

Vamos a ver clase a clase dentro de los paquetes com.maparecursos.webservice y com.maparecursos.webservice.impl.

❒ **Clase UsuarioWebService.java y UsuarioWebServiceImpl.java**

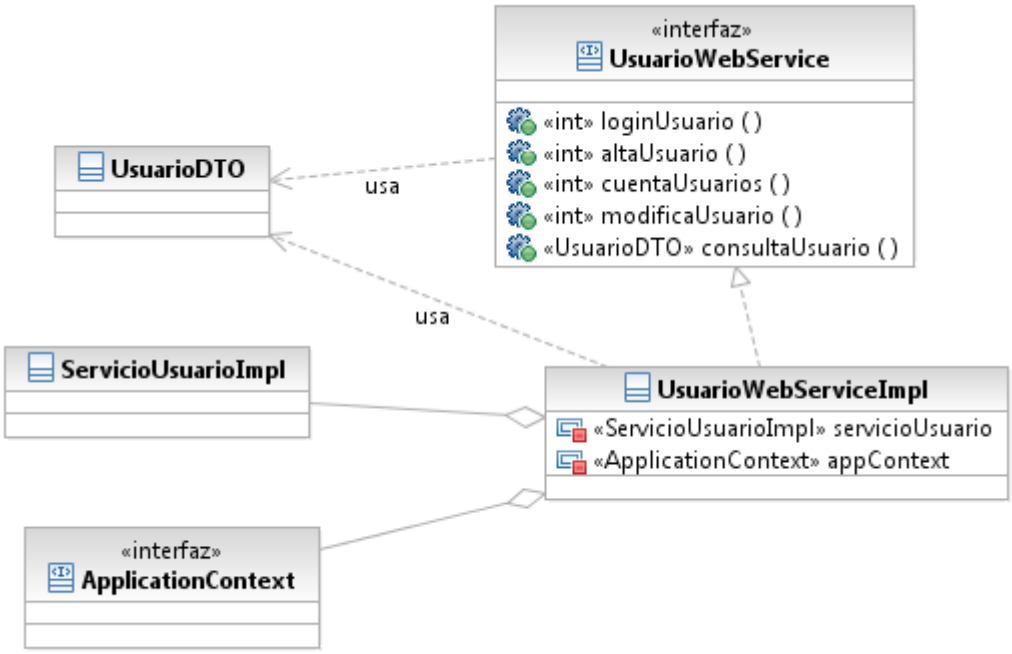


Figura 52. Web Service Usuario

La interfaz *UsuarioWebService* contiene los métodos a exponer como webservice al cliente que corresponden con usuarios. La clase *UsuarioWebServiceImpl* implementa esta interfaz, luego debe implementar todos estos métodos. Además esta clase contiene dos atributos, uno de tipo *ApplicationContext* y otro de tipo *ServicioUsuarioImpl*. Estos dos atributos se usan de forma similar a los test y la carga de datos usados en el proyecto MapaRecursosServer.

Creamos una instancia de tipo *ApplicationContext*, indicándole la ubicación del archivo de configuración de Spring llamado BeanLocations.xml contenido en el proyecto anterior. Dentro de él están contenidos los beans a crear por parte de Spring. Por ello cuando creamos el objeto de tipo *ServicioUsuarioImpl* obtenemos la instancia mediante el método *getBean*, que Spring ha creado mediante un bean.

```
ApplicationContext appContext =
      new ClassPathXmlApplicationContext("spring/config/BeanLocations.xml");

ServicioUsuarioImpl servicioUsuario =
     (ServicioUsuarioImpl)appContext.getBean("servicioUsuario");
```

De esta forma mediante la instancia servicioUsuario podemos llamar a todos los métodos de la capa de negocio implementados en la clase ServicioUsuarioImpl, llamados igual que los métodos de la interfaz UsuarioWebService.

Los métodos a los que el cliente podrá conectarse son los siguientes, y dentro simplemente tendrán la llamada a cada método correspondiente de la forma servicioUsuario.método():

- loginUsuario: para mirar si un usuario existe en el sistema. Recibe como parámetros el email y contraseña del usuario.

```
int resultado = servicioUsuario.loginUsuario(email, password);
```

- altaUsuario: para ingresar un usuario nuevo. Recibe como parámetro un objeto UsuatioDTO con los datos.

```
int resultado = servicioUsuario.altaUsuario(usuario);
```

- cuentaUsuarios: para contar cuantos usuarios hay en la base de datos dados de alta.

```
int resultado = servicioUsuario.cuentaUsuarios();
```

- modificaUsuarios: para modificar los datos de un usuario recibido por parámetro.

```
int resultado = servicioUsuario.modificaUsuario(usuario);
```

- consultaUsuario: para consultar el resto de los datos de un usuario dado un email.

```
UsuarioDTO userDTO = servicioUsuario.findByEmail(email);
```

❒ **Clase ParqueWebService.java y ParqueWebServiceImpl.java**

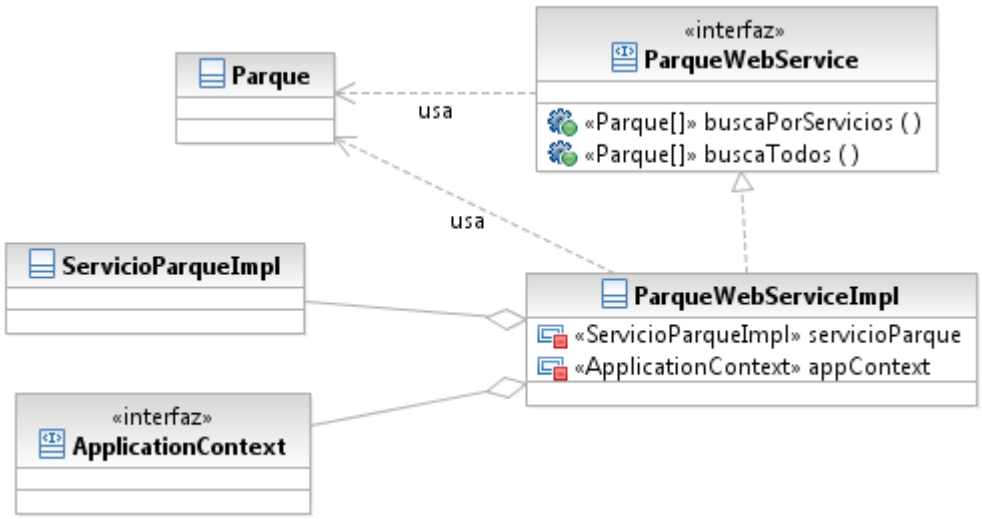


Figura 53. Web Service Parque

La interfaz *ParqueWebService* contiene los métodos a exponer como webservice al cliente que corresponde con parques. La clase *ParqueWebServiceImpl* implementa esta interfaz y debe implementar todos estos métodos. Además esta clase contiene dos atributos, uno de tipo *ApplicationContext*, y otro de tipo *ServicioParqueImpl*. Estos dos atributos se usan de forma similar a como se ha explicado en la clase anterior.

```
ApplicationContext appContext =
      new ClassPathXmlApplicationContext("spring/config/BeanLocations.xml");

ServicioParqueImpl servicioParque =
      (ServicioParqueImpl)appContext.getBean("servicioParque");
```

Mediante la instancia servicioParque podemos llamar a todos los métodos de la capa de negocio implementados en la clase ServicioParqueImpl. Los métodos de ServicioParqueImpl para añadir, actualizar o borrar un parque no los vamos a exponer al cliente ya que no queremos que el cliente posea esos privilegios. En este caso los métodos de ServicioParqueImpl devolverán una lista de objetos de tipo Parque. Ya explicamos el uso de Parque y no ParqueDTO para evitarnos complejidad a la hora de realizar la conversión de toda la lista dentro de la capa de negocio. Pero los web services no pueden devolver instancias de tipo List sino que deben ser arrays de objetos, en este caso de tipo Parque. Por ello tendremos que llamar al método de ServicioParqueImpl y, cuando obtengamos la lista, realizar la conversión a un array para poder ofrecerlo al cliente en el webservice.

```
Parque[] array=null;
      if(resultado!=null){
            array=new Parque[resultado.size()];
            Iterator<Parque> iter = resultado.iterator();
            int i=0;
            while (iter.hasNext()){
                  Parque p = (Parque) iter.next();
                  array[i]=p;
                  i++;
            }
      }
return array;
```

Los métodos a los que el cliente podrá conectarse son los siguientes:

- buscaPorServicios: para hacer la búsqueda de los parques según un filtrado de los servicios que contiene. Los servicios a filtrar se obtienen por parámetro.

```
List<Parque> resultado = servicioParque.findByServices(zonaInfantil,
             zonaMayores, zonaDeportiva, carrilBici, circuitoBiosaludable,
             pipican, puntoInformacion, aseoPublico);
```

- buscaTodos: para devolver todos los parques.

```
List<Parque> resultado = servicioParque.findAll();
```

- **Clase ParkCocheWebService.java y ParkCocheWebServiceImpl.java,**
  Clase ParkMotoWebService.java y ParkMotoWebServiceImpl.java,
  Clase ParkBiciWebService.java y ParkBiciWebServiceImpl.java,
  Clase PtoLimpioFijoWebService.java y
  PtoLimpioFijoWebServiceImpl.java,
  Clase PtoLimpioMovilWebService.java y
  PtoLimpioMovilWebServiceImpl.java,
  Clase ContRopaWebService.java y ContRopaWebServiceImpl.java

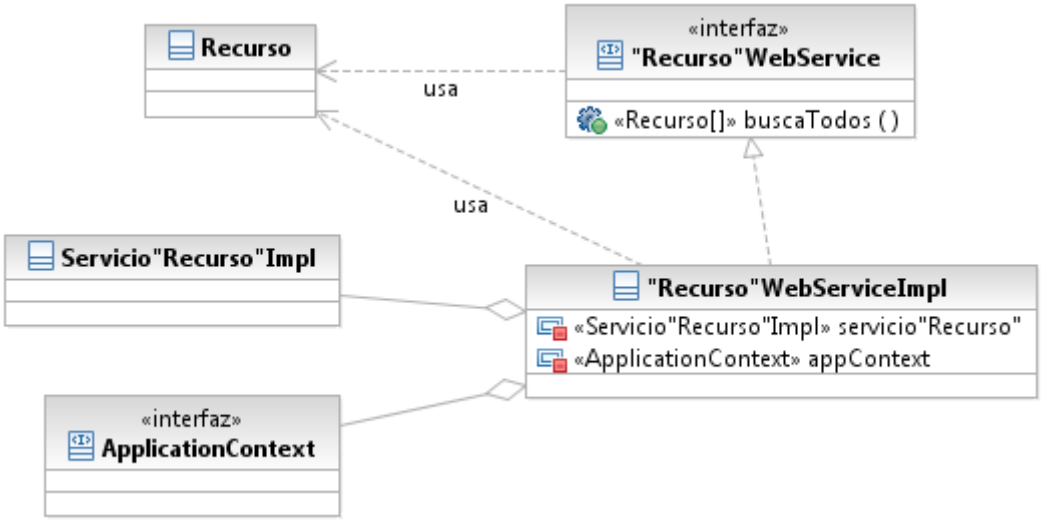


Figura 54. Web Service "Recurso"

El grupo de estos 6 recursos se comportan igual. La interfaz *"Recurso"WebService* contiene los métodos a exponer como webservice al cliente que corresponde con los respectivos a cada recurso. La clase *"Recurso"WebServiceImpl* implementa esta interfaz y todos los métodos correspondientes. Además esta clase contiene dos atributos, uno de tipo *ApplicationContext* y otro de tipo *Servicio"Recurso"Impl*. Estos dos atributos se usan de forma similar a como se ha explicado en la clase anterior.

```
ApplicationContext appContext =
      new ClassPathXmlApplicationContext("spring/config/BeanLocations.xml");

Servicio"Recurso"Impl servicio"Recurso" =
      (Servicio"Recurso"Impl)appContext.getBean("servicioRecurso");
```

Mediante la instancia servicio*"Recurso"* podemos llamar a todos los métodos de la capa de negocio implementados en la clase Servicio*"Recurso"*Impl. Los métodos de Servicio*"Recurso"*Impl para añadir, actualizar o borrar un recurso no los vamos a exponer al cliente ya que no queremos que el cliente posea esos privilegios. En este caso los métodos de Servicio*"Recurso"*Impl devolverán una lista de objetos de tipo *"Recurso"*. Ya explicamos el uso de *"Recurso"*y no *"Recurso"*DTO para evitarnos complejidad a la hora de realizar la conversión de toda la lista dentro de la capa de negocio. Pero los web services no pueden devolver instancias de tipo List, sino que deben ser arrays de objetos, en este caso de tipo *"Recurso"*.

Por ello tendremos que llamar al método de Servicio*"Recurso"*Impl y, cuando obtengamos la lista, realizar la conversión a un array para poder ofrecerlo al cliente en el webservice, en la forma que se ha expuesto en la clase anterior.

Los métodos a los que el cliente podrá conectarse son los siguientes:

- buscaTodos: para devolver todos los recursos correspondientes en cada caso.

```
List<"Recurso"> resultado = servicio"Recurso".findAll();
```

❒ **Clase PtoSuministroWebService.java y PtoSuministroWebServiceImpl.java,**

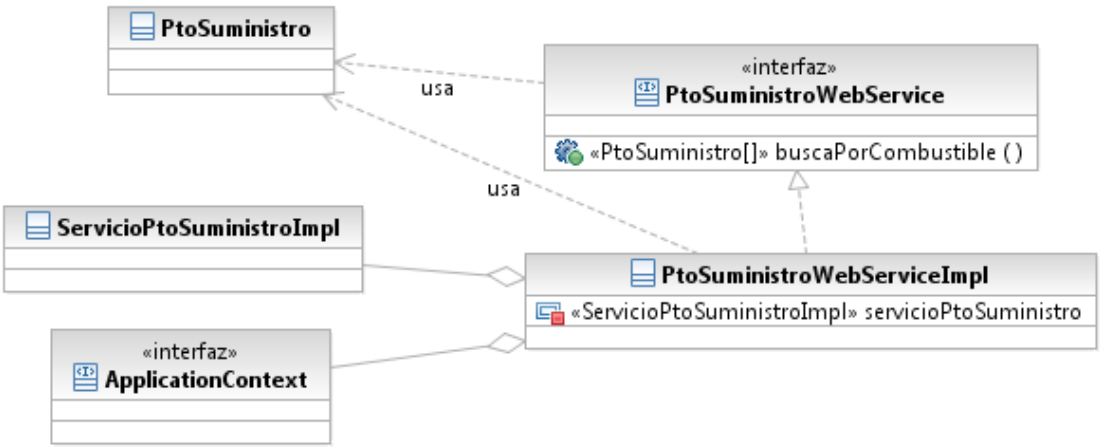


Figura 55. Web Service PtoSuministro

La interfaz *PtoSuministroWebService* contiene los métodos a exponer como webservice al cliente que corresponde con parques. La clase *PtoSuministroWebServiceImpl* implementa esta interfaz y todos estos métodos. Además esta clase contiene dos atributos, uno de tipo *ApplicationContext* y otro de tipo *ServicioPtoSuministroImpl*. Estos dos atributos se usan de forma similar a como se ha explicado en la clase anterior.

```
ApplicationContext appContext =
      new ClassPathXmlApplicationContext("spring/config/BeanLocations.xml");

ServicioPtoSuministroImpl servicioParque =
    (ServicioPtoSuministroImpl)appContext.getBean("servicioPtoSuministro");
```

Mediante la instancia *servicioPtoSuministro* podemos llamar a todos los métodos de la capa de negocio implementados en la clase *ServicioPtoSuministroImpl*. Los métodos de *ServicioPtoSuministroImpl* para añadir, actualizar o borrar un parque no los vamos a exponer al cliente ya que no queremos que el cliente posea esos privilegios. En este caso los métodos de *ServicioPtoSuministroImpl* devolverán una lista de objetos de tipo *PtoSuministro*. Ya explicamos el uso de *PtoSuministro* y no *PtoSuministroDTO* para evitarnos complejidad a la hora de realizar la conversión de toda la lista dentro de la capa de negocio. Pero los web

services no pueden devolver instancias de tipo List, sino que deben ser arrays de objetos, en este caso de tipo *PtoSuministro*. Por ello tendremos que llamar al método de Servicio*PtoSuministro*Impl y, cuando obtengamos la lista, realizar la conversión a un array para poder ofrecerlo al cliente en el webservice en la forma que se ha expuesto antes.

Los métodos a los que el cliente podrá conectarse son los siguientes:

- buscaPorCombustible: para hacer la búsqueda de los puntos de suministro según el tipo de combustible que el usuario haya elegido.

```
List<PtoSuministro> resultado =
            servicioPtoSuministro.findByCombustible(combustible);
```

✓ Directorio src/main/resources

Dentro de este directorio se incluyen 2 archivos:

- **applicationContext.xml**

Es un archivo para crear los bean de Spring. Por cada servicio que se quiere proporcionar para cada tabla se creará un bean, es decir, un objeto de esa clase que presta el servicio que creará Spring. Recordamos que estos servicios los instancia solo Spring y que sólo se crea una instancia de cada uno de ellos (patrón singleton), que posteriormente podremos obtener mediante el método *getBean* que nos proporciona la interface *ApplicationContext*.

Crearemos un bean por cada clase contenida en el paquete com.maparecursos.webservice.impl, de la forma en la que se muestra a continuación como ejemplo para usuario.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="usuarioWebService"
          class="com.maparecursos.webservice.impl.UsuarioWebServiceImpl">
    </bean>

    <!-- Resto de declaraciones de bean -->

</beans>
```

Como sabemos dentro de cada elemento bean introducimos un atributo id para poder referenciar cada servicio, y un atributo class para indicar la clase de la que Spring debe realizar la instancia.

- **log4j.properties**:

Log4j [67] es una biblioteca open source desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores elegir la salida y el nivel de granularidad de los mensajes o logs en tiempo de ejecución y no a tiempo de compilación, como es comúnmente realizado. La configuración de salida y granularidad de los mensajes es realizada en tiempo de ejecución mediante el uso de archivos de configuración en formato XML o en formato Java Properties (clave=valor), como este llamado log4j.properties.

Existen varios niveles de prioridad en los mensajes. Nuestra configuración elige el nivel WARN que se utiliza para mensajes de alerta sobre eventos de los que se desea mantener constancia, pero que no afectan al correcto funcionamiento del programa. Además configuramos un registro stdout para que imprima los mensajes por consola mediante líneas (%m%n). Esta API puede usarse desde java para el envío de estos mensajes o logs.

```
log4j.rootLogger=WARN, stdout
log4j.logger.org.springframework.ws=INFO
log4j.logger.org.springframework.xml=DEBUG

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
```

✓ Directorio src/main/webapp

Dentro de este directorio hay una carpeta de nombre WEB-INF, y dentro de ella nos encontramos con el archivo de configuración web.xml y con otra carpeta services.

- **WEB-INF/web.xml**

El archivo web.xml contiene la configuración de cómo se crean los web services. Lo primero que tenemos que hacer es declarar en nuestro web.xml el servlet que se encargará de mapear nuestro WebServices. Establecemos que nuestro servlet se hará cargo de las peticiones que contengan la cadena **/services/**. Además en contextConfigLocation indicamos dónde tenemos la configuración de Spring, en el archivo applicationContext.xml, y el listener *ContextLoaderListener* que se encarga de cargar el contexto de Spring.

Este archivo web.xml queda de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
    <display-name>Servicio Web Mapa Recursos</display-name>

    <context-param>
```

```
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath:/applicationContext.xml
        </param-value>
    </context-param>

    <listener>
        <listener-class>
             org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>

    <servlet>
        <servlet-name>AxisServlet</servlet-name>
        <servlet-class>
              org.apache.axis2.transport.http.AxisServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>

</web-app>
```

- **WEB-INF/services**

Dentro de esta carpeta services están contenidas una carpeta por cada servicio que vamos a crear, un servicio de nuevo por cada tabla de la base de datos. Estas carpetas se llaman: UsuarioWebService, ParqueWebService, ParkCocheWebService, ParkMotoWebService, ParkBiciWebService, PtoLimpioFijoWebService, PtoLimpioMovilWebService, ContRopaWebService, PtoSuministroWebService. Estas dos carpetas complementarán la url del web services ya que tendremos que navegar por ellas para su uso.

Cada una de estas carpetas contendrá de nuevo otra carpeta de nombre META-INF, y dentro de ella un archivo para configurar el servicio de nombre services.xml en todos los casos.

El archivo xml contiene un elemento *<serviceGroup>,* y dentro de él podemos definir tantos servicios como queramos con el elemento *<service>.* Nosotros declararemos un solo servicio. Este elemento *<service>* llevará un atributo *name* para indicar el nombre del servicio, que siempre se debe corresponder con el directorio donde está almacenado el xml.

Dentro de este elemento <service> encontramos otros 3 elementos importantes:

- *<description>:* para indicar una descripción del servicio que se está configurando. Este elemento es opcional y si no se incluyese se devolvería el mismo nombre del servicio como descripción, pero tener una descripción del servicio suele ser muy útil para los usuarios que van a acceder al servicio. El valor de este elemento puede ser texto puro o un segmento de código HTML.

- *<parameter>:* para indicar los parámetros que debe usar el wsdl. Este elemento contiene dos atributos, uno obligatorio *name* para indicar el nombre del parámetro, y otro opcional *locked* para permitir o no que el valor del parámetro pueda ser sobrescrito por un hijo del servicio. Nosotros por defecto este atributo *locked* lo hemos puesto a false, aunque no sería necesario ya que no usamos ninguna jerarquía de servicios. Debemos definir pues un parámetro para indicar el bean que debe crearse para el servicio, este parámetro de nombre *SpringBeanName* tendrá el mismo valor que le dimos al bean en cuestión en el archivo de creación de beans applicationContext.xml, y otro parámetro de nombre *ServiceObjectSupplier* para indicar el proveedor de estos objetos o beans que será *SpringServletContextObjectSupplier*.

- *<operation>:* para indicar las operaciones que existen en el servicio, con un atributo *name* para indicar el nombre de esa operación. Dentro del elemento *<operation>* se incluye un elemento *<messageReceiver>* para indicar el método de recepción de mensajes en el servicio. Cada operación puede contener un método diferente. Nosotros usamos el protocolo Llamada a Procedimiento Remoto (RPC, Remote Procedure Call). Este protocolo RPC es muy utilizado dentro del paradigma cliente-servidor, siendo el cliente el que inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y enviando éste de vuelta el resultado de dicha operación al cliente.

Por ejemplo, para parques, su archivo de configuración del servicio service.xml sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceGroup>
    <service name="ParqueWebService">
        <description>WebService de Parque</description>
        <parameter name="ServiceObjectSupplier" locked="false">
                  org.apache.axis2.extensions.spring.
                  receivers.SpringServletContextObjectSupplier</parameter>
        <parameter name="SpringBeanName" locked="false">
                  parqueWebService</parameter>
        <operation name="buscaPorServicios">
            <messageReceiver
                class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />
        </operation>
        <operation name="buscaTodos">
            <messageReceiver
                class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />
        </operation>
    </service>
</serviceGroup>
```

Una vez realizado todo esto ya tenemos nuestro servidor implementado. El siguiente paso es compilarlo en Eclipse mediante la opción Run As-> Maven build, con la configuración “clean install” como se dijo anteriormente. Esto creará un archivo .war con el servidor plegado que se almacena dentro de la carpeta target de la raíz del proyecto. Ya podemos subir este archivo .war a nuestro servidor de aplicaciones Cloudbees para que nos despliegue los servicios.

Una vez desplegado el archivo .war en Cloudbees podremos acceder a la url del web service para comprobar que está activo. Siguiendo el ejemplo de parques, si entramos a la url http://proyssii.maparecursos.cloudbees.net/services/ParqueWebService?wsdl desde un navegador obtendremos lo siguiente:


```
- <wsdl:definitions targetNamespace="http://impl.webservice.maparecursos.com">
    <wsdl:documentation>ParqueWebService</wsdl:documentation>
  - <wsdl:types>
    - <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="http://impl.webservice.maparecursos.com">
        <xs:import namespace="http://dominio.maparecursos.com/xsd"/>
      - <xs:element name="buscaPorServicios">
        - <xs:complexType>
          - <xs:sequence>
              <xs:element minOccurs="0" name="zonaInfantil" type="xs:short"/>
              <xs:element minOccurs="0" name="zonaMayores" type="xs:short"/>
              <xs:element minOccurs="0" name="zonaDeportiva" type="xs:short"/>
              <xs:element minOccurs="0" name="carrilBici" type="xs:short"/>
              <xs:element minOccurs="0" name="circuitoBiosaludable" type="xs:short"/>
              <xs:element minOccurs="0" name="pipican" type="xs:short"/>
              <xs:element minOccurs="0" name="puntoInformacion" type="xs:short"/>
              <xs:element minOccurs="0" name="aseoPublico" type="xs:short"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      - <xs:element name="buscaPorServiciosResponse">
        - <xs:complexType>
          - <xs:sequence>
              <xs:element maxOccurs="unbounded" minOccurs="0" name="return" nillable="true" type="ax216:Parque"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      - <xs:element name="buscaTodosResponse">
        - <xs:complexType>
          - <xs:sequence>
              <xs:element maxOccurs="unbounded" minOccurs="0" name="return" nillable="true" type="ax216:Parque"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:schema>
    - <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="http://dominio.maparecursos.com/xsd">
```

5.3. Arquitectura del Cliente

El cliente se compone de un proyecto Android de nombre MapaRecursos. La versión usada de Android es la 4.2.2 que se corresponde con la API 17 (noviembre 2012), y la versión mínima en la que puede ejecutarse la aplicación es Android 2.2 que se corresponde con la API 8 (mayo 2010).

Un proyecto Android está formado básicamente por un descriptor de la aplicación llamado AndroidManifest.xml, el código fuente Java y una serie de ficheros con recursos. Cada elemento se almacena en una carpeta específica. Vamos a ir viendo cada carpeta o fichero individualmente.

- **src**: carpeta que contiene el código fuente de la aplicación.

- **gen**: carpeta que contiene el código generado de forma automática por el SDK. Nunca hay que modificar estos ficheros. Dentro de esta carpeta encontraremos un paquete, en nuestro caso de nombre *com.maparecursos*, y dentro de él encontramos dos ficheros:

    - BuildConfig.java: que define la constante debug para que desde java sepamos si nuestra aplicación está en fase de desarrollo mediante un atributo estático puesto a true.

    - R.java: que define una clase que asocia los recursos de la aplicación con los identificadores. De esta forma los recursos podrán ser accedidos desde java. La clase R es controlada automáticamente por el complemento Eclipse de Android. Al colocar un archivo en cualquier parte del directorio res, el complemento se percata del cambio y añade automáticamente el identificador correspondiente de recurso en R.java en el directorio gen. Cuando se elimina o se cambia un archivo de recurso, R.java se vuelve a sincronizar.

  Además este paquete contiene, en nuestro caso, un fichero Manifest.java donde se da privilegios a la aplicación para el uso de los mapas.

  También dentro de esta carpeta gen se incluye otro paquete de nombre com.google.android.gms que incluye otro fichero R.java referente a los mapas.

- **Android 4.2.2**: contiene la librería .jar de Android de la versión seleccionada, en nuestro caso la 4.2.2, API 17.

- **Android Dependencies**: contiene las librerías asociadas al proyecto. Las librerías que necesitamos son:

    - **google-play-services_lib.jar** y **google-play-services.jar**: para el uso de la API de Google Maps en la aplicación.

- **android-support-v4.jar**: para que los mapas se pudiese mostrar en versiones antiguas de Android, en concreto desde la 2.2.

- **ksoap2-android-assembly-2.4-jar-with-dependencies.jar**: para la conexión con el servicio web y poder obtener y almacenar información en la base de datos.

- **additional.jar**, **mail.jar** y **activation.jar**: para el envío del email en caso de que el usuario quiera recuperar su contraseña en caso de olvido.

- **assets**: carpeta que puede contener una serie de ficheros o carpetas que podrán ser utilizados por la aplicación (ficheros de datos, fuentes,…). A diferencia de la carpeta res, nunca se modifica el contenido de los ficheros de esta carpeta ni se les asociará un identificador. Esta carpeta no la hemos usado en nuestro proyecto.

- **bin**: en esta carpeta se compila el código y se genera el .apk, que es el fichero comprimido que contiene la aplicación final lista para instalar en el dispositivo.

- **libs**: contiene las librerías que queramos usar en el proyecto. Aparecen las anteriormente nombradas.

- **res**: carpeta que contiene los recursos usados por la aplicación. Esta carpeta contiene a su vez varias carpetas:

    - drawable: en esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML. Puede haber varias carpetas de este tipo con un sufijo que especifica la densidad gráfica de los recursos que contiene. Estas densidades son: hdpi, ldpi, mdpi y xhdpi. De esta forma el dispositivo elegirá los recursos que más se adecúen a su resolución gráfica. Por motivos de almacenamiento, nosotros sólo usaremos la carpeta drawable-hdpi que contendrá todos los recursos de la aplicación, para evitar el duplicado de estos y agrandar los bytes que ocupa la aplicación.

    - layout: contiene ficheros XML que nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación.

    - values: contiene ficheros XML para indicar valores del tipo string, colores o estilo. De esta manera podremos cambiar los valores sin necesidad de ir al código fuente. Por ejemplo, nos permitiría traducir una aplicación a otro idioma. En esta carpeta también incluimos un fichero llamado tipo_via.xml que incluye los tipos de vía que el usuario podrá elegir en la aplicación, que son calle, avenida, carretera, paseo y plaza.

- raw: ficheros adicionales que no se encuentran en formato XML. Aquí encontramos dos ficheros llamados vciclistas.txt y callesseguras.txt con los tramos de las rutas de bici a pintar en el mapa. Ampliaremos esta información más adelante.

- **AndroidManifest.xml**: este fichero describe la aplicación Android. En él se indican las actividades, intenciones, servicios y proveedores de contenido de la aplicación. También se declaran los permisos que requerirá la aplicación, la versión mínima de Android para poder ejecutarla, el paquete Java, la versión de la aplicación, etc.

- **project.properties:** fichero generado automáticamente por el SDK. Nunca hay que modificarlo. Se utiliza para comprobar la versión del API y otras características cuando se instala la aplicación en el terminal.

❖ Interfaz

En Android existen dos posibilidades para el diseño de interfaces de usuario: el método procedimental y el método declarativo [68].

El método procedimental hace referencia al diseño de interfaces a través de código en Java. Este método ha sido el utilizado para crear y manipular los objetos de la interfaz de usuario más dinámicos. En la aplicación se utiliza este método para mostrar la información ampliada de cada recurso, ya que los campos de información que se muestran al usuario son diferentes según el tipo de recurso. Además también es utilizado para manipular algunos textos e iconos mostrados en las diferentes interfaces.

El diseño declarativo no implica introducir código. A través de etiquetas en archivos .XML se describen los elementos que se desea mostrar al usuario. Este ha sido el método habitual en el diseño de interfaces de la aplicación. Aunque cualquier componente de interfaz de Android realiza la misma función tanto si se gestiona desde Java como desde XML, este es el método recomendado por Google ya que el código XML suele ser más corto y sencillo de comprender que el código Java correspondiente; además es menos probable que cambie en futuras versiones. De esta forma se consigue gestionar cada uno de los elementos que componen la interfaz, situándolos y redimensionándolos mediante los atributos declarativos de cada componente.

Para rellenar el contenido de la pantalla de cualquier actividad se llama al método setContentView(). A este método se le pasa como parámetro un argumento de la forma “R.layout.main” que es un identificador de recurso que hace referencia al archivo “main.xml” situado en el directorio res/layout. En el archivo “main.xml” es donde se declara la interfaz de usuario en XML, por tanto este será el archivo que modificaremos para mostrar al usuario los elementos que nos interesen. Android analiza sintácticamente e instancia el recurso definido en tiempo de ejecución, y lo establece como vista de la actividad actual.

Cada archivo XML de Android empieza con una línea en la que se le indica al compilador que el archivo está en formato XML, con codificación UTF-8 que es prácticamente igual que el texto ASCII normal. La diferencia está en que UTF-8 presenta códigos de escape para caracteres que no están incluidos en ASCII.

El motivo por el cual Android utiliza XML es porque está optimizado para dispositivos móviles con memoria limitada. De todas formas, aunque la interfaz se diseñe en XML, el complemento de eclipse invoca al compilador de recursos Android AAPT (Android Asset Packaging Tool), que procesa el XML en un formato binario comprimido y éste formato es el que se almacena finalmente en el dispositivo.

- ## Composición

Cada archivo XML se compone de contenedores que incluyen uno o varios elementos secundarios en su interior. Estas composiciones se denominan layouts y en ellos se definen los comportamientos para posicionar cada elemento en la pantalla dentro del marco principal.

Las composiciones más comunes son:

- LinearLayout: Este contenedor coloca sus elementos en una sola columna o fila.
- RelativeLayout: Organiza los elementos que contiene en relación a otros elementos o al objeto principal
- TableLayout: Se encarga de colocar los elementos en filas y columnas.

Algunas de las cadenas de caracteres han sido incluidas en el archivo res/values/strings.xml. Además se han definido algunos colores utilizados en la aplicación en el archivo res/values/colors.xml y arrays necesarios en res/values/tipovia.xml.

- ## Dimensiones

Aunque anteriormente se diseñaban las interfaces dimensionando los elementos en términos de píxeles, existen otras alternativas. El problema de los píxeles está en que si ejecutamos el programa en pantallas nuevas con más y más puntos por pulgada (dpi), la interfaz de usuario aparece cada vez más pequeña.

Por ese motivo se han utilizado medidas independientes de resolución para el diseño de las interfaces de esta aplicación. La medida utilizada para establecer algunas dimensiones han sido los píxeles independientes de densidad (dp), que es una unidad abstracta basada en la densidad de pantalla.

Así conseguimos que la interfaz pueda redimensionarse en cualquier tipo de pantalla actual o futura, permitiendo que la aplicación pueda ser utilizada tanto en pantallas de smartphones como en las novedosas tablets con pantallas mucho más amplias.

- **Características**

La aplicación ha sido diseñada para ser utilizada en posición vertical, por tanto las interfaces tienen restringida la rotación de pantalla.

También se ha tratado de fomentar la estandarización del diseño, utilizando la asociación por colores dependiendo del tipo de recurso al que se acceda para conseguir minimizar los requisitos de conocimientos y habilidades por parte del usuario.

Se ha evitado utilizar el botón de "Menú", ya que la tendencia está siendo la desaparición de esté botón en los nuevos dispositivos y está cayendo en desuso.

Para más información sobre el diseño de interfaces de usuario puede acudirse al libro de la editorial Pearson con ese mismo nombre [69].

❖ AndroidManifest.xml

El archivo XML AndroidManifest es el más importante de un proyecto Android. Este archivo se genera automáticamente al crear un proyecto y en él se declaran todas las especificaciones de nuestra aplicación, como son: nombre de la aplicación, versión de Android, activities utilizadas, permisos de requiere la aplicación, etc…

Vamos a ir comentando los elementos que componen este archivo:

- *<manifest>,* dentro de este elemento encontramos los siguientes atributos:

  - *android:versionCode* y *android:versionName*, que hacen referencia a la versión de la propia aplicación.
  - *package*, es el paquete de la aplicación para poder referenciarla.

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

- *<uses-sdk>,* para determinar las distintas versiones Android que va a utilizar la aplicación, mediante los atributos *android:minSdkVersion* y *android:targetSdkVersion*.

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

- *<uses-permissions>* y *<permission>,* para especificar los permisos que va a necesitar la aplicación para poder ejecutarse. Además son los que deberá aceptar el usuario antes de instalarla. En nuestro caso estos permisos serán:

- INTERNET: para acceso a Internet.
- MAPS_RECEIVE: permite que una aplicación pueda recibir el mapa de servicios de Google y realizar procesamientos en ellos.
- READ_GSERVICES: permite que una aplicación pueda modificar el mapa de servicios Google.
- ACCESS_NETWORK_STATE: permite que las aplicaciones accedan a información sobre redes.
- WRITE_EXTERNAL_STORAGE: permite que una aplicación escriba en el almacenamiento externo.
- ACCESS_COARSE_LOCATION: permite que una aplicación acceda a la ubicación aproximada derivada de fuentes de ubicación de red y Wi-Fi.
- ACCESS_FINE_LOCATION: permite que una aplicación acceda a la ubicación precisa de fuentes de localización como el GPS.

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

- *<application>,* aquí adentro van todas las Activities, Services, Providers, Receivers y las bibliotecas que se usan en nuestra aplicación. Entre los atributos de este elemento destaca *android:icon*, para indicarle el icono de la aplicación. Además dentro de este elemento encontraremos otros elementos como son:

  - *<meta-data>:* lo usamos para introducir la API Key necesaria para el funcionamiento de los mapas. De ella se hablará en la sección de Google Maps.
  - *<activity>:* se tendrá una activity por cada pantalla de nuestra aplicación. Con el atributo *android:name* indicaremos el nombre de la clase de nuestra Activity. Además con el atributo *android:screenOrientation* determinamos que las actividades sólo se dispondrán en modo *portrait*, es decir, con el teléfono en posición vertical. El otro modo válido es el modo *landscape* para el teléfono en posición horizontal. Para el caso de la actividad inicial debemos crear dentro un intent para que salte de manera automática al iniciar la aplicación.

```
<meta-data
     android:name="com.google.android.maps.v2.API_KEY"
     android:value="AIzaSyDNEdPE9DCtXzFHfdlZvlZl4WLk0X3kuYQ" />

<activity android:name="com.maparecursos.estructura.MainActivity"
          android:screenOrientation="portrait">
           <intent-filter>
```

```
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
<activity android:name="com.maparecursos.estructura.MenuPrincipal"
        android:screenOrientation="portrait"></activity>
```

❖ Código fuente – carpeta src

Como hemos comentado, esta carpeta contiene el código fuente de la aplicación. El código está distribuido en 7 paquetes. En la siguiente figura podemos ver el nombre de estos paquetes y como se relacionan entre ellos.

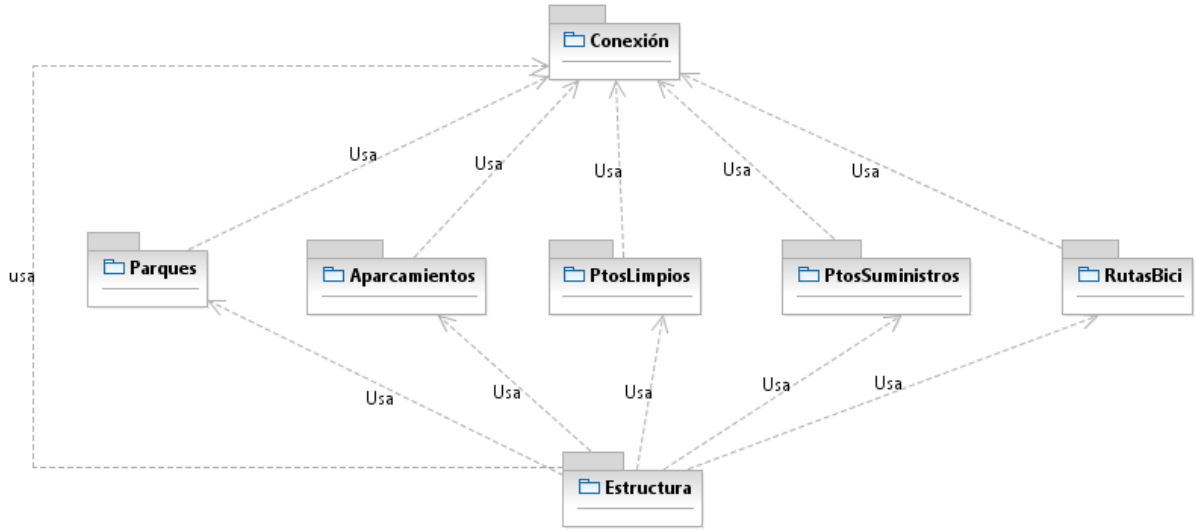


Figura 56. Paquetes código fuente proyecto Android

A grandes rasgos el paquete *Estructura* contiene las clases comunes y las que extienden de la clase Activity de Android para dar funcionalidad a las pantallas de la aplicación. Las clases de este paquete usarán las clases contenidas en los paquetes de nombre *Parques*, *Aparcamientos*, *PtosLimpios*, *PtosSuministros* y *RutasBici*. Estos paquetes contienen las clases relacionadas con el recurso que les da nombre. Además estos paquetes y el paquete *Estructura* usan las clases contenidas en el paquete *Conexión*. Este paquete dispone de todas las clases para realizar la conexión al web service y poder obtener la información almacenada en base de datos o almacenar nuevos datos.

Vamos a ir comentando cada paquete y destacando las técnicas especiales utilizadas.

• Paquete *Conexión*

Comenzamos hablando de este paquete ya que será usado por los demás. En él se incluyen las clases para realizar la conexión con el web service SOAP. Para ello usamos la librería kSOAP2-android, la cual está incluida en el proyecto. SOAP es uno de los protocolos para comunicación

de datos más usados, existiendo multitud de servicios web implementados bajo esta tecnología. La librería kSOAP2-android es una adaptación para Android de la librería kSOAP2 [70].

Además para que la conexión funcionase correctamente tuvimos que añadir las siguientes líneas de código dentro de la primera activity de la aplicación (MainActivity.java), para dar privilegios de acceso al servidor. Estos privilegios eran requeridos para versiones de Android 3.0 o superiores.

```
//politicas de acceso al servidor a partir versiones 3.0
StrictMode.ThreadPolicy policy =
            new StrictMode.ThreadPolicy.Builder().permitAll().build();
StrictMode.setThreadPolicy(policy);
getWindow().setSoftInputMode(
            WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HIDDEN);
```

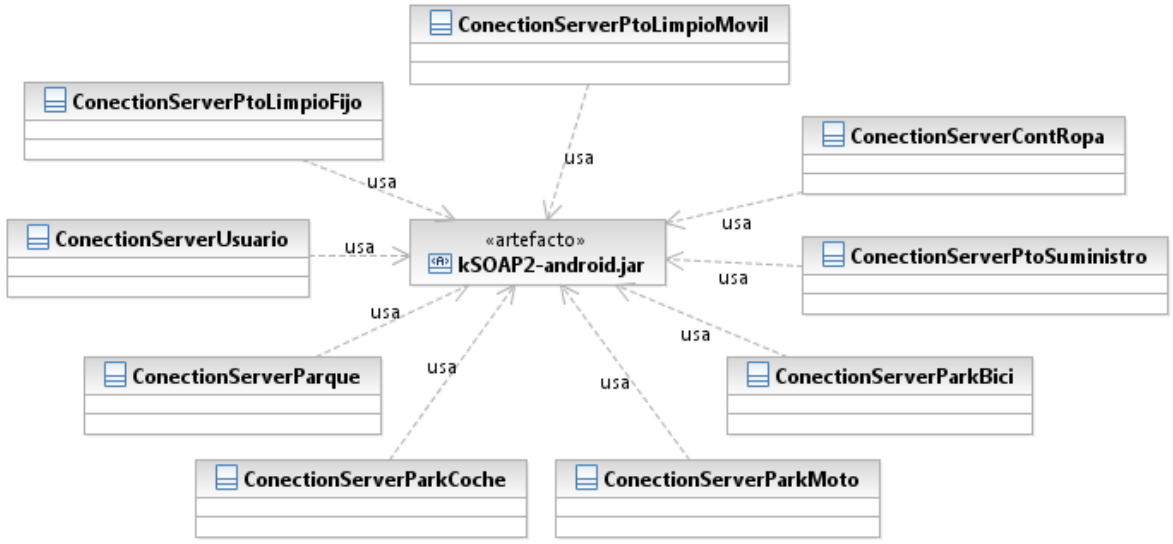


Figura 57. Clases que usan la librería kSOAP2-android

Tenemos 9 clases que usan la librería kSOAP2-android para la conexión con el servidor. Son 9 clases ya que cada una se conectará a uno de los 9 servicios diferentes que creamos según se explicó anteriormente.

Estas clases siguen el esquema de la siguiente figura.

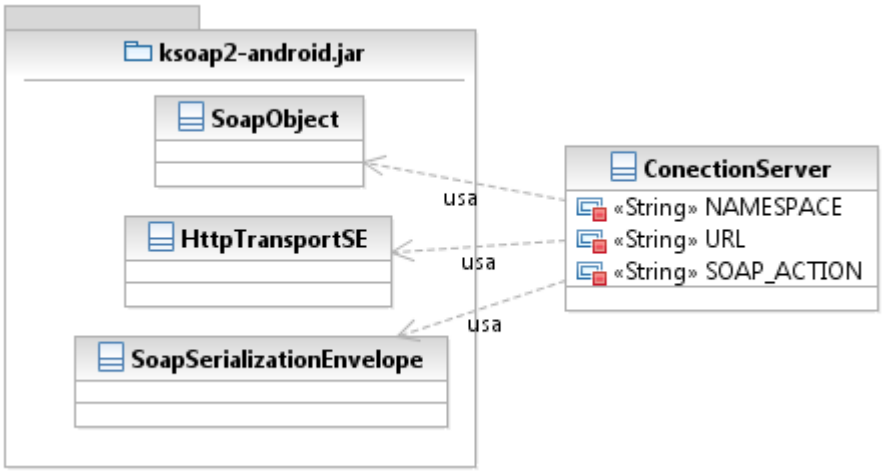


Figura 58. Esquema de Conexion a Servicio Web

Estas clases contienen 3 atributos en común, como se ve en la anterior figura, que son:

- **NAMESPACE:** para indicar el espacio de nombres del servicio web. En nuestro caso será http://impl.webservice.maparecursos.com.
- **URL:** para indicar la ruta del servicio web. En nuestro caso será: http://proyssii.maparecursos.cloudbees.net/services/NombreWebService?wsdl
- **SOAP_ACTION:** para indicar el nombre del servicio web al que estamos llamando. En nuestro caso será: NombreWebService.

Una vez tenemos estos atributos necesarios pasamos al siguiente paso. El protocolo SOAP se basa en cinco pasos, que es donde entran en juego el uso de las anteriores clases especificadas en el diagrama:

1. **Definir la petición (request):** definimos la petición mediante un objeto de tipo SoapObject. Al crearlo necesita que le indiquemos el NAMESPACE y el nombre del método del servicio web al que queremos hacer la petición. Además si ese método requiere algunos parámetros, podemos incluirlos en la petición mediante el método *addProperty(nombreParámetro, valorParámetro).*
2. **Configurar una envoltura (envelope):** definimos una envoltura para la petición mediante un objeto de tipo SoapSerializationEnvelope, y al crearlo le pasamos la constante SoapEnvelope.VER11. Además a este objeto debemos introducirle el objeto de la petición que hemos creado antes con el método setOutputSoapObject(request, para crear esa petición, y configuramos el atributo doNet a cierto.
3. **Definir el canal de transporte**: para ello creamos un objeto de tipo HttpTransportSE, y al crearlo necesita que le indiquemos la URL.
4. **Hacer la llamada:** con el objeto de tipo HttpTransportSE podemos realizar la llamada con el método *call(SOAP_ACTION, envelope).* Esta llamada debe ir dentro de un bloque try-catch para capturar la excepción en caso de error.
5. **Recoger los datos:** para obtener los datos de la llamada hacemos envelope.bodyIn, y obtenemos un objeto de tipo SoapObject del que podremos sacar los datos que nos ha devuelto la llamada mediante el método getProperty.

Un ejemplo de estos 5 pasos para llamar al servicio web de usuario con el método login sería:

```
public int loginUsuario(String email, String contraseña) {
            //Definimos la petición
            SoapObject request = new SoapObject(NAMESPACE, LOGIN_USUARIO);
            request.addProperty(LOGIN_USUARIO_PARAM1, email);
            request.addProperty(LOGIN_USUARIO_PARAM2, contraseña);
            //Configuramos la envoltura
            SoapSerializationEnvelope envelope =
                     new SoapSerializationEnvelope(SoapEnvelope.VER11);
            envelope.setOutputSoapObject(request);
            envelope.dotNet = true;
            //Definimos el canal de transporte
            HttpTransportSE androidHttpTransport =
                     new HttpTransportSE(URL_USUARIO);

            SoapObject result=null;
            int resultado=-2;
            try {
                  //Hacemos la llamada
                  androidHttpTransport.call(SOAP_ACTION_USUARIO, envelope);
                  //Recogemos los datos
                  result = (SoapObject) envelope.bodyIn;
            } catch (Exception e) {e.printStackTrace();}
            if(result!=null){
                  resultado =
                 Integer.parseInt(result.getProperty("return").toString());
            }
            return resultado;
      }
```

Vamos a repasar cada clase para la conexión y los métodos que poseen ya que los llamaremos desde las demás clases de la aplicación:

CLASES	MÉTODOS
ConectionServerUsuario	**int loginUsuario(String email, String contraseña):** Dado el email y la contraseña de un usuario, se comprueba que exista en la base de datos. Se devuelve un 0 si el login es correcto, y en otro caso, se devuelve otro número distinto de cero para indicar el error correspondiente. **int altaUsuario(String email, String contraseña, String sexo, int añoNac):** Dados los datos del usuario, se envían al servidor para dar de alta un nuevo usuario en la base de datos, si no existe ya un usuario con ese email. Se devuelve un 0 si el alta es correcto, y en otro caso, se devuelve otro número distinto de cero para indicar el error correspondiente. **int cuentaUsuarios():** Se devuelve el número de usuarios registrados que hay en la base de datos. Si el número es negativo indicará un error de conexión. **String[] consultaUsuario(String email):** Dado un email, se consulta en la base de datos el usuario que corresponde a ese email y se devuelven sus datos en un array de String. Se devuelve null si no hay ningún usuario con ese

	email. **`int modificaUsuario(String id, String email, String contraseña, String sexo, int añonacimiento):`** Dados los nuevos datos de un usuario, se actualizarán en la base de datos. Se devuelve un 0 si la modificación se ha realizado correctamente, y en otro caso, se devuelve otro número distinto de cero para indicar el error correspondiente.
ConectionServerParque	**`ListParque buscaTodos():`** Se devuelven todos los parques alojados en la base de datos en un objeto ListParque. **`ListParque buscaPorServicios(boolean zInf, boolean zMay, boolean zDep, boolean cBici, boolean cBio, boolean pipican, boolean pInfo, boolean aseos):`** Dados unos servicios, se devuelven todos los parques alojados en la base de datos que contengan esos servicios en un objeto ListParque.
ConectionServerParkCoche, ConectionServerParkMoto, ConectionServerParkBici, ConectionServerPtoLimpioFijo, ConectionServerPtoLimpioMovil, ConectionServerContRopa	**`List buscaTodos():`** Se devuelven todos los recursos correspondientes alojados en la base de datos en un objeto List correspondiente a cada recurso.
ConectionServerPtoSuministro	**`ListPtosSuministros buscaPorCombustible (String combustible):`** Se devuelven todos los puntos de suministro cuyo combustible se corresponda con el pasado por parámetro, en un objeto ListPtosSuministros.

Además de estas 9 clases para la conexión con el servidor disponemos de dos clases para el envío de emails desde el dispositivo. Estas dos clases se llaman *JSSEProvider* y *GMailSender*. Gracias a las 3 librerias incluidas en el proyecto (mail.java, additionnal.java y activation.java), nos servimos de la funcionalidad de JavaMail para el envío del email al usuario.

La clase *JSSEProvider* extiende la clase *Provider* y su objetivo es asegurar la seguridad del envío. La clase *GMailSender* entiende la clase *Provider* y además posee un objeto del tipo *JSSEProvider*. La clase *GMailSender* implementa dos métodos importantes. El primero es su propio constructor por parámetros que recibe el email y la contraseña desde la cuenta donde se va a enviar el email. De esta forma crea un objeto de tipo PasswordAuthentication con estos datos. El segundo método importante es *sendMail(String subject, String body, String sender, String recipients),* que recibe por parámetros el asunto del email, el cuerpo del email, el correo electrónico desde el que se realiza el envío y el correo electrónico al que se debe enviar el correo.

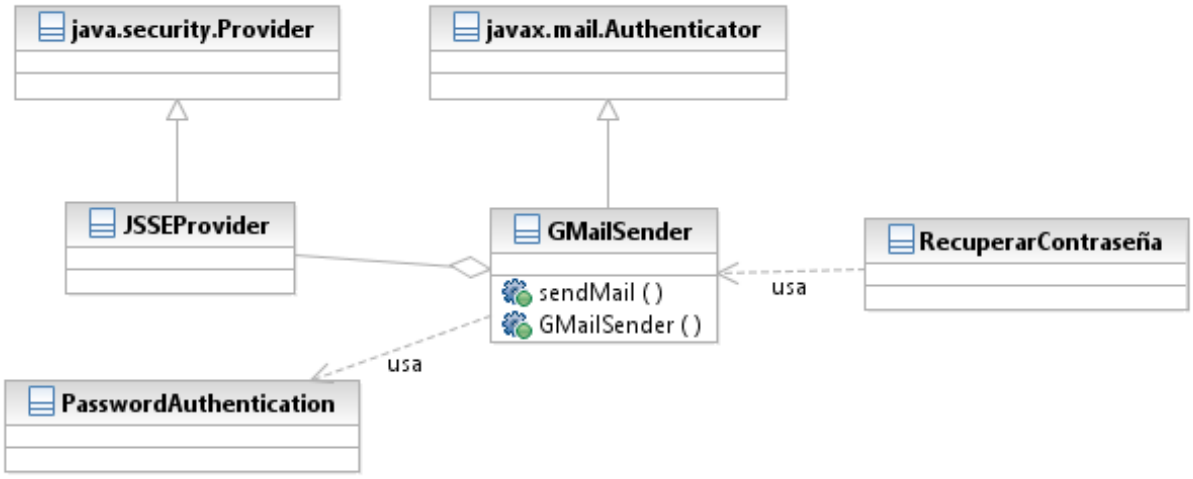


Figura 59. Clases para envío de email

De esta forma desde la clase *RecuperarContraseña* se crea un objeto de tipo *GMailSender*, y se llama al método *sendMail* con él de la siguiente forma:

```
GMailSender sender =
         new GMailSender("maparecursos2013@gmail.com", "****");
sender.sendMail("Recuperar contraseña - Mapa Recursos Ambientales",
   "Mapa Recursos Ambientales \n \n Su contraseña en la aplicación es: \n "+
    datos[2] + "\n Gracias por utilizar el servicio.",
    "maparecursos2013@gmail.com",
      email);
```

La contraseña del correo eléctronico que realiza el envío se ha ocultado por seguridad.

- ## Listas de Recursos

✓ ### Representación de los Recursos

A la hora de implementar nuestra aplicación nos decidimos por un modelo de **programación orientado a objetos** (POO), ya que Java es un lenguaje de programación que entre otras características está orientado a objetos. La POO no es un lenguaje específico, o una tecnología, sino una forma de programar. Las ventajas que más nos interesan de este modelo de programación son 3, que son las más características de la POO:

- **Abstracción**: la capacidad de separar los elementos para poder verlos de forma singular.
- **Encapsulación**: la encapsulación se encarga de mantener ocultos los procesos internos, dándole al programador acceso sólo a lo que necesita. Esto otorga dos ventajas iniciales, la primera es que lo que hace el usuario puede ser controlado internamente (incluso sus errores), evitando que todo se colapse. La segunda ventaja es que, al hacer que la mayor parte del código esté oculto, se pueden hacer cambios y/o mejoras sin que eso afecte al modo en que los usuarios van a utilizar el código.

- **Herencia**: la herencia nos permite, entre otras cosas, evitar tener que escribir el mismo código una y otra vez, puesto que al definir que una clase pertenece a otra, automáticamente estamos atribuyéndoles las características generales de la primera sin tener que definirlas de nuevo. Nos permite compartir automáticamente métodos y datos entre clases, subclases y objetos.

Nuestro proyecto está formado por diversos recursos medioambientales, por ello necesitamos una clase representativa de cada recurso. Pero todos tienen atributos comunes, así que decidimos hacer una clase padre ***Recurso*** que contuviera todos esos atributos comunes a todos los recursos. Estos atributos son:

- **identificador**: es un número entero que usaremos para identificar a cada recurso.
- **distancia**: la usaremos cada vez que vamos a mostrar un tipo de recurso en el mapa obteniendo la posición del usuario y calculando la distancia hasta ese recurso mediante la fórmula de Haversine, que se explicará más adelante. Esta distancia será recalculada cada vez que el usuario muestre distintos recursos.
- **ndp**: es un número con una codificación especial usada por el departamento de Informática del Ayuntamiento de Madrid (IAM). Este atributo en sí no se usa dentro de la aplicación, pero ha sido incluido para su futuro uso por parte del Ayuntamiento.
- **latitud y longitud**: contienen las coordenadas geográficas en formato decimal de la posición del recurso. Estos números son números decimales, pero no usamos el tipo de datos double, que es el tipo de datos primitivo (es decir, representa números decimales con precisión doble para sus decimales) y en cambio usamos Double que es la clase que representa a estos números como un objeto. Se las llama clase envoltorio y java las proporciona para cada uno de los tipos básicos de datos que incorpora. Estas clases contienen un único campo o atributo cuyo tipo es del tipo primitivo correspondiente. De esta forma nos permiten el tratamiento de los tipos básicos como objetos, proporcionándonos diferentes métodos de utilidad que necesitábamos a la hora de la implementación, como por ejemplo transformarlos a tipo String.

Además de todos estos atributos, la clase *Recurso* contiene todos los métodos getter y setter correspondientes a cada atributo así como el constructor por defecto y por parámetros.

De este objeto padre *Recurso* se extienden todos los recursos que mostramos en nuestra aplicación, como puede observarse en el siguiente diagrama. Así cada recurso dispone de sus propios atributos y de los atributos que hereda de la clase *Recurso*. Los atributos propios de cada recurso siguen la misma consonancia seguida desde la creación de la base de datos, luego no necesitan mayor explicación.

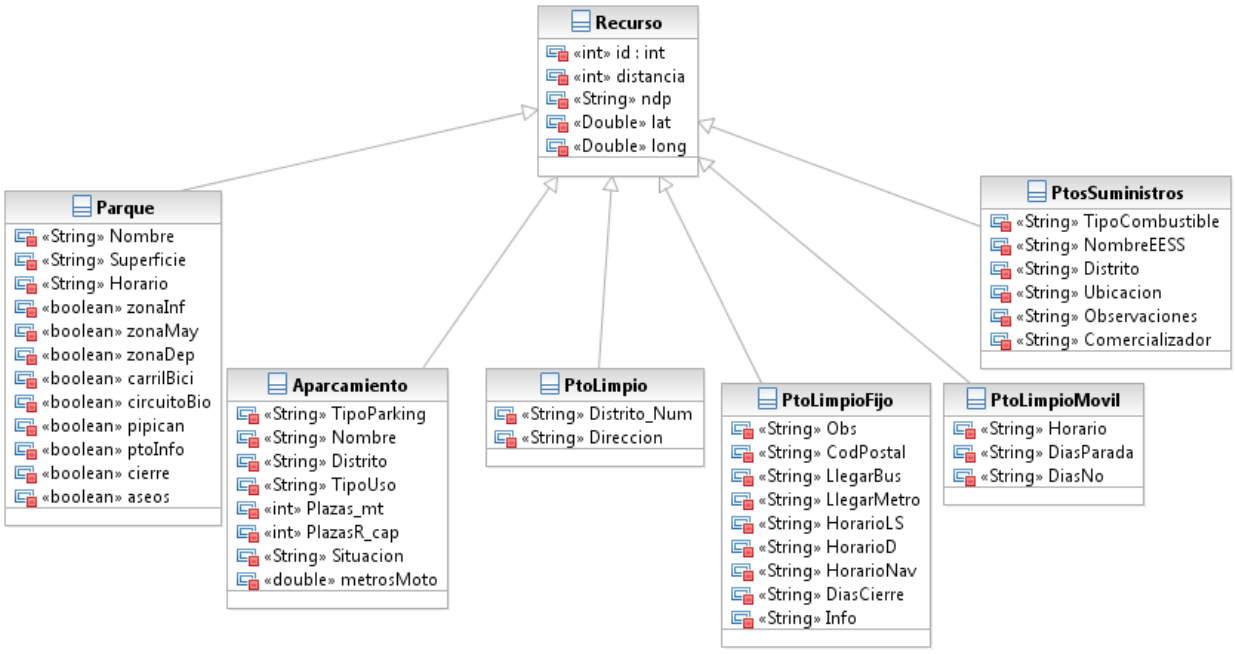


Figura 60. Herencia de la clase Recurso

Pero nosotros no trabajamos con recursos sueltos en sí, es decir, con un sólo recurso, sino que necesitamos manipular conjuntos de recursos, por ejemplo, ordenados en forma de lista. De esta forma creamos el siguiente concepto.

✓ Listas abstractas de Recursos

AbsLista es una clase abstracta, es decir, una clase de la que no se pueden crear objetos. La utilidad de este tipo de clases estriba en que otras clases hereden de ésta, por lo que con ello conseguiremos reutilizar código. La principal característica que nos interesa de este tipo de clases es que todas las subclases que hereden de una clase abstracta tendrán que redefinir los métodos abstractos dándole una implementación. También nos permite incluir variables y métodos no abstractos que heredarán las clases que extiendan ésta.

El objetivo principal de esta clase es crear un *ArrayList* que contenga nuestros propios objetos personalizados (*Recurso*) y que implemente la interfaz *Parceable* para poder mandar estos datos a través de un *Bundle* cargándolo en un *Intent*, que es la forma tradicional de mandar datos a una *Activity*. Explicaremos más adelante que es un *Intent* y un *Bundle*.

Como un *Intent* sólo nos ofrece la opción de mandar tipos primitivos y arrays de los mismos, para mandar nuestros propios objetos de tipo *Recurso* podríamos serializar nuestras clases implementando la interface *Serializable*. Sin embargo la serialización en Android supone un gran problema de rendimiento. Para solucionar este problema Android dispone de la interface *Parceable*, que es una interface específica para serializar nuestros objetos y poder pasarlos entre actividades optimizando el rendimiento.

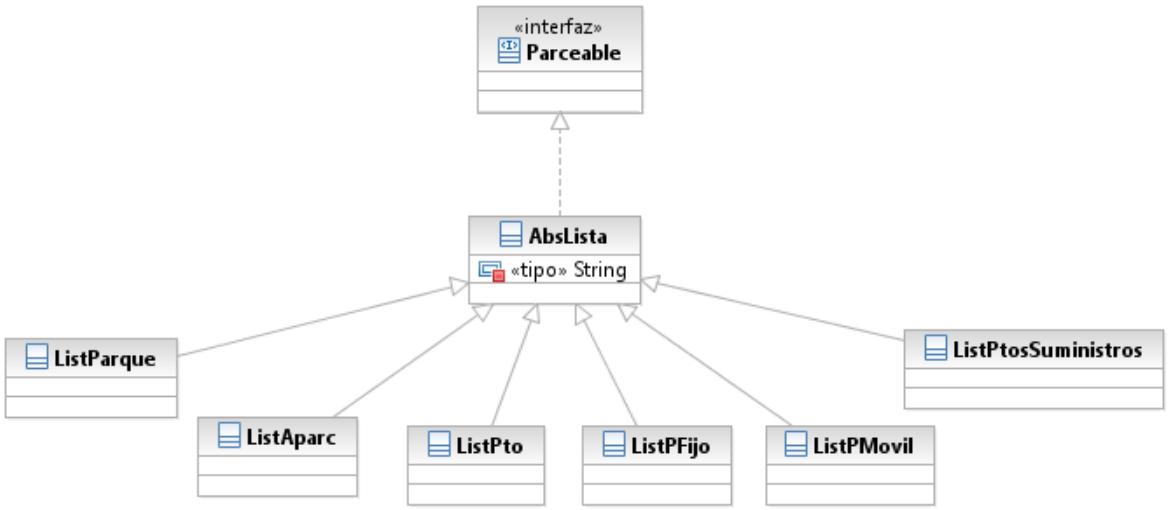


Figura 61. Lista abstracta AbsList con las clases que extienden de ella y la interface que implementa

Esta clase *AbsLista* contiene principalmente un atributo llamado *tipo*, de tipo String, en el cual a la hora de crear una clase que extienda de ésta rellenaremos con el nombre del tipo de lista que estamos definiendo para posteriormente identificarla para distintos usos. Esto es, si el *ArrayList* de la lista abstracta contiene objetos de tipo *Parque*, entonces el atributo *tipo* de *AbsLista* será "Parque".

Por el funcionamiento que tiene la interfaz *Parceable* es necesario que previamente todos los objetos *Recurso* tengan definidos sus métodos *Get* y *Set* para poder acceder al contenido de éstos. En cada lista de recursos que extiende de *AbsLista* damos valor al atributo llamado *tipo*, de tipo String, poniendo un nombre significativo de los recursos que contendrá esa lista. Por ejemplo la lista de aparcamientos su constructor por defecto será:

```
public ListAparc(){
            super("Aparcamientos");
}
```

Luego pasamos a definir un constructor para la "serialización de la clase", al cual llamaremos cuando queramos recuperar los valores. Para ello necesitamos declarar una variable estática de tipo *Parceable.Creator* que llamaremos *CREATOR* y que va a ser la encargada de implementar la interfaz *Parceable.Creator*, de la manera que se muestra en el ejemplo para la lista de aparcamientos:

```
public static final Parcelable.Creator CREATOR = new Parcelable.Creator(){
            public ListAparc createFromParcel(Parcel in){
                  return new ListAparc(in);
}};
```

Para esto tenemos que definir un constructor parametrizado el cual reciba un objeto de tipo *Parcel*:

```
public ListAparc(Parcel in){
            super("Aparcamientos");
            readfromParcel(in);
}
```

Y ahora pasamos a definir los dos métodos de la interfaz *Parceable.Creator* mediante los cuales vamos a poder leer y devolver los valores previamente almacenados. Para almacenar los datos tenemos que implementar el método *writeToParcel(Parcel destino,int flags)*. Este método se lanzará cada vez que escribimos en nuestro *ArrayList*. La clase *Parcel* nos provee métodos específicos para cada uno de los valores que vamos a poder almacenar, todos de tipos primitivos y arrays de esos tipos excepto de booleanos, los cuales sólo nos deja pasar arrays de booleanos pero no uno individual, lo que resolvimos mandando arrays de booleanos de tamaño uno.

```
public void writeToParcel(Parcel dest, int flags) {
            int size = this.size();
            dest.writeInt(size);
            for (int i = 0; i < size; i++){
                    Aparcamiento Aparc = (Aparcamiento) this.get(i);
                    dest.writeInt(Aparc.getId());
                    dest.writeString(Aparc.getTipoParking());
                    dest.writeString(Aparc.getNombre());
                    dest.writeString(Aparc.getDistrito());
                    dest.writeString(Aparc.getTipoUso());
                    dest.writeInt(Aparc.getPlazas_mt());
                    dest.writeInt(Aparc.getPlazasR_cap());
                    dest.writeString(Aparc.getSituacion());
                    dest.writeDouble(Aparc.getLat());
                    dest.writeDouble(Aparc.getLong());
                    dest.writeDouble(Aparc.getMetrosMoto());
                    dest.writeString(Aparc.getNdp());
            }
}
```

Para leer los datos disponemos del método *readToParcel(Parcel in).* Es fundamental que leamos los datos en el mismo orden en que los hemos escrito. Este método se ejecutará cada vez que leemos nuestro *ArrayList*.

```
private void readfromParcel(Parcel in){
            this.clear();
            int size = in.readInt();
            for (int i = 0; i < size; i++){
                  //el orden de los atributos SI importa
                  Aparcamiento Aparc = new Aparcamiento();
                  Aparc.setId(in.readInt());
                  Aparc.setTipoParking(in.readString());
                  Aparc.setNombre(in.readString());
                  Aparc.setDistrito(in.readString());
                  Aparc.setTipoUso(in.readString());
                  Aparc.setPlazas_mt(in.readInt());
                  Aparc.setPlazasR_cap(in.readInt());
                  Aparc.setSituacion(in.readString());
                  Aparc.setLat(in.readDouble());
```

```
                    Aparc.setLong(in.readDouble());
                    Aparc.setMetrosMoto(in.readDouble());
                    Aparc.setNdp(in.readString());
                    this.add(Aparc);
                }
        }
```

Finalmente, en nuestra clase *Activity*, una vez declarado el *Intent* pasaremos nuestra lista con *putExtra(String key, Parcelable value),* en la que *value* es la lista que queremos pasar y *key* será un texto identificativo del tipo de información que vamos a pasar, es decir, una clave para recuperar más adelante la información que acabamos de meter en *value*.

Al cargar una segunda actividad recuperamos los valores mediante un objeto *Bundle*, del que explicaremos a continuación su funcionalidad.

- ## Intents y Bundles

Un ***Intent*** es una descripción abstracta de una operación que vamos a realizar. Se puede utilizar con el método *startActivity* para lanzar una actividad, con *broadcastIntent* para enviarlo a cualquiera de los componentes *BroadcastReceiver* interesados, y *startService (Intent)* o *bindService (Intent, ServiceConnection, int)* para comunicarse con un servicio en segundo plano.

Un *Intent* nos proporciona facilidades para realizar enlaces en tiempo de ejecución en el código en diferentes aplicaciones. Su uso más importante es en la puesta en marcha de las actividades, en las que se puede considerar como la unión entre éstas. Se trata básicamente de una estructura de datos pasiva que sostiene una descripción abstracta de una acción a realizar.

Los *Intents* admiten, además de las acciones y datos, un atributo adicional llamado extras. Este tipo de dato viene dado por la forma clave/valor, en la cual el nombre de la clave normalmente suele empezar con el nombre del paquete y el valor puede ser de cualquiera de los tipos primitivos u objetos creados por nosotros, siempre que se implemente la interfaz *Parcelable*. Esta información extra se representa mediante la clase *Bundle*. Podemos definir un *bundle* como un "contenedor" de información que pasaremos de una actividad a otra con todo lo que queramos traspasar o recuperar en diferentes actividades. Para crearlo solo tendremos que hacer *Bundle b=new Bundle(),* y ahora podremos meterle todo la información que queramos mediante las diferentes funciones que nos ofrece como put*"Tipo"(String clave, Tipo valor*), donde *Tipo* del nombre de la función será el tipo que queremos insertar, ya sea un tipo primitivo, serializable, parceable o incluso otro *Bundle* de otro *Intent*. En *clave* pondremos el nombre por el cual luego podremos recuperar esa información, y en valor meteremos el objeto del tipo correspondiente al nombre de la función que queremos pasar.

Una vez recopilada toda la información que queremos en el *Bundle* solo tendremos que pasársela por parámetro a nuestro *Intent* mediante la función *putExtras(Bundle b).* Si el *intent* ya tiene un *bundle*, *putExtras* transfiere los pares clave/valor adicionales del *bundle* nuevo al que ya existía. Si no existe ningún *bundle* asociado, *putExtras* creará uno y copiará todos los valores. Un ejemplo del funcionamiento es el siguiente fragmento de código en el que

pasamos la lista de recursos (en este caso de parques) y la localización del usuario *pos,* que es una array, con dos reales que contienen la latitud y la longitud:

```
Intent intent=new Intent(this,Mapa.class);
Bundle contenedor=new Bundle();
contenedor.putDoubleArray("pos", pos);
contenedor.putParcelable("array",lista_parques);
intent.putExtras(contenedor);
startActivity(intent);
this.finish();
```

Para recuperar esa información desde otra actividad tendremos que usar la función *getExtras* que nos devolverá el B*undle* que contenga el *Intent*. Y una vez tenemos el *Bundle* que acabamos de pasar sólo tendremos que obtener los datos que hemos insertado mediante las funciones *get"Tipo" (String clave)* que nos proporciona el *Bundle*, cambiando el nombre de la función con el tipo que le corresponde a la información que queremos recuperar y escribiendo el String *clave* con la clave que hayamos puesto previamente a la hora de insertarlo mediante el correspondiente método *put*. A continuación vemos un ejemplo en el que recuperamos la información en la nueva actividad a la que le hemos pasado la información:

```
Bundle contenedor=getIntent().getExtras();
lista=contenedor.getParcelable("array");
pos=contenedor.getDoubleArray("pos");
posActual=new LatLng(pos[0],pos[1]);
```

- ## Google Maps

La **API de Google Maps** es una parte muy importante en nuestro proyecto, puesto que sobre ella mostramos todos los recursos y la localización del usuario. Cuando comenzamos el proyecto la versión disponible de esta API era la 1. Como nunca la habíamos usado dedicamos alrededor de un mes a investigar su manejo y sus funcionalidades. Cuando conseguimos hacer algunas pruebas interesantes, esta versión dejo de utilizarse. Oficialmente fue a día 3 de diciembre de 2012, y eso significaba que no añadirían ninguna característica nueva a esta API y aunque las aplicaciones realizadas en esa versión seguirían funcionando en los diferentes dispositivos en las que estuvieran instaladas, todavía no habíamos hecho algo suficientemente relevante como para no volver a empezar de nuevo con la versión 2.

Y eso fue lo que hicimos. Empezamos con la versión 2 la cual ofrecía muchas mejoras y facilidades, entre las que se encuentran:

- Integración con los Servicios de Google Play (Google Play Services) y la consola de APIs (esta consola intenta mejorar la organización de la API de Google en los proyectos en los que la utilicemos).
- Utilización a través de un nuevo tipo específico de fragment (*MapFragment*) que nos permite, entre otras cosas, añadir uno o varios mapas a cualquier actividad.

- Utilización de mapas vectoriales, lo que repercute en una mayor velocidad de carga y una mayor eficiencia en cuanto al uso de ancho de banda.
- Mejoras en el sistema de caché, lo que reducirá en gran medida las famosas áreas en blanco que tardan en cargar.
- Los mapas son ahora 3D, es decir, podremos mover nuestro punto de vista de forma que lo veamos en perspectiva.

En primer lugar, dado que la API v2 se proporciona como parte del SDK de Google Play Services, fue necesario incorporar previamente a nuestro entorno de desarrollo dicho paquete e importar la librería a nuestro proyecto. Es decir, accediendo desde Eclipse al Android SDK Manager y descargando del apartado de extras el paquete llamado Google Play Services.

Como último paso de configuración de nuestro proyecto, si queríamos que nuestra aplicación se pudiese ejecutar en versiones antiguas de Android, en concreto desde la versión 2.2, debíamos asegurarnos de que nuestro proyecto incluía la librería android-support-v4.jar. Las versiones más recientes de Android Development Tools (ADT) incluyen por defecto esta librería en nuestros proyectos, pero si no está incluida se puede hacer mediante la opción del menú contextual "Android Tools/Add Support Library…" sobre el proyecto, o bien de forma manual.

Además necesitábamos obtener una **API Key** para poder utilizar el servicio de mapas de Google en nuestra aplicación. Una API Key es un código aprobado en los programas informáticos para identificar el programa que utiliza esa API. Estas claves se utilizan para realizar un seguimiento y controlar cómo se utiliza la API, por ejemplo, para prevenir el uso inadecuado o abuso de la API. La API Key a menudo actúa como un identificador único y un elemento secreto para la autenticación, que por lo general tendrá un conjunto de derechos de acceso en la API asociada a él.

Para obtener la API Key accedemos a la herramienta de Google llamada Google APIs Console desde cualquier navegador e iniciamos sesión con nuestra cuenta de Google. Una vez dentro creamos un proyecto con el nombre que queramos y accedemos a Services. En esta ventana se encuentran todos los servicios de Google que podemos activar. En nuestro caso activamos el servicio llamado "Google Maps Android API v2". Una vez activado nos aparece una nueva opción llamada API Access.

Esa opción nos ofrece la posibilidad de obtener nuestra API Key que nos permita utilizar el servicio de mapas de Google desde nuestra aplicación particular. Para ello requiere varios datos identificativos de nuestra aplicación, en concreto dos:

- La huella digital (SHA1) del certificado con el que firmamos la aplicación.
- El nombre del paquete java utilizado.

Toda aplicación Android debe ir firmada para poder ejecutarse en un dispositivo, tanto físico como emulado. Y este proceso de firma es algo que se tiene que hacer siempre antes de distribuir públicamente una aplicación. Incluso cuando estamos realizando las pruebas estamos firmando la aplicación con un certificado de pruebas.

Con todo preparado ya podíamos empezar a programar con esta nueva versión. Desde el principio nos dimos cuenta de que todo era mucho más intuitivo de manejar y tenía muchas más opciones que nos daban más posibilidades de elegir la forma de representar nuestros recursos. En la anterior versión, el acceso y modificación de determinados datos del mapa era bastante poco homogéneo por ejemplo la consulta de la posición actual o la configuración del tipo del mapa se hacían directamente sobre el control *MapView* mientras que la manipulación de la posición y el zoom se hacían a través del controlador asociado al mapa *MapController*. Además, el tratamiento de las coordenadas y unidades utilizadas eran poco intuitivas teniendo que estar continuamente convirtiendo de grados a microgrados y de eso a objetos *GeoPoint* que son los que usaba la API v1.

Con la nueva versión todas las operaciones se realizan sobre un objeto *GoogleMap*, que es el componente base de la API. Accederemos a este componente llamando al método *getMap()* del fragmento *MapFragment* que contenga nuestro mapa. Una vez obtenida esta referencia a nuestro objeto *GoogleMap* podremos realizar la mayoría de acciones básicas del mapa.

```
GoogleMap mapa = ((SupportMapFragment)
        getSupportFragmentManager().findFragmentById(R.id.mapae)).getMap();
```

También podremos modificar el tipo del mapa que queremos mostrar (normal, híbrido, satélite o terrestre), o podremos mover libremente nuestro punto de vista por un espacio 3D. Además podremos configurar el movimiento por el mapa mediante la construcción de un objeto *CameraUpdate*, el cual nos proporcionara funciones para el zoom y la actualización de latitud y longitud que queremos mostrar del mapa. Luego lo único que tendremos que hacer es usar el método de nuestro objeto *GoogleMap moveCamera()* pasándole nuestro objeto *CameraUpdate* por parámetro. Esto nos facilita bastante el posicionamiento dentro del mapa al resultar el uso de las clases *CameraUpdate* muy intuitivo. Un ejemplo usado en la aplicación para centrar el mapa en la posición actual del usuario sería el siguiente:

```
CameraUpdate camUpd = CameraUpdateFactory.newLatLngZoom(posActual, 13F);
mapa.moveCamera(camUpd);
```

✓ Un mapa como representación de los recursos

La API de Google Maps nos permite pintar en el mapa cualquier marcador (markers) que queramos, personalizándolo mediante una foto, un título y un subtítulo. En nuestro caso esta es una parte importante del proyecto puesto que mostramos al usuario su localización en el mapa y los recursos más cercanos a esa posición mediante un marcador representativo en cada caso. De esta forma, obtenemos la información de los recursos almacenada en la base de datos del servidor, y creamos una lista de instancias del tipo Recurso, donde ordenamos esos recursos de menor a mayor distancia con el usuario gracias al algoritmo Quicksort del que se hablará más adelante.

Una vez obtenida la lista ordenada le mostramos al usuario los 10 más cercanos para que no se amontonen todos los recursos, puesto que de alguno de ellos hay aproximadamente 1000 y no

se verían con claridad en la pantalla. Para mostrarlos de nuevo nuestro objeto *GoogleMap* nos facilita la tarea poniendo a nuestra disposición el método llamado *addMarker*. Este método recibe un objeto *MarkerOptions* al cual debemos proporcionar las coordenadas geográficas decimales, latitud y longitud, donde queremos que se muestre, el título que nos aparecerá cuando pulsemos sobre él, que será el nombre del recurso en cuestión, y el icono personalizado que hayamos elegido para cada tipo de recurso, ya sea un parque, un punto de suministro, un punto limpio o un aparcamiento. Todos estos datos están contenidos en nuestro objeto de tipo Recurso. Un ejemplo de la aplicación para añadir un marker al mapa del recurso de una lista de recursos sería:

```
MarkerOptions marker=new MarkerOptions().position(
         new LatLng(lista.get(i).getLat(),lista.get(i).getLong()))
          .title(((Parque) lista.get(i)).getNombre())
         .snippet("Pulse para más info")
         .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker));
mapa.addMarker(marker);
```

Y la visualización que el mapa nos ofrecería en la aplicación sería la siguiente, donde el icono rojo representa la posición actual del usuario, y los markers azules la localización del recurso buscado:

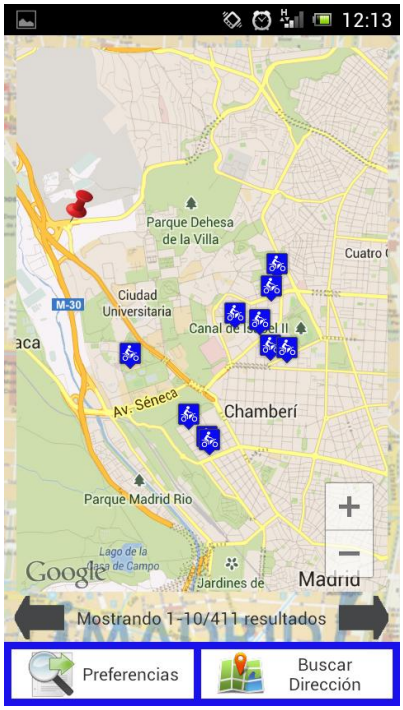


Figura 62. Visualización markers en el mapa según la posición del usuario

También en nuestra aplicación es posible no sólo mostrar los recursos sobre la ubicación del usuario sino también sobre una dirección buscada previamente. Se pintará en el mapa la dirección buscada mediante un marker representativo de un dibujo de una lupa, y se ordenarán los markers de los recursos sobre esta dirección para mostrarlos de la misma forma explicada anteriormente. Esto es posible gracias a una interfaz añadida en la que se pide al usuario que indique el tipo de vía a buscar, ya sea calle, avenida, carretera, paseo o plaza, el

nombre de esa vía y, opcionalmente, el número. Una vez recogidos esos datos se le pasan a un objeto de la clase *Geocoder*, clase proporcionada por Android que se encarga de la geocodificación, y calcula la latitud y longitud de esa dirección buscada que será la que usemos para mostrar en el mapa y ordenar los recursos por cercanía sobre esa posición. Un ejemplo usado en la aplicación para buscar las coordenadas de una dirección específica sería el siguiente, donde *callebuscada* es un String con la dirección, y el número 5 se refiere al número de coincidencias que buscará:

```
Geocoder geocoder = new Geocoder(this);
List<Address> addressList = geocoder.getFromLocationName(callebuscada,5);
```

La visualización que el mapa nos ofrecería en la aplicación sería la siguiente, donde el icono de la lupa representa la dirección buscada por el usuario, y los markers verdes la localización del recurso buscado:

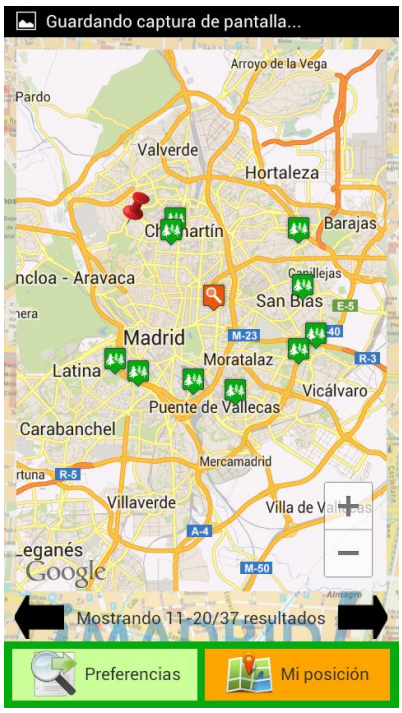


Figura 63. Visualización markers en el mapa según una dirección buscada por el usuario

Dentro del uso del mapa tenemos dos casos especiales para representar las rutas de bicis y el Área de Prioridad Residencial. Vamos a explicar ambos casos de forma individual.

✓ Un mapa para representar áreas

El objeto *GoogleMap* nos ofrece dos opciones más para pintar el Área de Prioridad Residencial o APR. Estas opciones son las de pintar un polígono y pintar una línea por segmentos. En este caso lo primero que hacemos es pintar un polígono que indicará los límites del APR, lo que es posible gracias al método que proporciona el objeto *GoogleMap addPolygon().* Éste recibe por

parámetro un objeto *PolygonOptions* al cual le damos los diferentes vértices del polígono mediante coordenadas geográficas decimales. Entre otras personalizaciones disponibles a la hora de pintar este polígono nosotros hemos usado tres: el grosor de la línea, los colores de las aristas del polígono y el relleno de éste, diferenciándose claramente estos colores. También hemos necesitado la opción de pintar líneas por segmentos en el mapa, pintando las diferentes calles dentro del polígono para indicar las calles por las que se puede circular, gracias al método del objeto *GoogleMap addPolyline* que recibirá un objeto por parámetro *PolylineOption* al cual le proporcionaremos las diferentes coordenadas geográficas de los diferentes segmentos de la línea, el grosor y el color de ésta.

```
mapa.addPolygon(new PolygonOptions()
        .add(new LatLng(40.41419549981287 ,-3.703555163229339),
             new LatLng(40.41282314478989 ,-3.7000039184936213), ……)
        .strokeColor(0xFFFFA500)// color del borde
        .strokeWidth(5)
        .fillColor(0x88FFFF00));//color del relleno
```

Además en el mapa se muestra una leyenda informativa sobre lo que indican estos colores. En el caso de las calles de libre circulación irán en color verde, puesto que son las calles por las que se puede circular con nuestro vehículo, y en el caso del polígono el borde irá en naranja y el interior en amarillo, indicando precaución porque son las calles con acceso restringido. La aplicación muestra el área de esta forma:

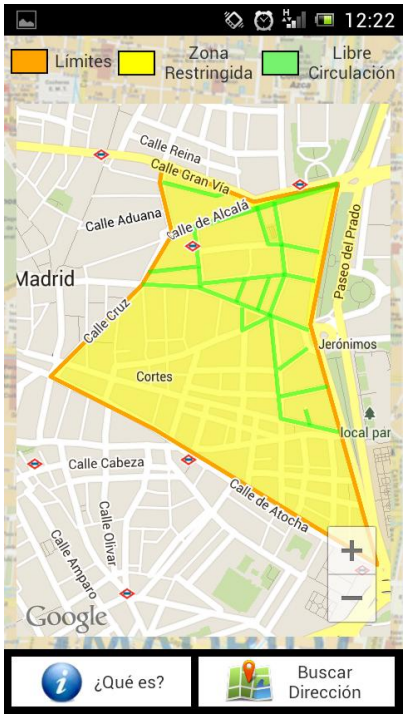


Figura 64. Visualización del área de prioridad residencial

✓ Un mapa para representar rutas de bicis

La información recibida del Ayuntamiento de Madrid sobre el recurso de las rutas de bicis fue en un formato desconocido para nosotros hasta ese momento. Nos proporcionaron los archivos correspondientes a un proyecto realizado con el programa ArcGIS de la compañía ESRI, quien crea y comercializa software para Sistemas de Información Geográfica como Arc/Info o ArcGIS.

ArcGIS es una completa plataforma de información que permite crear, analizar, almacenar y difundir datos, modelos, mapas y globos en 3D, poniéndolos a disposición de todos los usuarios según las necesidades de la organización.

Como sistema de información, ArcGIS es accesible desde clientes desktop, navegadores web, y terminales móviles que se conectan a servidores de departamento, corporativos o con arquitecturas de computación en la nube (Cloud Computing).

En un principio, nos planteamos la opción de añadir la API de ArcGIS Runtime SDK a nuestro proyecto debido a las facilidades que nos daría a la hora de mostrar la información recibida, ya que sólo deberíamos familiarizarnos un poco con la API, añadirla al proyecto, y usarla para mostrar las rutas ya recibidas en ese mismo formato de la compañía.

Pero tras unas pequeñas pruebas con la API de ArcGIS, creando varios mapas, nos dimos cuenta de que los mapas de ArcGIS ocupaban mucho más que los de Google Maps, su tiempo de respuesta era mucho más lento y nos limitaba más los accesos a sus servidores. Nosotros necesitábamos hacer una aplicación de menor peso, más rapidez a la hora de mostrar la información y más simple para el usuario. Además ya usábamos la API de Google Maps para los demás recursos, la cual nos proporciona 25.000 cargas del mapa cada día desde nuestra aplicación, y queríamos que los demás recursos estuvieran en consonancia con el resto.

Una vez descartado el uso de la API de ArcGIS llegó un nuevo reto que fue el de transformar esa información recibida para poder usarla y mostrarla en Google Maps. La información proporcionada era un proyecto en el formato ESRI Shapefile (SHP), un formato de archivo informático propietario de datos espaciales desarrollado por la compañía ESRI.

Un shapefile es un formato vectorial de almacenamiento digital donde se guarda la localización de los elementos geográficos y los atributos asociados a ellos. No obstante carece de capacidad para almacenar información topológica. Es un formato multiarchivo, es decir, está generado por varios ficheros informáticos nombrados a continuación:

- .shp: es el archivo que almacena las entidades geométricas de los objetos.
- .shx: es el archivo que almacena el índice de las entidades geométricas.
- .dbf: es la base de datos donde se almacena la información de los atributos de los objetos.

Además de estos tres archivos requeridos, opcionalmente se pueden utilizar otros para mejorar el funcionamiento en las operaciones de consulta a la base de datos, información sobre la proyección cartográfica o almacenamiento de metadatos. Estos archivos son:

- .prj: es el archivo que guarda la información referida al sistema de coordenadas en formato WKT.
- .sbn y .sbx: almacena el índice espacial de las entidades.
- .fbn y .fbx: almacena el índice espacial de las entidades para los shapefiles que son inalterables (solo lectura).
- .ain y .aih: almacena el índice de atributo de los campos activos en una tabla o el tema de la tabla de atributos.
- .shp.xml: almacena los metadatos del shapefile.

Una vez analizado el tipo de archivo y entendiendo el tipo de información que nos habían proporcionado, nos dispusimos a visualizarla, para lo cual teníamos que usar el programa ArcGIS en una versión de prueba puesto que los trámites para conseguir una licencia eran innecesarios para sólo observar la información. Cuando conseguimos abrir el proyecto, nuestra sorpresa fue que las rutas que nos habían proporcionado estaban desplazadas en el mapa, como puede observarse en la siguiente imagen.

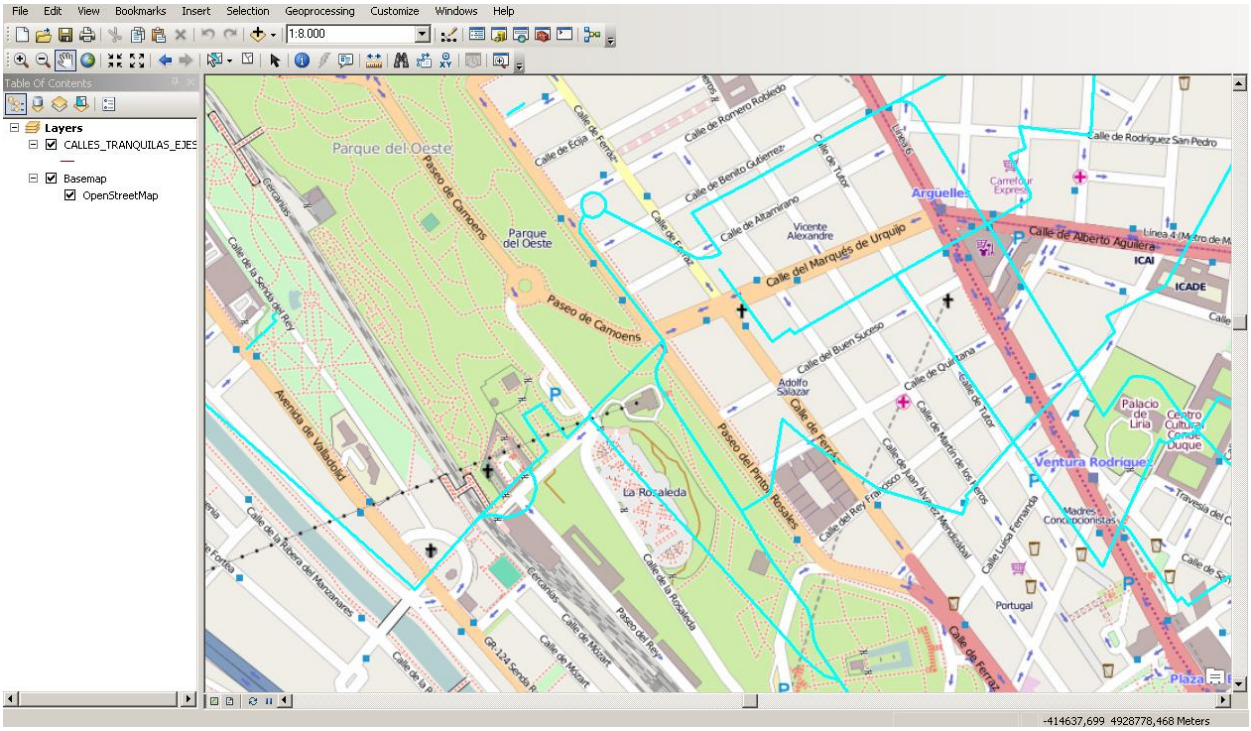


Figura 65. Apertura del proyecto ArcGIS, desplazamiento del mapa

Investigando un poco ArcGIS conseguimos colocar bien las rutas lo cual era algo imprescindible para poder transformar esa información a Google Maps, puesto que ese movimiento afectaría a todas las coordenadas de las rutas y ese error se iría arrastrando en cada paso que fuéramos dando.

Una vez solucionado el problema anterior teníamos que buscar una manera de transformar el shapefile a algún tipo de información que pudiéramos tratar. Tras muchas horas de

investigación conseguimos encontrar una página web (http://shpescape.com/) en la cual su creador (actualmente trabajando como Senior Developer Programs Engineer en Google) nos facilitaba dos opciones: transformar el shapefile a Google Fusion Tables o a GeoJSON and TopoJSON.

Decidimos pues transformarlo a Google Fusion Tables debido a que es un servicio web de Google para la gestión de datos que permite manejar hojas de cálculo, archivos CSV y KML de gran tamaño, al igual que un medio para visualizar los datos con gráficos circulares, gráficos de barras, diagramas de dispersión y líneas de tiempo así como mapas geográficos basados en Google Maps. Esto último fue el gran motivo que nos movió a realizar la conversión a este formato ya que, una vez conseguidas nuestras rutas mostradas en Google Maps, nos sería más fácil acceder a la información que necesitábamos y mostrarlas en nuestra propia aplicación. Puede verse la visualización de estas rutas en la siguiente imagen.

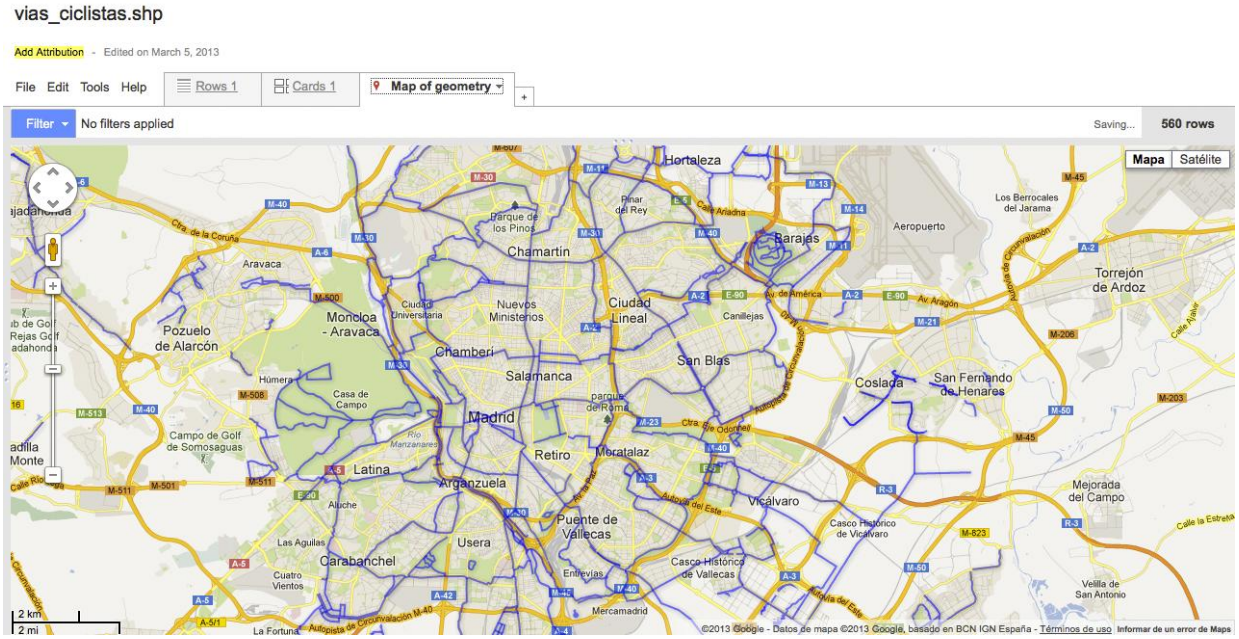
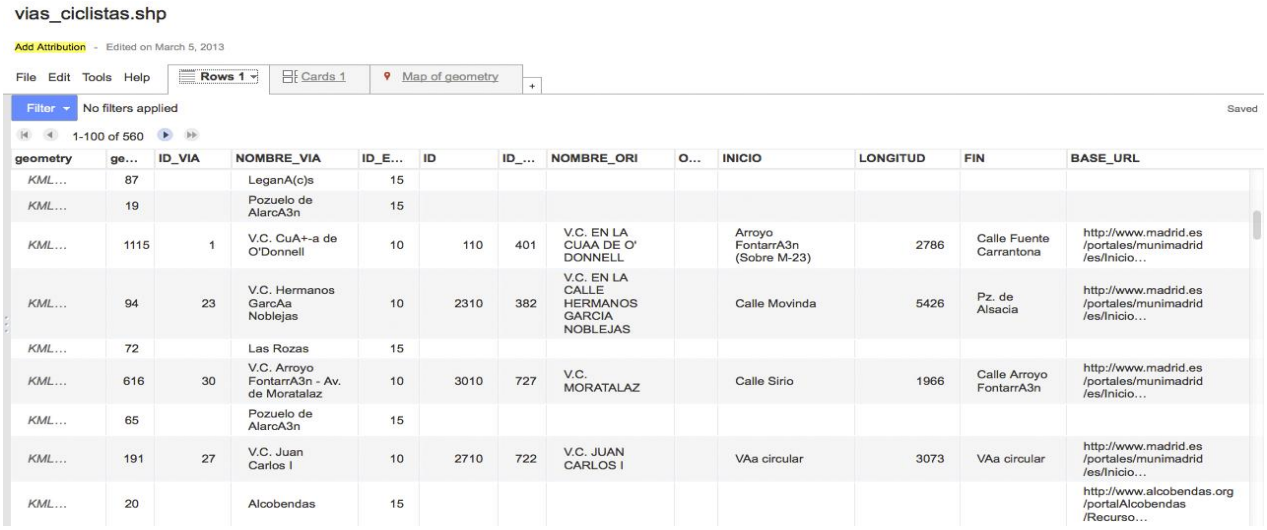


Figura 66. Mapa con las rutas pintadas en Google Maps

Una vez transformado nuestro shapefile a Google Fusion Tables usamos el servicio web que ofrece para transformar nuestros datos (ver figura 67) a un archivo CSV, el cual es un tipo de documento en formato abierto, sencillo para representar datos en forma de tabla en el que las columnas se separan por comas o punto y coma, y las filas por saltos de línea.



vias_ciclistas.shp

Add Attribution - Edited on March 5, 2013

File Edit Tools Help | Rows 1 | Cards 1 | Map of geometry

Filter | No filters applied | Saved

1-100 of 560

geometry	ge...	ID_VIA	NOMBRE_VIA	ID_E...	ID	ID_...	NOMBRE_ORI	O...	INICIO	LONGITUD	FIN	BASE_URL
KML...	87		LeganA(c)s	15								
KML...	19		Pozuelo de AlarcA3n	15								
KML...	1115	1	V.C. CuA+-a de O'Donnell	10	110	401	V.C. EN LA CUAA DE O' DONNELL		Arroyo FontarrA3n (Sobre M-23)	2786	Calle Fuente Carrantona	http://www.madrid.es/portales/munimadrid/es/Inicio...
KML...	94	23	V.C. Hermanos GarcAa Noblejas	10	2310	382	V.C. EN LA CALLE HERMANOS GARCIA NOBLEJAS		Calle Movinda	5426	Pz. de Alsacia	http://www.madrid.es/portales/munimadrid/es/Inicio...
KML...	72		Las Rozas	15								
KML...	616	30	V.C. Arroyo FontarrA3n - Av. de Moratalaz	10	3010	727	V.C. MORATALAZ		Calle Sirio	1966	Calle Arroyo FontarrA3n	http://www.madrid.es/portales/munimadrid/es/Inicio...
KML...	65		Pozuelo de AlarcA3n	15								
KML...	191	27	V.C. Juan Carlos I	10	2710	722	V.C. JUAN CARLOS I		VAa circular	3073	VAa circular	http://www.madrid.es/portales/munimadrid/es/Inicio...
KML...	20		Alcobendas	15								http://www.alcobendas.org/portalAlcobendas/Recurso...

Figura 67. Tabla con los datos de la conversión

Teniendo la información en ese formato ya solo quedaba transformarla a un formato que nosotros pudiéramos tratar en nuestra aplicación.

Cada línea del archivo csv se correspondía con un tramo de la ruta dibujada en el mapa, y dentro de esa línea entre comas se encontraban los diferentes atributos como longitud del tramo de la ruta, nombre de la calle, inicio y fin del tramo, etc... Y la más importante: las coordenadas geográficas decimales de cada punto del tramo de la ruta.

Por fin ya teníamos la información que queríamos, y nos dispusimos a tratarla para meterla en la aplicación. El siguiente paso fue crearnos un pequeño programa en java que leyera nuestro archivo csv (previamente transformado a un archivo de texto sin formato), cogiera los datos que necesitábamos y los escribiera en otro archivo de texto de salida que fue el que finalmente usamos.

Una vez obtenido este fichero de texto, en el cual cada línea tiene las coordenadas geográficas decimales de cada punto del tramo de la ruta, lo único que tuvimos que hacer en la aplicación era recorrer línea a línea, y mediante la API de Google Maps v2 dibujamos una línea en el mapa que se correspondía con cada tramo de la ruta original.

Google Maps nos ofrece una función mediante la cual permite dibujar una línea mediante el método *addPolyline* , es decir, permite pintar una superposición lineal de segmentos de línea conectados en el mapa, a la cual le tenemos que pasar como parámetro un objeto *PolylineOptions* al que añadimos una lista de objetos *LatLng* (proporcionados por Android, los cuales están formados por un par de números reales que representan la latitud y la longitud) que serán las coordenadas de cada punto del segmento de la línea y diferentes opciones referentes al diseño gráfico de esa línea como grosor, color, etc… De esta forma pintamos todos los tramos incluidos en la ruta de bicis. Para ello nos ayudamos de una clase privada llamada *ArrayLatLong*, la cual contiene una lista de objetos *LatLng* que ayuda a almacenar estos tramos de ruta y poder pasárselos fácilmente a la función *addPolyline*.

✓ Clases para la representación de los mapas

Las clases java que usamos en la aplicación para la visualización del mapa gracias a la API de Google Maps son 3: la clase *Mapa* para mostrar los recursos con la ayuda de los markers, la clase *MapaApr* para pintar en el mapa el área de prioridad residencial mediante polígonos, y la clase *MapaBicis* para pintar las rutas de bicicletas.

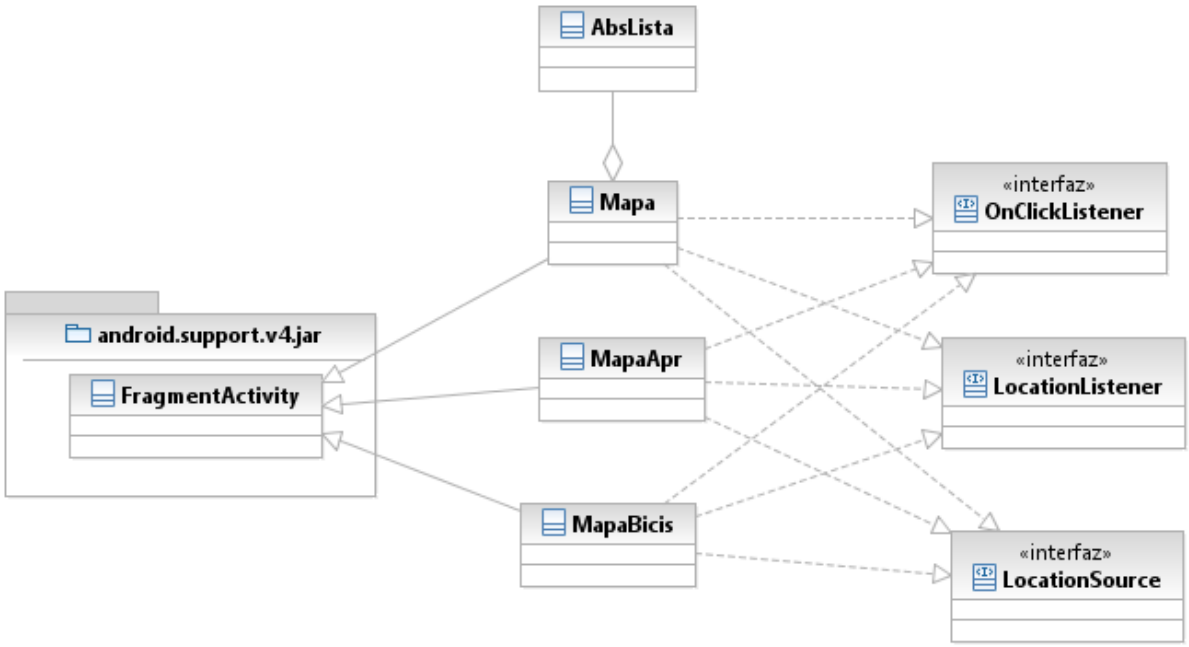


Figura 68. Clases Mapa de la aplicación

Como vemos en la figura 68, las 3 clases mapa extienden la clase *FragmentActivity* para disponer del mapa, y deben además implementar las interfaces de *OnClickListener* para que la aplicación reaccione si el usuario pulsa la pantalla, y las interfaces de *LocationListener* y *LocationSource* para disponer de los métodos necesarios para la localización del usuario. Además vemos que la clase Mapa contiene un atributo de tipo *AbsLista*, ya que recibe una lista de este tipo con todos los recursos que deberá mostrar en el mapa.

✓ Localización del usuario

La localización geográfica en Android es uno de esos servicios que, aunque no requieren mucho código de programación, no es nada intuitivo por lo que fue complicado de manejar correctamente en nuestra aplicación.

En primer lugar hay diferentes formas de obtener la ubicación de un dispositivo móvil, ya sea por GPS, a través de las antenas de telefonía móvil (3G), o mediante puntos de acceso Wi-Fi cercanos. Cada uno de estos modos de obtener la ubicación tiene diferente precisión, velocidad y consumo de recursos distintos. En nuestra aplicación nos decidimos por obtener esa ubicación mediante antenas de telefonía móvil (3G) o puntos de acceso Wi-Fi. Puesto que

es un requisito de nuestra aplicación tener acceso a internet ya sea mediante 3G o Wi-Fi vimos lógico obtener su ubicación mediante estos mecanismos tanto por simplicidad para el usuario de no tener que activar el GPS como por la ventaja de que en interiores de edificios también podrá obtener su localización, algo que con el GPS no es posible.

Como obtendremos la ubicación mediante 3G o Wi-Fi, primeramente comprobamos que el dispositivo móvil tiene conectividad a internet. Para ello hacemos uso de la clase *ConnectivityManager* la cual se encarga del estado de la conectividad de la red obteniendo una instancia de esta clase llamando a *getSystemService(CONNECTIVITY_SERVICE).* Una vez obtenida esta instancia ya podemos acceder a los métodos de esta clase y mediante el método *getActiveNetworkInfo()* podemos comprobar si tenemos conexión a internet. En caso de estar desactivada le mostramos un mensaje al usuario en el que le aconsejamos que la active si quiere continuar en la aplicación. Una vez que hemos comprobado que tenemos conectividad a internet miramos si tiene activa la opción en ajustes para obtener su ubicación por redes. Esto lo hacemos comprobando qué proveedores de localización tiene disponibles el dispositivo o location providers mediante la clase *LocationManager*, clase principal en la que nos basaremos siempre a la hora de obtener la localización en Android. Haciendo una referencia al *LocationManager* llamando al método *getSystemService(LOCATION_SERVICE*) y posteriormente obteniendo si tenemos activada la ubicación por redes gracias a la función *isProviderEnabled(NETWORK_PROVIDER)*, en caso de que ésta esté desactivada mostramos un mensaje al usuario que mediante su aceptación le lleva directamente al apartado de ajustes donde puede activar este servicio.

Una vez comprobado que todas las opciones están activadas, sólo tenemos que usar el método de *LocationManager* llamado *requestLocationUpdates(NETWORK_PROVIDER, 1000, 0, (LocationListener) this).* Este método recibe por parámetro en primer lugar el proveedor al que nos queremos suscribir para recibir las actualizaciones, en segundo lugar el tiempo mínimo en milisegundos entre actualizaciones de la ubicación, en tercer lugar la distancia en metros entre actualizaciones y, por último, la instancia de un objeto de la clase *LocationListener* que tendremos que implementar previamente para definir las acciones a realizar al recibir cada nueva actualización de la posición. En nuestro caso lo configuramos para recibir actualizaciones cada 10 segundos, ignorando los metros que se pueda mover el usuario, y sabiendo que estos parámetros especificados se entienden como simples indicaciones para el proveedor por lo que puede que no se cumplan de forma estricta.

En cuanto al objeto de la clase *LocationListener* contendrá una serie de métodos con los distintos eventos que podemos recibir del proveedor:

- *onLocationChanged(location),* lanzado cada vez que se recibe una actualización de la posición.
- *onProviderDisabled(provider),* lanzado cuando el proveedor se deshabilita.
- *onProviderEnabled(provider*), lanzado cuando el proveedor se habilita.
- *onStatusChanged(provider, status, extras),* lanzado cada vez que el proveedor cambia su estado, que puede variar entre *OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE, AVAILABLE.*

Nosotros usaremos solo *onLocationChanged(location)* en el cual cada vez que recibamos una actualización de la posición se llamará a este método y recibiremos esos cambios por parámetros en un objeto de la clase *Location*. Esta clase de datos representa una localización geográfica la cual tiene, entre otros atributos, la latitud y la longitud que son los que nosotros necesitamos. Cada vez que recibimos una actualización con este método modificamos el marker que corresponde a la ubicación del usuario en el mapa indicado con una chincheta roja.

- ✓ **Quicksort**

Para la ordenación de nuestras listas de recursos por distancia a la localización del usuario utilizamos el algoritmo de ordenación Quicksort, creado por el científico en computación británico Charles Antony Richard Hoare [71]. Utilizamos este algoritmo porque no es difícil de implementar, proporciona buenos resultados generales (funciona bien en una amplia diversidad de situaciones), y en muchos casos consume menos recursos que cualquier otro método de ordenación.

Entre las ventajas de haber usado este algoritmo de ordenación rápida destacan: trabaja in situ (utiliza solo una pequeña pila auxiliar), necesita solamente del orden de N·log N operaciones en promedio para ordenar N elementos y tiene un bucle interno extremadamente corto. Los inconvenientes son que es recursivo (si no se puede utilizar recursión la implementación es complicada), que en el peor caso necesita aproximadamente N^2 operaciones y que es frágil: si durante la implementación pasa inadvertido un simple error, puede causar un mal comportamiento en ciertos archivos.

El quicksort es un método de ordenación que sigue la técnica "divide y vencerás". Funciona dividiendo un archivo en dos partes, y ordenando independientemente cada una de ellas. Hacemos la llamada inicial sobre los parámetros first igual a 0 y last igual a la longitud del array de recursos que vamos a ordenar. El quicksort que utilizamos en nuestra aplicación se muestra a continuación:

```
void Quicksort( int first, int last){
            int i=first, j=last;
            int pivote=((Recurso) array[(first + last) / 2]).getDistancia();
            Recurso auxiliar;
            do{
                 while(((Recurso) array[i]).getDistancia()<pivote) i++;
                 while(((Recurso) array[j]).getDistancia()>pivote) j--;
                 if (i<=j){
                      auxiliar=(Recurso) array[j];
                      array[j]=array[i];
                      array[i]=auxiliar;
                      i++;
                      j--;
                 }
            }
            while (i<=j);
                  if(first<j){
                       Quicksort(first, j);
                  }
```

```
                if(last>i){
                     Quicksort(i, last);
                }
        }
```

- **Fórmula Haversine**

En nuestro proyecto hemos decidido utilizar la fórmula de Haversine para calcular la distancia que hay entre un recurso seleccionado por el usuario y su localización actual. Calcular la distancia entre dos puntos sobre un plano podría llegar a ser relativamente sencillo. Sin embargo, cuando estos dos puntos los ubicamos sobre la esfera terrestre, es decir, lo que pretendemos es calcular la distancia lineal entre dos posiciones dadas, la cosa se complica.

Esto es debido a que en el cálculo de la distancia entre ambas posiciones debemos contemplar la curvatura terrestre. Es aquí donde entra en escena la Fórmula del Haversin, que es:

$$d = R \cdot c$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$a = \sin^2(\Delta lat/2) + \cos(lat1) \cdot \cos(lat2) \cdot \sin^2(\Delta long/2)$$

donde,

R = radio de la Tierra

$\Delta lat = lat2 - lat1$

$\Delta long = long2 - long1$

Para utilizar la Fórmula del Haversine necesitamos, además de las dos posiciones (en formato latitud y longitud), el radio de la Tierra. Este valor es relativo a la latitud, pues al no ser la Tierra perfectamente redonda el valor del radio ecuatorial es de 6378 km mientras que el polar es de 6357 km. El radio equivolumen es de 6371 km. Nosotros utilizamos el radio equivolumen.

Pese a que la Fórmula del Haversine es de las más utilizadas para el cálculo de distancias entre dos puntos tenemos en cuenta que la fórmula asume que la Tierra es completamente redonda, con lo que cabe esperar una tasa de error que se puede asumir.

- **Coordenadas geográficas**

Unos de los principales problemas que tuvimos a la hora de mostrar todos los recursos en el mapa fueron las coordenadas que se nos proporcionaron desde el Ayuntamiento. La información era muy diversa y en algunas ocasiones incompleta. Como hemos explicado anteriormente nosotros necesitábamos las coordenadas en formato geográfico decimal para poder mostrarlas con comodidad y exactitud en el mapa.

De todos los recursos de que consta nuestra aplicación había algunos que no nos dieron ningún tipo de coordenada, sólo nos proporcionaron las direcciones, y en los que nos proporcionaron las coordenadas estaban en otros formatos que para nosotros en ese momento eran desconocidos, como por ejemplo las coordenadas UTM (Sistema de Coordenadas Universal Transversal de Mercator). Por ello decidimos proceder a transformarlas nosotros e intentar calcular las coordenadas de las ubicaciones de las que sólo habíamos recibido su dirección.

Tuvimos que transformar tres tipos de coordenadas: UTM, geográficas en grados sexagesimales, y de una dirección dada. A continuación explicaremos el proceso que conllevó cada tipo de transformación.

El caso de las coordenadas UTM fue el más complejo de todos ya que desconocíamos este formato. Explicaremos un poco la representación cartográfica y porque existen diferentes formatos.

El sistema UTM es un sistema de proyección geodésica ideado en 1569 por Gerhard Kremer, denominado Mercator al latinizar su apellido [72]. Es un sistema en el cual se construye geométricamente el mapa de manera que los meridianos y paralelos se transformen en una red regular, de manera que se conserven los ángulos originales.

La Proyección UTM conserva, por lo tanto, los ángulos pero distorsiona todas las superficies sobre los objetos originales así como las distancias existentes.

El sistema de Proyección UTM tiene las siguientes ventajas frente a otros sistemas de proyección:

- Conserva los ángulos.
- No distorsiona las superficies en grandes magnitudes (por debajo de los 80ª de Latitud).
- Es un sistema que designa un punto o zona de manera concreta y fácil de localizar.
- Es un sistema empleado en todo el mundo, universal, fundamentalmente por su uso militar.

El sistema UTM es un sistema comúnmente utilizado entre los 0º y los 84º de latitud norte y los 80º de latitud sur, por lo que es un sistema estandarizado de empleo en España. Localiza un punto por coordenadas del tipo: X= 462.130, Y= 4.634.140. Únicamente con estos datos el punto no queda definido ya que carece de los siguientes datos:

- Los datos no tienen unidades: ej. metro, kilometro, etc.
- Los datos no localizan el hemisferio donde se encuentra
- Los datos no localizan el Huso UTM de proyección
- Los datos no localizan el Datum (origen del sistema de coordenadas)

Para que el punto quede localizado perfectamente se debe detallar como sigue:

X= 462.130 m, Y= 4.634.140 m, Huso=30, Zona=T, Datum: European 50 (ED50)

La localización de un punto en coordenadas UTM hace necesario, la inclusión del Datum de referencia ya que el no incluirlo trae consigo que, además de producir una indeterminación en la situación geográfica del punto, suceda que en el replanteo de los puntos, el punto replanteado no sea el punto buscado.

Para España la mayor parte de la cartografía perteneciente al "Instituto Geográfico Nacional" Y el "Servicio Cartográfico del Ejercito" se encuentra georreferencida con el "European Datum – 1950", más conocido por sus siglas "ED50". Bajo este Datum se localiza la península ibérica, el archipiélago Balear, y Ceuta y Melilla. Este Datum toma como referencia el "elipsoide Internacional", también llamado "elipsoide de Hayford" con base en Potsdam, Helmertturm, (Alemania). Existen otros Datum de empleo en España, el "European Datum - 79", ("ED79"), y el "European Datum - 87", "ED87" aunque el empleo de ambos esta menos extendido.

Después de esta explicación del formato de coordenadas UTM vamos a aclarar los problemas surgidos con ella y cómo los solucionamos. Una vez que vimos los datos necesarios para calcular estas coordenadas, el Ayuntamiento no nos proporcionó ninguno de esos datos tales como unidades de medida (metro, kilometro, etc.), ni hemisferio o Huso UTM de la proyección, los cuales investigando un poco se pueden obtener. Pero también faltaba uno de los datos más importantes como es el Datum que, como ya hemos visto, es crucial para calcular las coordenadas. Como la petición de las coordenadas al Ayuntamiento no dió sus frutos, nos pusimos a transformarlas con diferentes Datum y a posicionar varios recursos en el mapa con esas coordenadas a ver si coincidían. Finalmente comprobamos que en algunos casos seguían el Datum ED50, y en otros el WGS84.

Para esa transformación de coordenadas nos creamos un proyecto java, en el que gracias a la librería JScience [73] pudimos realizarlo de una forma medianamente sencilla. Recoge métodos para poder ser utilizados en todas las ciencias (matemáticas, física, sociología, biología, economía, etc.). Gracias a esta librería y al método *utmToLatLong.convert* conseguimos convertir las coordenadas UTM en coordenadas geográficas, como se muestra en el siguiente método:

```
public LatLong deUTMaGEO(Double east, Double north){
      UTM utm = UTM.valueOf(30, 'N', east, north,SI.METER);
      CoordinatesConverter<UTM, LatLong> utmToLatLong =
                                  UTM.CRS.getConverterTo(LatLong.CRS);
      LatLong latLong = utmToLatLong.convert(utm);
      return latLong;
}
```

Para los recursos que no nos dieron las coordenadas sino una dirección, creamos un programa que dándole esa dirección obtuviéramos esas coordenadas. Primero juntamos todas esas direcciones en un archivo de texto plano y creamos una aplicación Android que fuera leyendo esos datos y los fuera procesando. Usamos la utilidad *Geocoder*, clase proporcionada por Android que se encarga de la geocodificación, y calcula la latitud y longitud a partir de una dirección. Esta clase es la misma que usamos para obtener las coordenadas de la dirección introducida por el usuario para buscar los recursos más cercanos a ella.

Hubo varios recursos cuyas direcciones estaban mal especificadas, es decir, en vez de darnos un número de la calle y la calle en cuestión, nos proporcionaban una calle que hacía esquina con otra calle. Estos recursos no han podido ser calculados y de momento no están incluidos en la aplicación puesto que son una gran cantidad de datos para hacerlo de forma manual y el método usado para las direcciones bien especificadas no funciona en este caso. El Ayuntamiento de Madrid fue informado de dicha incidencia y de los recursos no incluidos.

- **Paquete *Estructura***

Clases Activity

Dentro de este paquete encontramos las clases que extienden la clase Activity de Android. Habrá una clase de este tipo por cada pantalla incluida en la aplicación. Estas clases pueden verse en la siguiente figura 69, y cómo extienden a esta clase Activity.

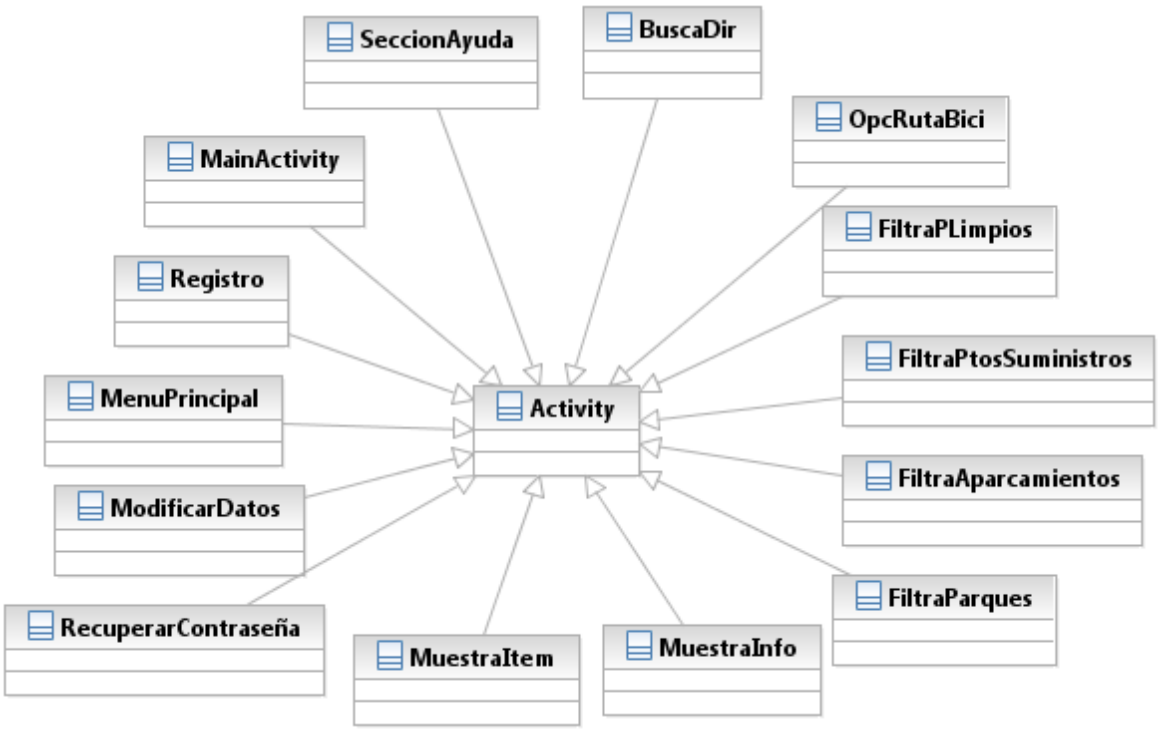


Figura 69. Clases que extienden Activity

Al extender la clase Activity introduce por defecto un método de nombre *onCreate().* Este método se llama para iniciar la actividad. Además la clase Activity surte de otros métodos como *onPause*(), *onStart*(), etc… como ya vimos en el capítulo de tecnologías especiales utilizadas.

Además estas clases deben implementar la interfaz *onClickListener*. De esta forma podemos implementar en estas clases el método *onClick*() y captar cuando el usuario pulsa sobre los diferentes elementos de la pantalla y así darle una funcionalidad en cada caso. En la siguiente figura vemos como estas clases implementan esta interfaz.

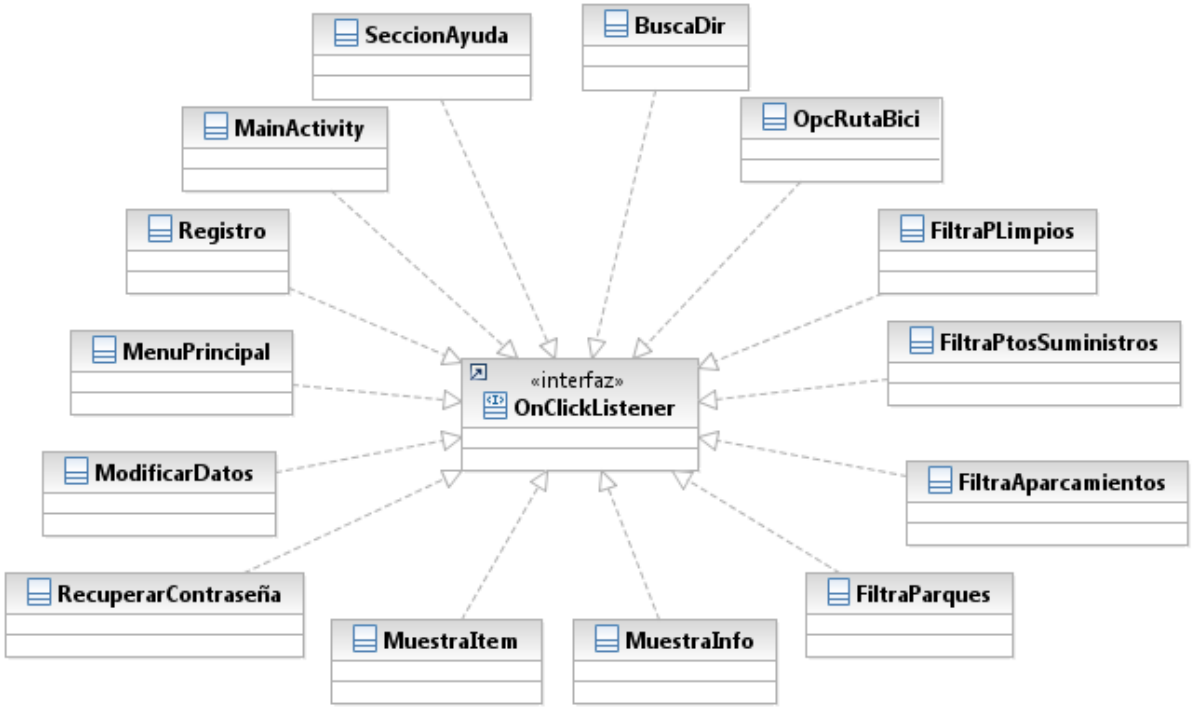


Figura 70. Clases que implementan OnClickListener

Vamos a ir comentando cada una de estas clases y adjuntaremos una captura de la misma, mencionando el archivo xml contenido en la carpeta res/layout que hace posible la visualización de cada actividad gracias al método *setContentView(R.layout.miarchivoXML).* Haremos referencia a las técnicas comentadas anteriormente para la visualización de la información en los mapas, la localización del usuario, el acceso al servidor para obtener información de la base de datos, etc…

Además de estas clases que extienden la clase Activity podemos incluir las clases de Mapa, MapaBicis y MapaApr, aunque en su caso extienden la clase FragmentActivity. Se comentarán también estas clases y el funcionamiento interno que contienen.

❒ Clase MainActivity.java

Esta clase representa la pantalla inicial de la aplicación. El archivo xml asociado a esta clase para la interfaz es main_activity.xml. Como observamos en la imagen ofrece un login al usuario para entrar en la aplicación. Para ello debe estar registrado primero. Si no lo está puede pulsar en *¿Eres nuevo aquí?* lo que le llevará a la pantalla de Registro. Si ha olvidado la contraseña para entrar en la aplicación puede recordarla pulsando sobre el candado, y se le llevará a otra

actividad para que introduzca su email de usuario. También dispone de un botón de información para conocer el uso de la aplicación.

Además el usuario podrá recordar sus datos para no tenerlos que volver a introducir la próxima vez que entre en la aplicación. Para ello esta clase posee un atributo de tipo *SharedPreferences*. Android nos proporciona esta clase para almacenar esta información en las aplicaciones. Cada preferencia se almacenará en forma de clave-valor, es decir, cada una de ellas estará compuesta por un identificador único (en nuestro caso, *user* y *password*) y un valor asociado a dicho identificador (los datos del usuario).

Figura 71. Actividad MainActivity

Figura 72. Actividad Registro

❒ Clase Registro.java

Mediante esta clase el usuario puede registrarse en la aplicación. Los datos que se requieren son un email y una contraseña (que usará más tarde para acceder a la aplicación), y el año de nacimiento y sexo para que el Ayuntamiento de Madrid pueda obtener información acerca del uso de la aplicación. El archivo xml asociado a esta clase para la interfaz es registro.xml.

Primero se realiza una comprobación de que el usuario haya introducido todos los datos. A continuación se realizan otras comprobaciones como son: que los dos campos de la contraseña coincidan, que se haya seleccionado uno de los dos sexos, que el año de nacimiento esté entre el 1913 y el 3000, y que el email se corresponda con un formato válido mediante la siguiente expresión regular, es decir, que se componga de letras o números, seguidos de una '@', más letras y números, un punto '.', y termine con letras:

```
email.matches("[a-zA-Z0-9._-]+@[a-z]+.[a-z]+")
```

Para realizar el alta del usuario se llamará al método altaUsuario de la clase ConectiorServerUsuario, pasándole por parámetro los datos introducidos por el usuario. Si este método devuelve un 0 entonces el alta se ha realizado correctamente y se mostrará al

usuario unos términos que debe aceptar para entrar en la aplicación. En otro caso, si se ha producido algún error al realizar el alta, se le comunicará al usuario.

❒ Clase RecuperarContrasena.java

Si el usuario no recuerda su contraseña para entrar en la aplicación podrá recuperarla introduciendo de nuevo su email, donde recibirá la contraseña que usó en su registro. El archivo xml asociado a esta clase para la interfaz es recuperarcontrasena.xml.

Esta clase hace uso de la clase GMailSender para el envío del email al usuario según se ha explicado anteriormente, llamando al método sendMail. Para obtener la contraseña del usuario se llama al método consultaUsuario de la clase ConectiorServerUsuario, pasándole por parámetro el email del usuario. Si este método devuelve null se avisa al usuario de que se ha producido un error al recuperar sus datos. En otro caso se envía un correo electrónico al email que ha proporcionado el usuario con la contraseña recuperada asociada a ese usuario.

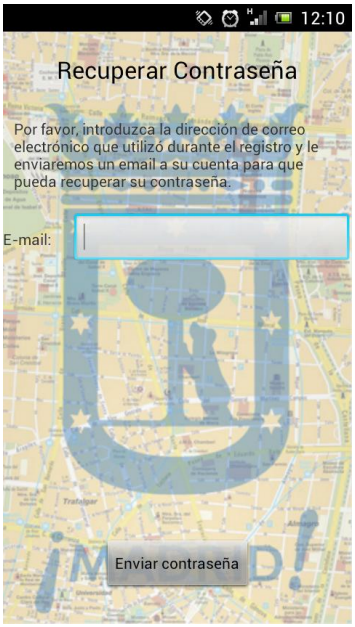


Figura 73. Actividad RecuperarContraseña

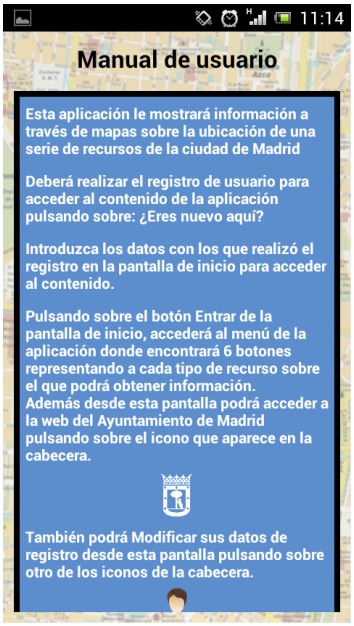


Figura 74. Actividad SeccionAyuda

❒ Clase SeccionAyuda.java

Gracias a esta pantalla el usuario puede conocer información acerca del funcionamiento de la aplicación así como la funcionalidad que le ofrece cada botón de la misma. El archivo xml asociado a esta clase para la interfaz es informacion.xml.

❒ Clase MenuPrincipal.java

Esta clase enlaza todo el funcionamiento de la aplicación, es el menú principal. El archivo xml asociado a esta clase para la interfaz es menu_principal.xml. Dentro de esta clase el usuario puede pulsar sobre el botón de Mis Datos situado en la esquina superior derecha para modificar sus datos de usuario. También puede pulsar sobre el escudo de la Ciudad de Madrid situado en la esquina superior izquierda, para acceder a la web del Ayuntamiento.

En la zona central de la pantalla aparecen 6 botones; cada uno permite realizar una búsqueda del recurso identificado en cada botón. Los botones de Aparcamientos, Puntos Limpios y Puntos de Suministro llevarán primero a un filtrado para que el usuario elija más específicamente lo que desea buscar. Los botones de Parques, Rutas de Bici y Áreas de Prioridad llevarán directamente a la clase Mapa, MapaBicis y MapaApr respectivamente. En cualquier caso se debe localizar la posición del usuario. En esta clase MenuPrincipal comienza los cálculos correspondientes, de forma que evitamos tiempo de espera del usuario hasta que se realice esta acción. Además para el caso de los parques se irá ejecutando el cálculo de distancias desde esa ubicación a la posición de cada recurso, e irá ejecutando también el método Quicksort para ir obteniendo los parques ordenados de menor a mayor distancia con el usuario.

Figura 75. Actividad MenuPrincipal

Figura 76. Actividad ModificarDatos

❒ Clase ModificarDatos.java

Mediante esta clase el usuario podrá modificar sus datos de registro de la aplicación. Como se muestra en la imagen anterior, los cuadros de texto aparecen rellenos con nuestros datos de forma que el usuario pueda decidir si quiere modificar alguno. El archivo xml asociado a esta clase para la interfaz es modificar_datos.xml.

Para realizar la modificación del usuario se llamará al método *modificaUsuario* de la clase *ConectiorServerUsuario*, pasándole por parámetro los datos introducidos por el usuario. Si este método devuelve un 0, entonces la modificación se ha realizado correctamente. En otro caso, si se ha producido algún error al realizar la modificación, se le comunicará al usuario.

❒ Clase MuestraItem.java

Cuando un usuario pulsa sobre algún marker ubicado en el mapa se despliega un bocadillo con información extra del recurso elegido. Además si vuelve a pulsar sobre este bocadillo se le llevará a la clase MuestraItem. El objetivo de esta clase es mostrar toda la información

ampliada del recurso escogido, así como una imagen del mismo. El archivo xml asociado a esta clase para la interfaz es muestra_item.xml.

El objeto del recurso específico se obtiene en la clase Mapa, y ésta se lo envía a esta nueva Activity mediante el uso del Bundle como hemos visto anteriormente.

En las siguientes imágenes podemos observar un par de ejemplos de esta pantalla para el caso de un recurso de tipo aparcamientos de motos, y para un punto de suministro eléctrico.

Figura 77. Actividad MuestraItem

❒ Clase MuestraInfo.java

El funcionamiento de esta clase es parecida a la anterior. El usuario dispondrá de ciertos botones en varias pantallas de la aplicación, donde podrá obtener información asociada a qué es un punto limpio, a los residuos admitidos en un punto limpio fijo, los admitidos en un punto limpio móvil, qué es un área de prioridad residencial, y qué son los combustibles alternativos. Cuando el usuario pulsa sobre alguno de esos botones se abrirá una nueva pantalla asociada a esta activity MuestraInfo, donde su función es simplemente mostrar la información adecuada en cada caso. Para ello la clase que contiene el botón enviará a esta clase mediante un Bundle qué información deberá mostrar. El archivo xml asociado a esta clase para la interfaz es muestra_info.xml.

En las siguientes imágenes podemos observar un par de ejemplos de esta pantalla, para el caso de los residuos admitidos en los puntos limpios fijos, y para conocer qué es un área de prioridad residencial.

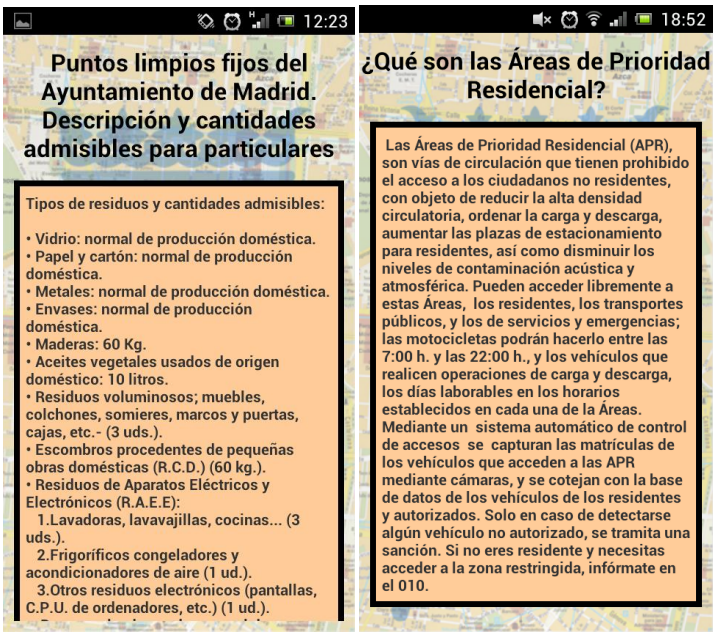


Figura 78. Actividad MuestraInfo

❒ Clase FiltraParques.java

Esta pantalla aparece cuando usuario ha elegido Parques en el menú principal y a continuación ha pulsado sobre el botón de preferencias del mismo. Entonces le aparecen varias opciones que se corresponden con los servicios que pueden disponer los parques, de forma que el usuario puede elegir los que desee y realizar el filtrado. El archivo xml asociado a esta clase para la interfaz es opciones_parques.xml.

Para realizar este filtrado de parques se recogen los servicios seleccionados por el usuario y se llama al método *buscaPorServicios* de la clase *ConectiorServerParque*, pasándole por parámetro esos servicios elegidos. Si este método devuelve null, entonces es que no hay parques que contengan todos esos servicios que el usuario ha elegido, luego se le notificará que la búsqueda realizada no ha obtenido resultados. En otro caso se pasará a mostrarle esos parques en el mapa.

Si el usuario ya ha realizado un filtrado y vuelve a esta pantalla de filtrado de parques, se le da la opción de borrar el filtrado anterior, si es que desea volver a realizar la búsqueda de todos los parques. Para eliminar este filtrado lo que se hace es volver a llamar al método *buscaTodos* de la clase *ConectiorServerParque*, y se vuelven a mostrar en el mapa los parques obtenidos.

Figura 79. Actividad FiltraParques

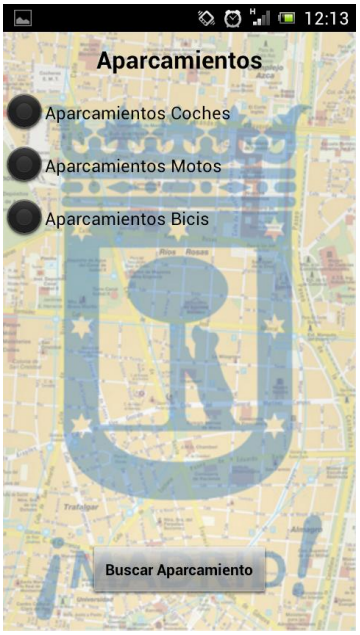


Figura 80. Actividad FiltraAparcamientos

❒ Clase FiltraAparcamientos.java

Esta pantalla aparece cuando usuario ha elegido Aparcamientos en el menú principal. Aparecen 3 opciones que se corresponden con los tipos de aparcamientos que puede buscar, ya sean de coches, de motos o de bicicletas. El archivo xml asociado a esta clase para la interfaz es opciones_aparcamientos.xml.

Para realizar esta búsqueda de aparcamientos se recoge la opción seleccionada por el usuario y en función de eso se llama al método *buscaTodos* de la clase *ConectiorServerParkCoche*, *ConectiorServerParkMoto* o *ConectiorServerParkBici* según corresponda. Los recursos obtenidos se mostrarán en el mapa al usuario.

❒ Clase FiltraPtosSuministros.java

Esta pantalla aparece cuando usuario ha elegido Puntos de Suministro en el menú principal. Aparecen 4 opciones que se corresponden con los tipos de combustibles que puede buscar: bioetanol, glp, gnc o puntos eléctricos. El archivo xml asociado a esta clase para la interfaz es opciones_electrolineras.xml.

Para realizar esta búsqueda de puntos de suministro se recoge la opción seleccionada por el usuario y se llama al método *buscaPorCombustible* de la clase *ConectiorServerPtoSuministro,* pasándole por parámetro el combustible elegido. Los recursos obtenidos se mostrarán en el mapa al usuario.

Figura 81. Actividad FiltraPtosSuministros

Figura 82. Actividad FiltraPLimpios

❒ Clase FiltraPLimpios.java

Esta pantalla aparece cuando usuario ha elegido Puntos Limpios en el menú principal. Aparecen 3 opciones que se corresponden con los puntos limpios fijos, puntos limpios móviles, y contendores de ropa, que el usuario tendrá que seleccionar para realizar la búsqueda. El archivo xml asociado a esta clase para la interfaz es opciones_plimpios.xml.

Para realizar esta búsqueda de puntos limpios se recoge la opción seleccionada por el usuario y en función de eso se llama al método *buscaTodos* de la clase *ConectiorServerPtoLimpioFijo*, *ConectiorServerPtoLimpioMovil* o *ConectiorServerContRopa* según corresponda. Los recursos obtenidos se mostrarán en el mapa al usuario.

❒ Clase OpcRutaBici.java

Esta pantalla aparece cuando el usuario ha elegido Rutas de Bicis en el menú principal. En ella se le da la opción de que elija qué tipos de rutas desea buscar, las vías ciclistas que se corresponden con los carril bici, o las calles seguras que se corresponden con calles con menor tráfico que facilitan la circulación con bicicleta por ellas. Una vez seleccionada la opción, se le mostrará el mapa correspondiente a las rutas de bicis. El archivo xml asociado a esta clase para la interfaz es opc_rutasbici.xml.

Figura 83. Actividad OpcRutasBici

Figura 84. Actividad BuscaDir

❒ Clase BuscaDir.java

Esta pantalla aparece cuando el usuario pulsa el botón de Buscar Dirección ubicado en cualquiera de las clases que representan los mapas. El usuario podrá buscar una dirección concreta para que se le muestre en el mapa. Los campos obligatorios son el de tipo de vía y nombre de la misma. El campo de número de la vía es opcional. Pulsando sobre el botón de Buscar Calle se le mostrará la primera coincidencia que devuelve la clase *Geocoder* según se explicó anteriormente. Si es la que buscaba puede pulsar sobre Ver en mapa; sino puede modificar la búsqueda. El archivo xml asociado a esta clase para la interfaz es busca_direccion.xml.

❒ Clase Mapa.java

Esta clase es común para visualizar los recursos de parques, aparcamientos, puntos limpios y puntos de suministro. La diferencia entre ellas se muestra mediante el color usado en los markers y alrededor de los botones inferiores, usando el verde para parques, turquesa para puntos limpios, azul para aparcamientos, y marrón para puntos de suministro. Los marcadores se mostrarán en el mapa mediante el método *addMarker*() explicado anteriormente. El archivo xml asociado a esta clase para la interfaz es map.xml.

Gracias a la disposición de una lista con los recursos ordenados de menor a mayor distancia con el usuario (previamente ordenada gracias al algoritmo Quicksort), le mostramos al usuario los recursos de 10 en 10, para que no se amontonen en la pantalla ya que de algunos recursos se podrían obtener hasta mil resultados. De esta forma, gracias a las flechas incluidas en la pantalla, el usuario podrá desplazarse por toda la lista y elegir el recurso que más se acomode a sus necesidades.

Mediante el botón de preferencias situado en la esquina inferior izquierda el usuario podrá filtrar la búsqueda de cada recurso llevándole a las pantallas anteriormente comentadas. Mediante el botón de buscar dirección situado en la esquina inferior derecha el usuario podrá introducir una dirección para buscar los recursos más cercanos a partir de esa dirección. Si realizamos esta búsqueda de dirección y volvemos al mapa observamos que el botón de buscar dirección ha cambiado y ahora se llama Mi posición, esto es, para que pulsando sobre él podamos volver a la búsqueda anterior de los recursos por cercanía a la posición del usuario.

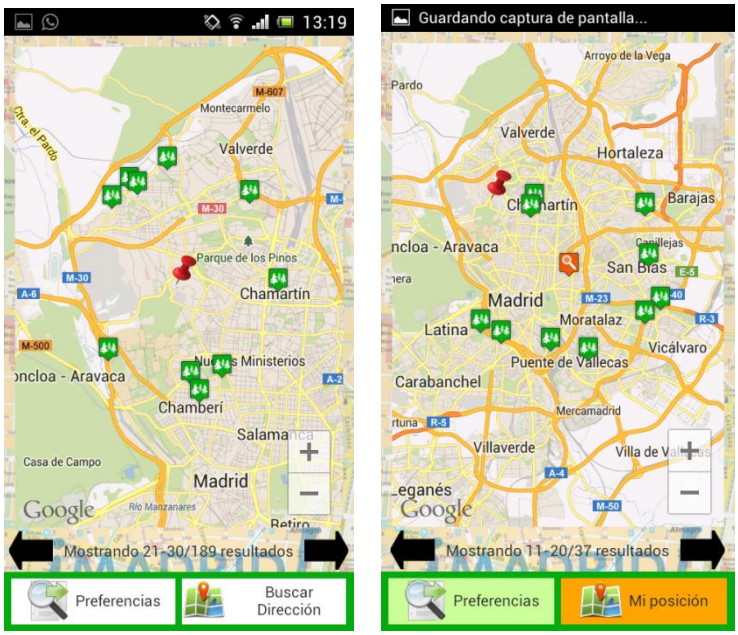


Figura 85. Actividad Mapa

❒ Clase MapaBicis.java

Esta pantalla aparece cuando el usuario eligió Rutas de Bicis en el menú principal y a continuación uno de los dos tipos de vías que se ofrecen. En el mapa se le ofrecen las rutas de bicicletas marcadas en color azul. Además el mapa se le encuadra en su posición actual, pero puede moverse libremente por él, así como aumentar o alejarse del mismo. La visualización de las rutas se realiza de la forma en que se explicó anteriormente. Además de nuevo podrá buscar una dirección para poder observar las rutas de bicis que la rodean. El archivo xml asociado a esta clase para la interfaz es mapbici.xml.

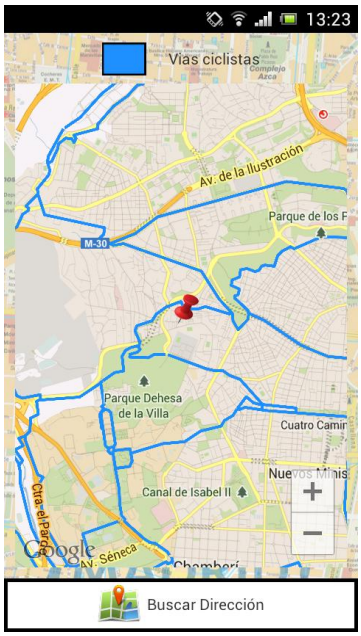


Figura 86. Actividad MapaBicis

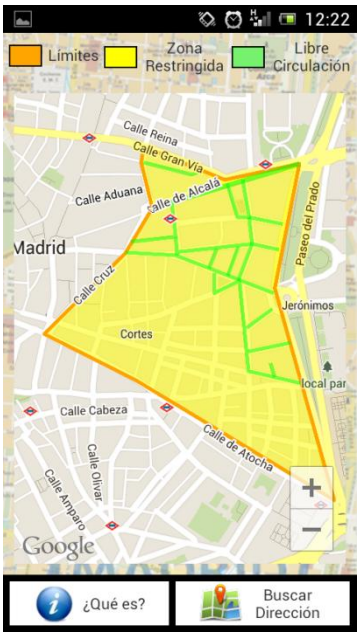


Figura 87. Actividad MapaApr

❒ Clase MapaApr.java

Esta pantalla aparece cuando el usuario eligió Área de Prioridad en el menú principal. En esta clase mapa se le ofrece el área pintado, destacando los límites del mismo, así como las calles de libre circulación o las calles de acceso restringigo. Podrá ver si su localización cae dentro de esta área o por el contrario buscar una dirección para ver si está situada dentro de esta área para preveer su desplazamiento y evitar las calles restringidas al tráfico.

Capítulo 6. MANUAL DE USUARIO

6.1. Uso de la aplicación

❖ Descarga

Actualmente la aplicación puede ser obtenida a través del CD adjunto a esta memoria o bien desde la web www.tecnologiaucm.es. Próximamente la aplicación podrá ser descargada desde el "Play Store" de Android y desde la web del Ayuntamiento de Madrid, www.madrid.es, con el que se ha colaborado para llevar a cabo este proyecto.

Las aplicaciones móviles para el sistema operativo Android tienen la extensión .apk que es una variante del formato JAR de Java y se usa para distribuir e instalar componentes empaquetados para la plataforma Android para smartphones y tablets.

El archivo MapaRecursos.apk que se encuentra disponible en el CD y en la página web anteriormente mencionada debe ser copiado en la tarjeta de memoria del dispositivo móvil, o bien, dentro de la memoria interna del dispositivo. Gracias a un gestor de archivos se podrá acceder a la localización del archivo descargado con extensión .apk y proceder a la instalación del ejecutable como en cualquier otro sistema operativo.

❖ Instalación

Para poder instalar el archivo con extensión .apk en un dispositivo móvil Android se deben seguir los siguientes pasos:

1. Ajustes
2. Seguridad -> En este menú se deberá activar la opción de "Fuentes desconocidas" que permite la instalación de aplicaciones que no proceden del "Play Store" de Android.
3. Buscar el archivo con extensión .apk (MapaRecursos.apk) en el dispositivo a través de un explorador de archivos.
4. Pulsando sobre el archivo, la aplicación comenzará a instalarse en el dispositivo.
   NOTA: Antes de que la aplicación se instale completamente, el dispositivo informará al usuario sobre los permisos que debe aceptar para que la instalación se realice correctamente (Figura 88).

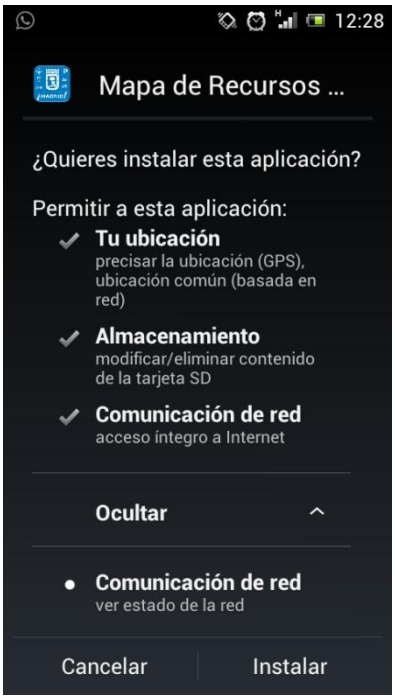


Figura 88. Instalación

5. La aplicación estará instalada en el dispositivo móvil lista para ser utilizada.

❖ Ejecución de la aplicación

Para ejecutar la aplicación en un dispositivo móvil, el usuario sólo tendrá que localizar el icono de la aplicación entre las aplicaciones que ya tenga instaladas en su dispositivo y pulsar sobre él para que la aplicación se inicie.

❖ Controles

El manejo de la aplicación deberá realizarse a través del uso de la pantalla táctil. El usuario podrá manipular los iconos, botones y el teclado presionando sobre ellos en la pantalla del dispositivo móvil. Un simple toque es suficiente.

Además el usuario podrá ampliar o reducir la imagen mostrada sobre un mapa o una página web pulsando dos veces seguidas sobre él. Esto también es posible pellizcando la pantalla con dos dedos para reducir la imagen, o bien alejarla separando los dedos. También podrá desplazarse por el mapa deslizando un dedo sobre él [74].

Para retroceder en cualquiera de las pantallas de la aplicación a una anterior deberá usarse el botón “Volver” del que disponen todos los dispositivos móviles Android.

❖ Requisitos del sistema

El dispositivo móvil Android debe cumplir las siguientes características:

- Sistema Operativo Android 2.3 Gingerbread o versiones posteriores.
- Servicios de ubicación
- Transferencia de datos mediante Wifi o 3G

❖ Manual de usuario

Figura 89. Icono de la aplicación

Al instalar la aplicación sobre un dispositivo móvil Android el usuario encontrará entre sus aplicaciones un icono como el de la Figura 89. Pulsando sobre él, iniciará la aplicación "Mapa de recursos ambientales".

Para el correcto funcionamiento de la aplicación se solicitará al usuario que active los servicios de ubicación y se le informará de que debe tener activada la conexión 3G o Wifi en caso de no tenerlos activados previamente.

- **Pantalla de inicio**

Figura 90. Pantalla inicial

Al iniciar la aplicación aparece la pantalla de la Figura 90. En ella se puede observar la imagen distintiva de la aplicación con todos los elementos que encontraremos en su interior y el escudo del Ayuntamiento de Madrid.

Figura 91. Imagen representativa

Justo debajo de la imagen observamos dos campos de texto para que el usuario introduzca su dirección de correo electrónico y su contraseña para tener acceso a la aplicación.

→ 

A continuación, aparece otro campo que permite al usuario seleccionar la opción "Recordarme". En caso de estar seleccionada, evitará que el usuario tenga que introducir sus datos cada vez que quiera acceder a la aplicación.

→ 

Lo siguiente que se observa en la pantalla es el botón "Entrar" que permite al usuario acceder a la aplicación si ha introducido sus datos correctamente en los campos destinados a la dirección de correo electrónico y la contraseña.

→ 

También aparece una línea de texto con la pregunta "¿Eres nuevo aquí?". Pulsando sobre ella llevará al usuario a otra pantalla para poder realizar el registro de usuario, en caso de que acceda a la aplicación por primera vez.

→ 
El icono de información le servirá al usuario para acceder a una sección de ayuda con un breve manual donde encontrará información relacionada con el manejo de la aplicación.

→ 
Por último se incluye un icono al lado del campo para introducir la contraseña que permite al usuario recuperar su contraseña si no la recuerda.

- **Sección de ayuda**

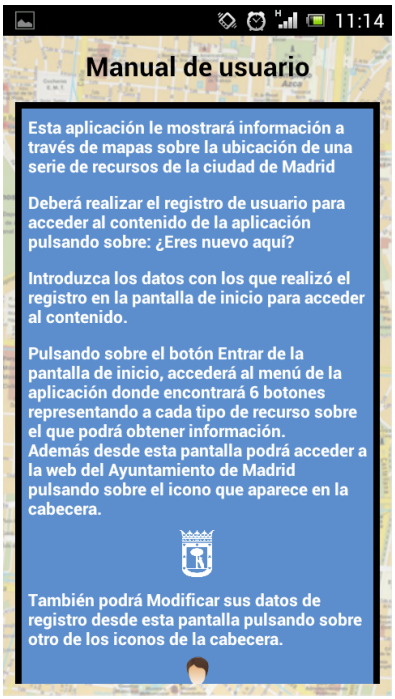


Figura 92. Información

Al pulsar sobre el icono de información de la Pantalla de inicio (Figura 90), el usuario accederá a una pantalla como la de la Figura 92 en la que encontrará la información necesaria para comprender el manejo de la aplicación y conocer los primeros pasos que debe realizar.

- **Registro de Usuario**

Figura 93. Registro de usuario

En el primer acceso a la aplicación, el usuario después de seleccionar "¿Eres nuevo aquí?", accederá a la pantalla de la Figura 93 para realizar el Registro con sus datos.

En ella encontrará varios campos de texto para rellenar:

- → **E-mail**: donde el usuario introducirá su dirección de correo electrónico que luego le servirá para acceder a la aplicación.
- → **Contraseña**: que también será usada para acceder a la aplicación
- → **Repita Contraseña**: donde el usuario deberá introducir el mismo dato que en el campo "Contraseña"
- → **Año de nacimiento**: que servirá para futuras estadísticas sobre el tipo de usuarios de la aplicación.
- → **Sexo**: de igual manera, este dato se utilizará para futuras estadísticas sobre el uso de la aplicación.

Después de rellenar estos campos el usuario deberá pulsar sobre el botón "Registrar" y le aparecerá un mensaje como el de la Figura 94 informándole de los términos legales sobre el uso que se va a hacer de los datos que acaba de introducir.

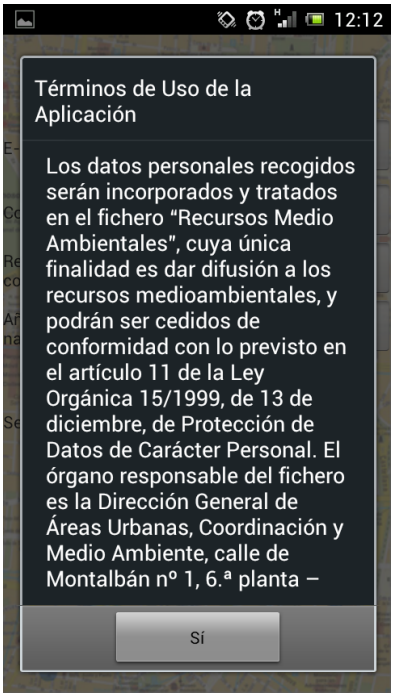


Figura 94. Términos de uso de la aplicación

Al aceptar estos términos el usuario podrá ver un mensaje informándole de que el registro se ha realizado correctamente.

A continuación volverá a la pantalla de inicio para introducir los datos proporcionados en el registro y poder acceder al contenido de la aplicación.

- **Recordar contraseña**

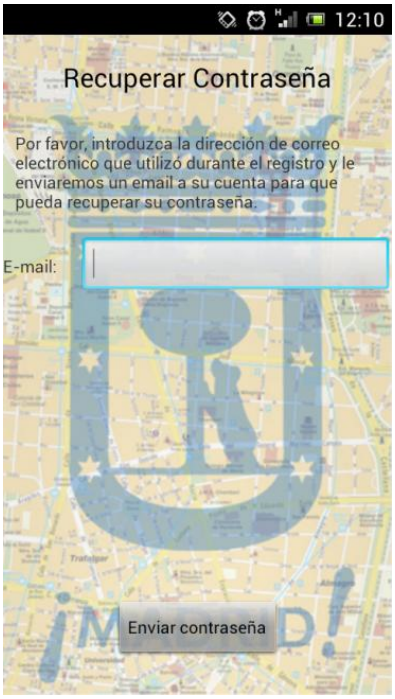


Figura 95. Recuperar contraseña

Pulsando sobre el icono del candado (Recordar Contraseña) de la pantalla de inicio (Figura 90), el usuario accede a la pantalla de la Figura 95, en la cuál se le indica que debe rellenar el campo de texto con la dirección de correo electrónico que proporcionó a la aplicación durante el registro para que se le pueda enviar un email a su dirección con su contraseña.

El usuario recibirá un correo en su bandeja de entrada desde la dirección de email maparecursos2013@gmail.com.

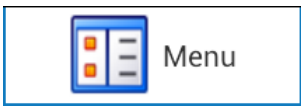
- **Menú principal**

Figura 96. Menú principal

Tras realizar el proceso de registro e introducir los datos de acceso, el usuario encontrará la pantalla de la Figura 96 con los iconos representativos de cada uno de los recursos sobre los que la aplicación ofrece información.

Además, el usuario observa una cabecera con los siguientes botones:

-  El escudo del Ayuntamiento de Madrid, permite al usuario acceder a la web www.madrid.es . En el momento en el que este botón sea pulsado, el usuario podrá seleccionar entre la lista de navegadores que tenga instalados con cuál de ellos desea abrir la página web.

-  Este botón de "Menú" aparece seleccionado ya que permite al usuario volver a la pantalla del menú principal y en este momento es en la que nos encontramos.

-  El botón "Mis Datos" permite al usuario acceder a una pantalla como la de la Figura 97 y modificar cualquiera de los datos que introdujo durante el proceso de registro. Además podrá volver al menú principal después de modificarlos a través del botón "Menú" de la cabecera.

Figura 97. Modificar datos

Desde cualquier pantalla de la aplicación el usuario podrá pulsar el botón de retroceso de Android para volver a una pantalla anterior. Si este botón es pulsado desde el menú principal, el usuario recibirá una notificación como la de la Figura 98 para que confirme si realmente desea salir de la aplicación.

Figura 98. Salir de la aplicación

6.2. Parques

Figura 99. Icono Parques

Pulsando sobre este icono el usuario tendrá acceso a toda la información relacionada con los parques de Madrid.

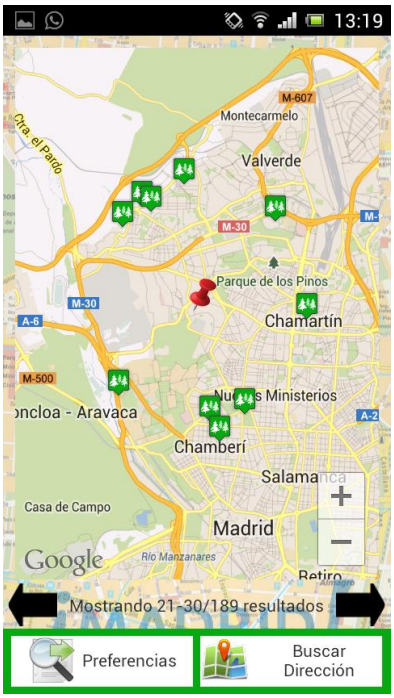


Figura 100. Mapa de parques

En primer lugar se encontrará con la pantalla de la Figura 100, en la que se le muestra información a través de un mapa.

-  Este icono rojo representa la posición geolocalizada del usuario.

-  Los 10 markers de este tipo representan los 10 parques más cercanos a la posición del usuario.

-  Debajo del mapa encontrará dos flechas que le permitirán obtener posiciones de parques más lejanos a la posición del usuario o volver a

los más cercanos en intervalos de 10, ya que los parques estarán ordenados por cercanía al usuario.

- 

Un mensaje entre las flechas le informa del intervalo que se está mostrando en cada momento.

Además existen dos botones en la parte inferior de la pantalla que ofrecen al usuario más opciones para obtener información sobre los parques.

- 

Pulsando este botón el usuario puede acceder a la pantalla de la Figura 101 en la que se le muestra un listado de los servicios que puede tener un parque.

Figura 101. Servicios que ofrecen los parques

En ella el usuario puede seleccionar aquellos servicios en los que está interesado para realizar una nueva búsqueda de parques que contengan esos servicios, y obtener la información a través del mapa nuevamente pulsando sobre el botón “Buscar Parques”. El resultado obtenido al hacer una búsqueda filtrada presenta el aspecto de la Figura 102.

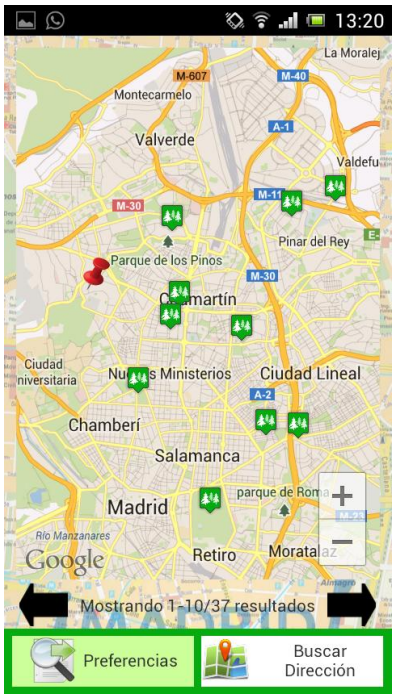


Figura 102. Mapa de parques aplicando preferencias

Observamos que el botón de Preferencias se ha coloreado de verde para que el usuario sepa que tiene un filtro aplicado. Pulsando nuevamente sobre el botón preferencias regresará a la pantalla de la Figura 101 pudiendo modificar el filtro haciendo una nueva búsqueda, o bien eliminar el filtro que tenía aplicado y obtener nuevamente todos los parques gracias al botón "Eliminar Filtro".

- 

Si el usuario decide pulsar la opción de "Buscar Dirección" le aparecerá una pantalla como la de la Figura 103.

Figura 103. Buscar dirección

En esta pantalla el usuario encuentra un selector con los distintos tipos de vía para realizar la búsqueda (Calle, Avenida, Carretera, Paseo y Plaza), un campo de texto en el que podrá introducir el nombre de la vía que desea buscar y otro campo de texto en el que podrá introducir el número de la vía.

Después deberá pulsar el botón de "Buscar calle" y en el campo de texto "Resultado" se le mostrará la vía que se ha localizado a partir de los datos introducidos para que el usuario pueda comprobar si es la vía que realmente desea buscar o modifique alguno de los datos introducidos.

-  Si el resultado obtenido es el deseado, el usuario sólo tendrá que pulsar sobre el botón "Ver en mapa" y se le mostrará una pantalla como la de la Figura 104 en la que encontrará un nuevo mapa que muestra información sobre los parques a partir de la dirección introducida.

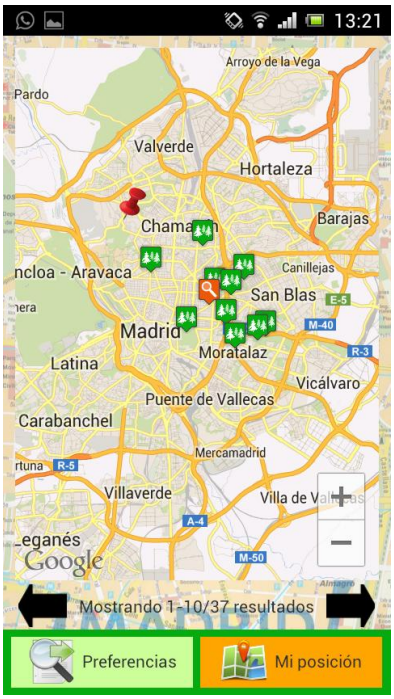


Figura 104. Mapa con dirección buscada

En esta pantalla, el usuario puede observar el icono que muestra su posición y los markers que señalizan la posición de los parques localizados.

-  Además puede observar un nuevo marker como éste que indica dónde se encuentra la dirección que introdujo para realizar la búsqueda.

  En esta ocasión, los parques que se muestran están ordenados por cercanía a la dirección introducida por el usuario. También podrá desplazarse por ellos obteniendo otros más alejados en intervalos de 10.

- 

El botón de “Buscar dirección” en esta pantalla ha sido sustituido por el de “Mi posición”. Pulsando en él, el usuario podrá volver nuevamente a la pantalla de la Figura 100 en la que los parques aparecían ordenados por cercanía a su posición.

Desde el mapa, el usuario podrá pulsar sobre cualquier recurso de los que se muestran y obtener la distancia a la que se encuentra. Además al pulsar sobre él aparecerá un mensaje con el nombre del parque que se encuentra en esa posición como en la Figura 105.

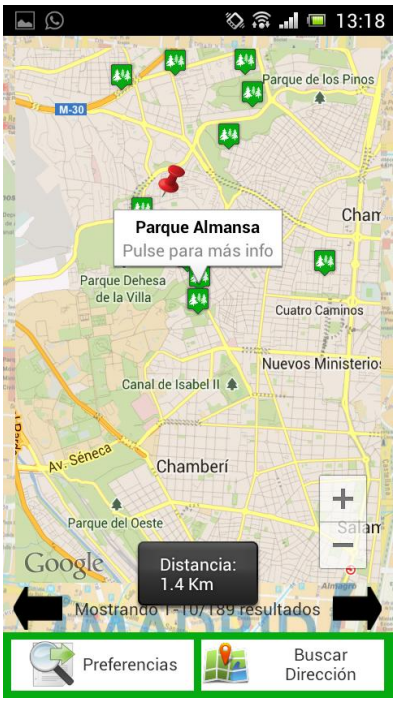


Figura 105. Nombre del parque seleccionado

El usuario podrá pulsar sobre el nombre del parque para obtener más información, que se le mostrará en una pantalla como la de la Figura 106. En ella aparece la superficie del parque, el horario y un listado de los servicios que ofrece el parque.

Figura 106. Información ampliada

6.3. Aparcamientos

Figura 107. Icono Aparcamientos

Pulsando sobre este icono el usuario tendrá acceso a toda la información relacionada con los aparcamientos de Madrid.

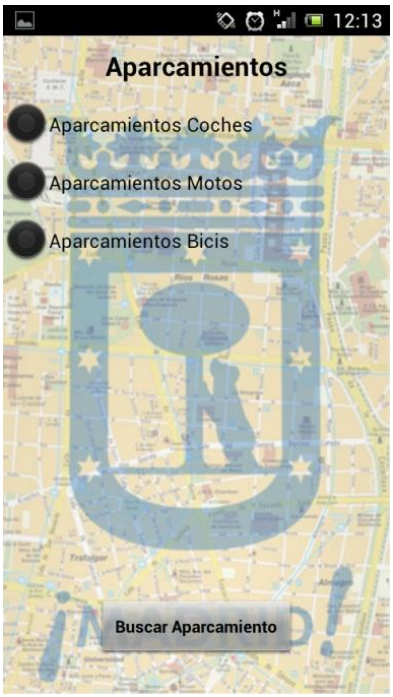


Figura 108. Opciones de aparcamientos

Al seleccionar este recurso lo primero que el usuario encuentra es la pantalla de la Figura 108, donde se le da la opción de elegir entre tres tipos de aparcamientos según el vehículo que se desea estacionar.

- Aparcamientos para coches
- Aparcamientos para motos
- Aparcamientos para bicicletas

El usuario deberá seleccionar una de las tres opciones y pulsar sobre el botón “Buscar Aparcamientos”. En ese momento accederá a una pantalla como las de la Figura 109 en la que se le mostrarán los aparcamientos del tipo seleccionado.

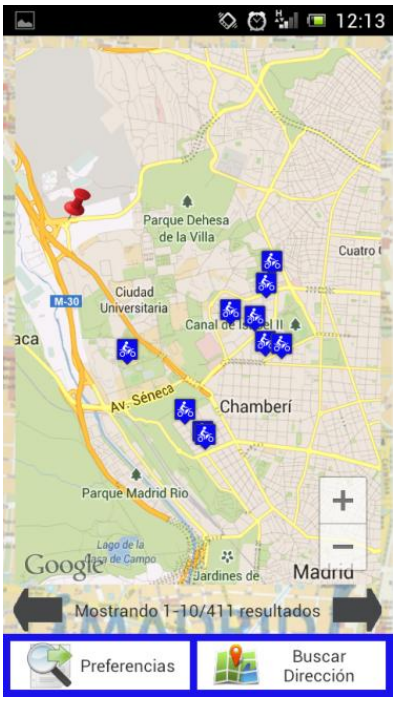


Figura 109. Mapa de aparcamientos de moto

Para distinguir los tipos de aparcamientos en el mapa se han utilizado 3 markers diferentes:

-  Este marker señalizará las ubicaciones de los aparcamientos para coches.

-  Este marker señalizará las ubicaciones de los aparcamientos para motos.

-  Este marker señalizará las ubicaciones de los aparcamientos para bicicletas.

Dentro de la pantalla de la Figura 109, el usuario podrá realizar las mismas opciones descritas en el apartado 6.2 Parques. Podrá desplazarse por los recursos ordenados por cercanía respecto a su posición geográfica a través de las flechas, observando entre ellas el mensaje que indica el intervalo de recursos que está siendo mostrado. Además en la parte inferior de la pantalla también encontrará los dos botones que le permitirán gestionar sus preferencias respecto a los recursos mostrados:

- 

Al pulsar este botón, el usuario podrá volver a la pantalla de opciones de la Figura 108 en la que se le permitirá cambiar nuevamente el tipo de aparcamiento que desea localizar en función del tipo de vehículo.

- 

Pulsando sobre este botón, el usuario podrá obtener los aparcamientos del tipo que haya elegido ordenados por cercanía a la nueva dirección dada, siguiendo el mismo proceso descrito en el apartado 6.2 Parques para este botón.

Además, el usuario podrá pulsar sobre cualquiera de los markers que aparezcan situados en el mapa para obtener su distancia hasta él y un mensaje que le permitirá ampliar la información sobre el aparcamiento seleccionado en una pantalla como la de la Figura 110.

Figura 110. Información ampliada

En esta pantalla se proporciona información detallada sobre el aparcamiento al usuario.

En el caso de los aparcamientos de moto se muestra la situación del aparcamiento, el tipo de aparcamiento, las medidas y su capacidad.

La información ampliada será similar en el caso de los aparcamientos para bicicletas y para coches.

6.4. Puntos Limpios

Figura 111. Icono Puntos Limpios

Pulsando sobre este icono el usuario tendrá acceso a toda la información relacionada con los puntos limpios de Madrid.

Figura 112. Opciones de Puntos Limpios

Al seleccionar este recurso lo primero que el usuario encuentra es la pantalla de la Figura 112, donde se le da la opción de elegir entre tres puntos limpios diferentes:

- Puntos Limpios Fijos
- Puntos Limpios Móviles
- Contenedores de ropa

Además en esta pantalla se ofrece información adicional sobre los Puntos Limpios:

-  ¿Qué es un Punto Limpio?: Pulsando sobre este botón el usuario podrá encontrar una breve descripción sobre lo que son los puntos limpios tanto fijos como móviles.

-  Residuos Puntos Limpios Fijos: Si el usuario desea saber qué residuos se admiten en los Puntos Limpios Fijos, obtendrá la información necesaria pulsando sobre este botón y se le mostrará en una pantalla como la de la Figura 113.

-  Residuos Puntos Limpios Móviles: Si el usuario desea saber qué residuos se admiten en los Puntos Limpios Móviles podrá obtener la información necesaria pulsando sobre este botón y se le mostrará en una pantalla como la de la Figura 113.

Figura 113. Información sobre residuos

El usuario deberá seleccionar una de las tres opciones y pulsar sobre el botón “Buscar Punto Limpio”. En ese momento accederá a una pantalla como las de las Figura 114 en la que se le mostrarán los puntos limpios del tipo seleccionado.

Figura 114. Mapa de Puntos Limpios

Para distinguir los tipos de puntos limpios sobre el mapa, se han utilizado 2 markers diferentes:

-  Este marker señalizará las ubicaciones de los puntos limpios tanto fijos como móviles.

-  Este marker señalizará las ubicaciones de los contenedores de ropa.

Dentro de la pantalla de la Figura 114 el usuario podrá realizar las mismas opciones descritas en el apartado 6.2 Parques. Podrá desplazarse por los recursos ordenados por cercanía respecto a su posición geográfica a través de las flechas, observando entre ellas el mensaje que indica el intervalo de recursos que está siendo mostrado. Además, en la parte inferior de la pantalla también encontrará los dos botones que le permitirán gestionar sus preferencias respecto a los recursos mostrados:

- 

Al pulsar este botón, el usuario podrá volver a la pantalla de opciones de la Figura 112 en la que se le permitirá cambiar nuevamente el tipo de punto limpio que desea localizar en función del tipo de sus intereses.

- 

Pulsando sobre este botón, el usuario podrá obtener los puntos limpios del tipo que haya elegido ordenados por cercanía a la nueva dirección dada, siguiendo el mismo proceso descrito en el apartado 6.2 Parques para este botón.

Además el usuario podrá pulsar sobre cualquiera de los markers que aparezcan situados en el mapa para obtener su distancia hasta él y un mensaje que le permitirá ampliar la información sobre el punto limpio seleccionado en una pantalla como la de la Figura 115.

Figura 115. Información ampliada

En esta pantalla se proporciona información detallada sobre el punto limpio al usuario.

En el caso de los puntos limpios fijos se muestra: la dirección del punto limpio, el distrito, el horario, cómo llegar hasta el punto limpio, alguna observación y un apartado con más información sobre el punto limpio.

En el caso de los puntos limpios móviles se muestra la dirección del punto limpio, el distrito, los días de parada, el horario y los días exentos de parada.

En el caso de los contenedores de ropa se muestra la dirección y el distrito donde se encuentra el contenedor de ropa.

6.5. Puntos de Suministro

Figura 116. Icono Suministros

Pulsando sobre este icono el usuario tendrá acceso a toda la información relacionada con los puntos de suministro de combustibles ecológicos de Madrid.

Figura 117. Opciones de Suministros

Al seleccionar este recurso lo primero que el usuario encuentra es la pantalla de la Figura 117, donde se le da la opción de elegir entre cuatro tipos de combustibles.

- Bioetanol
- GLP (Gas Licuado de Petróleo)
- GNC (Gas Natural Comprimido)
- Electrolineras

Además en esta pantalla se ofrece información adicional sobre los Puntos de Suministro:

-  ¿Qué son los combustibles alternativos?: Pulsando sobre este botón el usuario podrá encontrar una descripción sobre qué son los combustibles alternativos.

Para localizar los recursos el usuario deberá seleccionar una de las cuatro opciones y pulsar sobre el botón “Buscar Puntos de Suministro”. En ese momento accederá a una pantalla como la de la Figura 118 en la que se le mostrarán los puntos de suministro del combustible seleccionado.

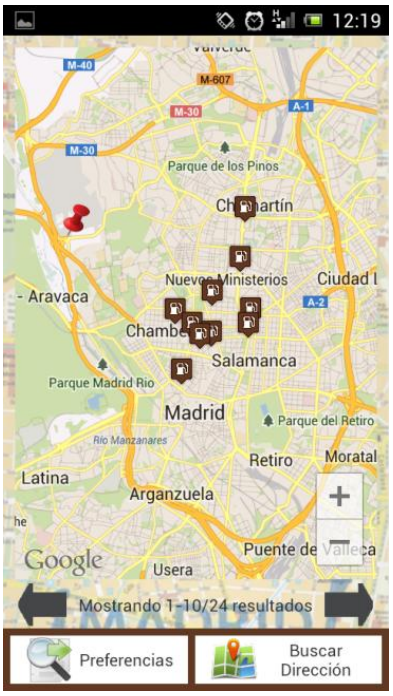


Figura 118. Mapa de Puntos de Suministro

-  Este marker señaliza las ubicaciones de los puntos de suministro para cualquiera de los cuatro tipos de combustible.

Dentro de la pantalla de las Figura 118 el usuario podrá realizar las mismas opciones descritas en el apartado 6.2 Parques. Podrá desplazarse por los recursos ordenados por cercanía respecto a su posición geográfica a través de las flechas, observando entre ellas el mensaje que indica el intervalo de recursos que está siendo mostrado. Además en la parte inferior de la pantalla también encontrará los dos botones que le permitirán gestionar sus preferencias respecto a los recursos mostrados:

- 

Al pulsar este botón el usuario podrá volver a la pantalla de opciones de la Figura 117 en la que se le permitirá cambiar nuevamente el tipo de combustible para el cual desea localizar los puntos de suministro.

- 

Pulsando sobre este botón el usuario podrá obtener los puntos de suministro del tipo que haya elegido ordenados por cercanía a la nueva dirección dada, siguiendo el mismo proceso descrito en el apartado 6.2 Parques para este botón.

Además el usuario podrá pulsar sobre cualquiera de los markers que aparezcan situados en el mapa para obtener su distancia hasta él y un mensaje que le permitirá ampliar la información sobre el punto de suministro seleccionado en una pantalla como la de la Figura 119.

Figura 119. Información ampliada.

En esta pantalla se proporciona información detallada sobre el punto de suministro.

En el caso del bioetanol, GNC y GLP se muestra la ubicación del punto de suministro, el distrito, el tipo de combustible y el comercializador.

En el caso de las electrolineras se informa sobre la ubicación del punto de suministro, el distrito, el tipo de combustible, algunas observaciones y el comercializador.

6.6. Rutas en bici

Figura 120. Icono Rutas Bici

Pulsando sobre este icono el usuario tendrá acceso a toda la información relacionada con las rutas en bicicleta de Madrid.

Figura 121. Opciones Rutas en Bici

Al seleccionar este recurso lo primero que el usuario encuentra es la pantalla de la Figura 121, donde se le da la opción de elegir entre dos tipos de rutas en bici.

-  Vías ciclistas: Seleccionando este tipo de ruta, el usuario obtendrá un mapa como el de la Figura 122, en el que aparecen señalados cada uno de los carriles bici que existen en la ciudad de Madrid.

-  Calles seguras: Si el usuario pulsa sobre este botón se le proporcionará información sobre las calles que han sido clasificadas como seguras por suponer un menor riesgo para el ciclista. Estas calles aparecerán señaladas en un mapa similar al de la Figura 122.

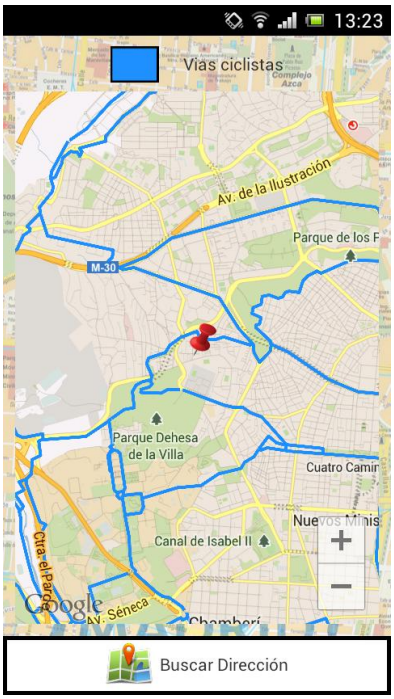
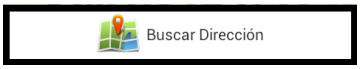


Figura 122. Mapa de Rutas en Bici

Dentro de la pantalla de la Figura 122 el usuario podrá ver señaladas en color azul cada una de las rutas ciclistas según el tipo que haya seleccionado y el marker representativo de su posición.

Además en la parte superior de la pantalla el usuario puede observar una leyenda con lo que representa cada una de las vías señaladas.

- 

En la parte inferior de la pantalla el usuario puede observar nuevamente el botón “Buscar dirección”, que le permitirá situar el marker destinado a la señalización de direcciones y observar las rutas ciclistas que se encuentran cercanas a esa dirección, siguiendo el mismo proceso descrito en el apartado 6.2 Parques para este botón.

Además el usuario podrá pulsar sobre el icono de su posición y obtener la distancia a la que se encuentra de la dirección buscada como se muestra en la pantalla de la Figura 123.

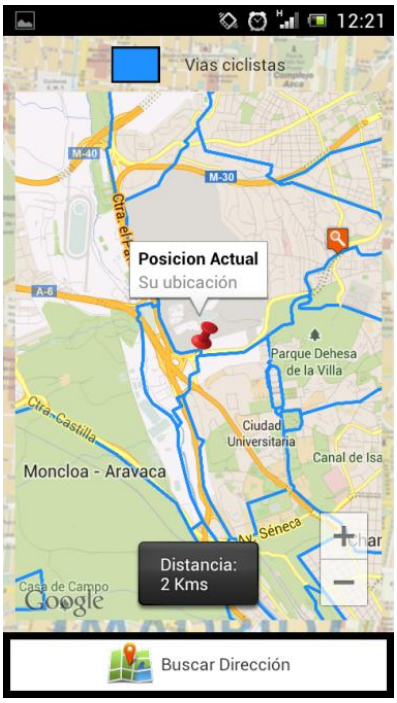


Figura 123. Mapa buscando una dirección

6.7. APR – Área de Prioridad Residencial

Figura 124. Icono APR

Pulsando sobre este icono el usuario tendrá acceso a toda la información relacionada con las Áreas de Prioridad Residencial de Madrid.

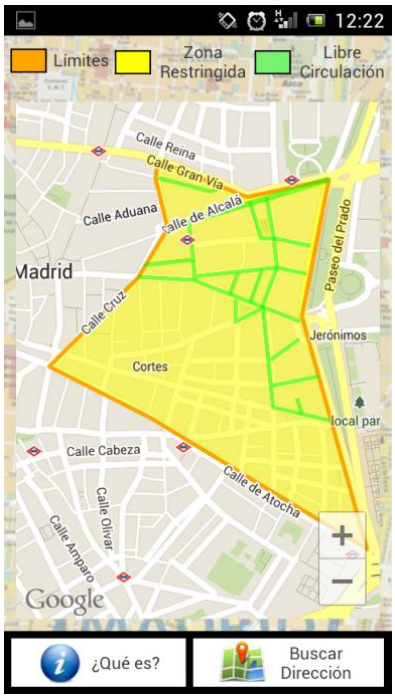


Figura 125. Mapa APR

Al seleccionar este recurso el usuario encuentra una pantalla como la de la Figura 125, donde aparece el área de prioridad residencial delimitada en un mapa.

En la parte superior de la pantalla se puede observar una leyenda sobre los colores utilizados para señalar los datos de interés sobre la zona y lo que representan.

- Límites: Las calles que aparecen señaladas en el mapa con este color son aquellas que comprenden los límites de la zona que se rige bajo las normas de Área de Prioridad Residencial.

- Zona Restringida: La zona señalada en este color señaliza las vías de circulación en las que tiene prohibido el acceso cualquier ciudadano no residente.

- Libre Circulación: Las vías de circulación que han sido señaladas en este color son aquellas en las que tanto los ciudadanos residentes como no residentes tienen libre acceso.

Dentro de la pantalla de la Figura 125 el usuario puede observar en la parte inferior de la pantalla dos botones que le ofrecen opciones sobre este recurso:

- 

Al pulsar este botón el usuario obtiene una breve descripción sobre lo que es un Área de Prioridad Residencial y cuáles son las normas que rigen sobre ellas.

- 

Pulsando sobre este botón el usuario podrá situar sobre el mapa el marker representativo para la búsqueda de direcciones (Figura 126) y comprobar si la vía buscada tiene libre el acceso o por el contrario está restringido, siguiendo el mismo proceso descrito en el apartado 6.2 Parques para este botón.

Además el usuario podrá pulsar sobre el icono de su posición y obtener la distancia a la que se encuentra de la dirección buscada.

Figura 126. Mapa APR buscando una dirección

6.8. Ejemplo de uso

Supongamos que nos encontramos en nuestra casa y queremos conocer un parque nuevo para ir con nuestra familia que disponga de Zona infantil, Zona de mayores y Aseos públicos.

Para ello vamos a entrar en la aplicación “Mapa de Recursos Ambientales” con nuestros datos.

1.- Introducimos la dirección de correo electrónico utilizada durante el registro, la contraseña y pulsamos sobre el botón “Entrar”

2.- En el menú principal pulsamos sobre el icono de “Parques”, accedemos al mapa y pulsamos sobre el botón preferencias.

3.- A continuación seleccionamos Zona infantil, Zona de mayores y Aseos Públicos y pulsamos el botón “Buscar”.

4.- La aplicación ha localizado 5 parques diferentes, entre ellos el Parque del Retiro que parece una buena opción.

5.- Podemos comprobar información adicional sobre otros servicios que ofrece este parque pulsando sobre él y ya tenemos elegido nuestro parque (Figura 127).

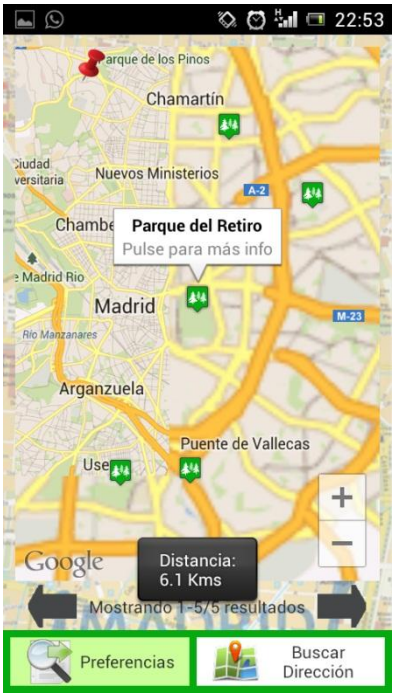


Figura 127. Mapa Parque del Retiro

Además supongamos que tenemos previsto desplazarnos en coche y mientras planeamos la ruta nos surge la duda de si hemos escogido un buen recorrido porque no sabemos si algunas calles pueden pertenecer a un Área de Prioridad Residencial.

6.- Pulsando el botón “Volver” de Android hasta regresar al menú principal y seleccionamos el icono APR

7.- Pulsamos sobre “Buscar dirección” e introducimos la calle sobre la que tenemos duda, en este caso pondremos la Calle Huertas

8.- Pulsando sobre el icono “Ver en Mapa” la aplicación nos muestra un marker sobre la calle introducida y comprobamos en la leyenda que aparece en la parte superior de la pantalla que efectivamente esta calle pertenece a la zona restringida y por tanto no podremos circular por ella (Figura 128).

Figura 128. Mapa APR con dirección buscada

Capítulo 7. COLABORACIÓN CON EL AYUNTAMIENTO DE MADRID

Gracias al convenio marco de colaboración entre el Ayuntamiento de Madrid y la Universidad Complutense de Madrid para el diseño y ejecución de proyectos fin de carrera y de prácticas de los alumnos de la Facultad de Informática, firmado entre el Vicerrector de Relaciones Institucionales y Relaciones Internacionales y el Delegado del Área de Gobierno de Medio Ambiente, Seguridad y Movilidad, hemos podido llevar a cabo este proyecto con un cliente real.

❖ Reuniones

• Primera Reunión

El día 11 de octubre de 2012 fuimos convocados a nuestra primera reunión con personal del Ayuntamiento. En ella se nos asignó a Ángeles Toribio como tutora del proyecto por parte del cliente, quien ha sido la encargada de hacer de intermediaria entre el equipo de desarrollo y los distintos departamentos del Ayuntamiento que se han visto implicados en el desarrollo de la aplicación, proporcionándonos la información necesaria en cada momento para el correcto funcionamiento de la misma.

Además durante el transcurso de esta reunión se trató de definir a grandes rasgos el contenido de la aplicación que se iba a desarrollar.

En principio se planteó la idea de incluir en la aplicación información sobre el alquiler de bicicletas en la ciudad de Madrid, pero hubo que descartarla debido a que la plataforma para proporcionar este servicio estaba aún en desarrollo y no se adecuaba a los plazos de nuestro proyecto. También se mencionó la existencia de unas rutas diseñadas para el ciclista que incluían infraestructuras de carril bici y calles que tienen menos tráfico y resultan más seguras para los ciclistas.

Otro punto que se mencionó en esta reunión fue la existencia de Puntos Limpios Fijos y Puntos Limpios Móviles, las diferencias entre ellos y la importancia de informar sobre el tipo de residuo que admite cada punto limpio y las cantidades de ellos, ya que por cuestiones de espacio los puntos limpios móviles están más limitados.

Por otro lado planteamos nuestras dudas respecto a los puntos de recarga eléctricos que nos había mencionado nuestra tutora de proyecto María Victoria López, ya que no sabíamos exactamente qué tipo de información se quería ofrecer al ciudadano. Debido a que el representante de este departamento no se encontraba en la reunión no se pudo hablar con detenimiento sobre el tema, pero se nos indicó que en la siguiente reunión acudiría la persona encargada para concretar esta información.

Además se planteó la idea de incluir nuevos tipos de recursos que inicialmente no se contemplaban en este proyecto, como son los parques o las gasolineras de gas (GNC y GLP) y se nos indicó una web en la que aparecía información sobre una serie de recursos con posibilidad de ser incluidos en la aplicación para que pudiéramos informarnos.

http://www.madrid.es/portales/munimadrid/es/Inicio/Ayuntamiento/Medio-Ambiente?vgnextfmt=default&vgnextchannel=4b3a171c30036010VgnVCM100000dc0ca8c0RCRD

Respecto a los requisitos sobre las herramientas que debíamos utilizar, se nos indicó que la base de datos debía hacerse en SQLite.

Por último solicitaron como requisito que desde la aplicación se informara al usuario que sus datos serían almacenados en una base de datos para clasificarlos y elaborar estadísticas sobre el uso de la aplicación a partir de ellos.

En esta reunión pudimos observar las grandes dudas del cliente sobre el contenido que debía incluir la aplicación por lo que decidimos ir definiendo una estructura para la aplicación que pudiera adaptarse a los cambios que podían producirse en futuras reuniones, ya que los trámites con el cliente iban a ser lentos y querían ver un prototipo funcionando en febrero de 2013.

- **Segunda Reunión**

La siguiente toma de contacto con el cliente fue en una reunión más específica para decidir de forma concreta el contenido de la aplicación. Esta reunión tuvo lugar el 27 de noviembre de 2012 y a ella acudieron varios representantes de las distintas áreas del Ayuntamiento.

- Directora del proyecto de Sistemas Informáticos – UCM: Victoria López
- Dirección General de Gestión Ambiental Urbana: Eva Hernández Sevillano
- Dirección General de Patrimonio Verde: María Eugenia García Flores
- Dirección General de Sostenibilidad: Sergio Fernández Balaguer
- Dirección General de Áreas Urbanas, Coordinación y Educación Ambiental y tutoras del proyecto por parte del Ayuntamiento: Maria Sol Mena Rubio, Ángeles M. Toribio Celda. Alicia Méndez Moreno (ausente en esta reunión)
- Jefe de Proyecto del I.A.M (Informática del Ayuntamiento de Madrid) – Jesús Jaime de Diego de Lucas
- Componentes del grupo de trabajo para el desarrollo de la aplicación: Ana Alfaro Orgaz, Sergio Ballesteros Navajas, Lidia Sesma Salgado
- Responsable de estandarización por parte de la UCM: Mariam Saucedo

En primer lugar se hizo una descripción general de la aplicación para informar a todos los asistentes del proyecto que se iba a desarrollar.

> “La aplicación mostrará al usuario una serie de recursos ambientales ofrecidos por el Ayuntamiento al ciudadano que se consideran de interés general. Cada uno de estos recursos podrá localizarse en relación a la ubicación GPS del usuario o bien

> introduciendo textualmente la dirección de interés. Además podrán filtrarse los resultados de la búsqueda dependiendo de los intereses del usuario."

Durante esta reunión se tomaron decisiones respecto a los recursos que ofrecen cada uno de los distintos departamentos del Ayuntamiento, que se consideraban de mayor interés para el ciudadano y por tanto debían ser incluidos en la aplicación.

Respecto al **área de Movilidad** se decidió que la información más relevante para el usuario debía hacer referencia a los aparcamientos públicos para coches de la ciudad de Madrid, las áreas reservadas para el estacionamiento de motocicletas, ubicaciones de los puntos de recarga para vehículos eléctricos, puntos de suministro de combustibles alternativos como son el bioetanol, GNC y GLP, lugares reservados para el estacionamiento de bicicletas e infraestructuras para planear rutas ciclistas, y en último lugar, información sobre las Áreas de Prioridad Residencial.

El mayor problema después de acordar la información que se incluiría en la aplicación referente a este departamento surgió al solicitar los datos sobre estos recursos (direcciones, coordenadas, etc…), ya que el representante no sabía cómo podría facilitarnos esta información ni en que formato disponía de ella.

En relación al **área de parques y jardines** solicitamos que nos enviaran el punto centroide de cada parque o jardín, ya que en Madrid existen parques con una amplia extensión de superficie.

Además se decidió que se debía incluir en la aplicación información sobre el horario de cada parque, direcciones y accesos, y la disponibilidad o ausencia de servicios como: aseos, zonas biosaludables para mayores, zonas infantiles, pipican, zonas deportivas regladas y puntos de información medioambiental dentro de los parques.

También se propuso colocar un código QR en la cartelería del parque que publicitara la aplicación.

En el caso del área de **Gestión Ambiental Urbana** se acordó que se nos facilitaría información sobre las coordenadas para la localización de puntos limpios fijos, móviles y contenedores de ropa, así como su dirección para poder ofrecérsela al usuario como información adicional. También se decidió que se nos proporcionarían datos sobre los horarios de cada punto limpio, los teléfonos en el caso de los puntos limpios fijos, los residuos admitidos y las cantidades máximas de éstos para cada tipo de punto limpio.

Toda la información respecto a ubicaciones de recursos fue solicitada en coordenadas de latitud y longitud en formato Excel.

Debido a que la información recibida provenía de un departamento distinto para cada recurso, recibimos una amalgama de archivos con los que tuvimos que lidiar para poder gestionarlos según nuestros intereses.

Algunos archivos fueron enviados protegidos con contraseña, otros en imágenes en PDF, como en el caso de las Áreas de Prioridad Residencial, Figura 129, de las cuales hubo que sacar las

coordenadas una a una ya que era la única forma posible de dibujar en la aplicación las calles que lo componen a partir de la imagen que se nos proporcionó.

Figura 129. Área de Prioridad Residencial

Además cada archivo contenía las coordenadas en un formato diferente que tuvo que ser transformado para poder gestionar las localizaciones de todos los recursos de forma similar. Incluso algunos archivos fueron recibidos sin coordenadas y hubo que calcularlas a partir de la pequeña referencia que se nos envió.

El último tema tratado en esta reunión fue en relación al IAM (Informática del Ayuntamiento de Madrid). En nuestra primera toma de contacto con este organismo del Ayuntamiento se nos comunicó que existían unos estándares y una serie de condiciones que debíamos cumplir para que la aplicación pudiera ser auditada por el departamento de calidad encargado de los temas relacionados con Informática del Ayuntamiento de Madrid.

El IAM es el encargado de coordinar los contenidos una vez que la aplicación esté operativa, Para ello debe mantener un contacto permanente con los servicios implicados que se encargarán de actualizar los contenidos. Por eso el equipo de desarrollo de la aplicación debe dejar la aplicación preparada para que sea aceptada por el IAM.

- ## Tercera Reunión

El día 11 de febrero de 2013 se nos convocó a una reunión con el IAM. Los representantes de este organismo, Jesús Jaime de Diego de Lucas e Isabel Pachón, nos informaron sobre las condiciones que debíamos cumplir para que el IAM aceptara la aplicación.

Comenzamos la reunión comentando la forma en la que se pensaba distribuir la aplicación y cómo se iban a encargar los distintos departamentos de actualizar los datos que se muestran en la aplicación.

Se nos indicó el funcionamiento de los 3 entornos que utilizan en el IAM:

Primero tiene lugar el **desarrollo,** ya que es necesario para el despliegue inicial. Este entorno se compone de varias fases:

- Primera instalación
- Pedir acceso para actualización
- Solicitud de una nueva Base de Datos mediante un Script de creación de tablas o bien un Backup
- Ver el diseño de la base de Datos y contemplar una posible ampliación
- Una vez desplegada la app y habiendo comprobado que funciona, se requiere un primer acceso a tarjetas SIMs específicas antes de sacar la aplicación al mercado.

En segundo lugar, se realizan tareas en el entorno de **preproducción**:

- Procesos de pruebas de estrés con muchos usuarios atacando el mismo servicio.
- Batería de pruebas de todo el sistema.
- Plan de implantación, necesario para el despliegue. En caso de fallo se podría recuperar lo anterior.

Por último tienen lugar las tareas en el entorno de **producción**:

- La aplicación funcionando.
- Los usuarios han hecho pruebas.
- Despliegue de la aplicación.

A continuación se nos explicó el tratamiento que hacen de un **subsistema móvil,** en este caso Android. También se mencionó la existencia de una API llamada PHONEGAP, que permitiría desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas y facilitando que la aplicación fuera multiplataforma, ya que el resultado es un híbrido que no es nativo al dispositivo. Esta idea se descartó debido a que el proyecto que se iba a desarrollar en este caso sería únicamente para Android.

En este aspecto se acordó que trabajaríamos sobre Java nativo para la implementación de la aplicación y evitar así problemas de rendimiento en caso de tener que trabajar con documentos geográficos ESRI.

El **Servicio Web** debe seguir una estructura de tres capas: capa de presentación, capa de negocio y capa de persistencia. Siguiendo esta estructura se consigue que cualquier modificación en una capa no afecte a las otras.

En la capa de persistencia deberemos usar Hibernate para comunicarnos con la base de datos. En la capa de negocio deberemos usar Spring, para incluir toda la lógica de negocio a ejecutar.

Para integrar el servidor con el cliente, se seguirá el protocolo estándar SOAP que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

Por último, se nos pidió que los datos que se solicitaran al usuario durante el registro no incluyeran el DNI por motivos legales de privacidad.

- ## Cuarta Reunión

El último contacto que tuvimos con el cliente se produjo en una reunión el día 11 de Marzo de 2013 con nuestras tutoras de proyecto por parte del Ayuntamiento, María Sol Mena Rubio y Ángeles M. Toribio Celda.

En ella pudieron comprobar el estado de la aplicación en esa fecha haciendo las correcciones necesarias y permitiéndonos comentar las incidencias que se habían producido hasta el momento.

Entre las correcciones nos solicitaron el cambio de algunas imágenes del logo corporativo del Ayuntamiento y la modificación de algunas funcionalidades, incluyendo un campo denominado “Tipo de vía” con los elementos que utilizan en el callejero oficial de Madrid (http://www-2.munimadrid.es/guia/visualizador/login.do), cuando el usuario introdujera una dirección para realizar la búsqueda de un tipo de recurso o añadiendo la opción de que el usuario pueda recuperar su contraseña, entre otras.

Respecto a las incidencias, comentamos aquellos recursos sobre los cuales no disponíamos aún de información, que dependían de terceros con los que no podíamos comunicarnos directamente, o aquellos que nos habían enviado de forma incompleta o en formatos que no podíamos manejar.

Además se comentó la idea de añadir una nueva información sobre itinerarios turísticos peatonales, pero no ha sido posible debido a que aún es un proyecto en desarrollo del Ayuntamiento y no han podido facilitarnos los datos referentes a esto.

Por último se mencionó la posibilidad de conseguir como patrocinadores alguna empresa de flotas y a partir de sus conocimientos comprobar si se nos había escapado incluir en la aplicación algún dato que resultara útil.

Capítulo 8. CONCLUSIONES Y TRABAJOS FUTUROS

8.1. Conclusiones

El “Mapa de Recursos Ambientales” ha resultado ser una aplicación muy útil y práctica debido a la gran cantidad de información que ofrece a los usuarios.

Al reunir en la misma aplicación una gran variedad de recursos y gestionarlos con un comportamiento similar hemos conseguido un aprendizaje rápido del uso y manejo de la herramienta por parte del usuario.

La importancia de diseñar una aplicación móvil para ofrecer esta información se ve reflejada en los estudios que demuestran que el porcentaje de accesos a Internet a través de dispositivos móviles ya se equipara al porcentaje de accesos a través de PC y continúa con tendencia al crecimiento. Además Android es el sistema operativo utilizado por más del 50% de los usuarios de smartphones.

A pesar de la existencia de aplicaciones con el fin de informar sobre alguno de los recursos que se contemplan en el Mapa de Recursos Ambientales, no existe ninguna aplicación que los reúna a todos y ofrezca una información tan completa.

El valor añadido a este proyecto lo aporta el cliente, al habernos facilitado información oficial de una institución como es el Ayuntamiento de Madrid, lo que nos ha permitido dotar a la aplicación de una gran fiabilidad en los datos que se ofrecen a los usuarios.

Además los requisitos estándares que nos propuso el servicio de Informática del Ayuntamiento de Madrid nos ha servido para superar un reto tecnológico el cual nos ha hecho crecer intelectualmente y creemos que nos ayudará también en nuestro próximo futuro laboral.

En un escenario donde nos hemos enfrentado por primera vez a un problema real, sin tener experiencia previa alguna en el trato con el cliente, hemos podido desarrollar una serie de habilidades relacionadas con el desempeño de las funciones que deberemos poner en juego durante nuestra carrera profesional.

Consideramos que el mejor modo de aprender y comprender la teoría es a través de la resolución de problemas de ingeniería reales utilizando como herramientas nuestras habilidades profesionales para alcanzar el objetivo de resolver de forma satisfactoria el problema que el cliente plantea.

En general, la experiencia ha sido un éxito tanto para el cliente como para los desarrolladores que hemos logrado alcanzar un objetivo común.

8.2. Trabajos futuros

La aplicación que se ha desarrollado admite varias extensiones. A continuación se enumeran algunas opciones que podrían formar parte de futuras ampliaciones:

- Debido a la gran extensión de la ciudad de Madrid, consideramos que podría resultar bastante útil integrar un navegador en la aplicación que pueda guiar al usuario desde su ubicación hasta el recurso que haya seleccionado.

- Para facilitar la labor de actualización de los datos recogidos en la aplicación por parte de los empleados del Ayuntamiento de Madrid podría resultar de gran utilidad la elaboración de un formulario web. De esta manera se conseguiría que la aplicación no quede obsoleta con el tiempo.

- A pesar de haber elegido Android como sistema operativo mayoritario para desarrollar la aplicación, existen otros sistemas operativos móviles que nos permitirían llegar a un número mayor de usuarios, como es el caso de iOS o Windows Phone.

- Actualmente no se dispone de una imagen de cada recurso, pero podría plantearse la posibilidad de ampliar la base de datos añadiendo a la información de cada elemento su imagen concreta para hacer esta información más descriptiva y el recurso más fácil de localizar.

- Otra posibilidad es crear un sistema de valoración y opinión para cada recurso, de manera que el usuario pueda evaluar la calidad/utilidad del recurso que ha utilizado, y así poder elaborar listas con los recursos mejor valorados por los usuarios o bien informar mediante la aplicación de qué infraestructuras necesitan ser mejoradas.

- Además la mejor forma de promocionar la aplicación es a través de las redes sociales. Cabe la posibilidad de integrar las principales redes sociales en la aplicación para poder sugerir recursos que puedan interesarle a amigos y familiares.

- Ya que la aplicación recibirá actualizaciones de recursos si se amplían las infraestructuras, resultaría interesante permitir al usuario recibir notificaciones, en caso de que lo desee, sobre la existencia de un nuevo recurso en una determinada zona.

Bibliografía

[1] Intensidad de tráfico en la ciudad de Madrid - http://www.madrid.es/UnidadesDescentralizadas/UDCMovilidadTransportes/ContenidosBasicos/ContenidosGenericos/IMD-VMD/ficheros/Listado_completo_IMD_2010.pdf

[2] Áreas de Prioridad Residencial - http://www.madrid.es/portales/munimadrid/es/Inicio/Ayuntamiento/Movilidad-y-Transportes/A.P.R.%3A-Areas-Prioridad-Residencial?vgnextfmt=default&vgnextoid=f4625a43ea2bf110VgnVCM1000000b205a0aRCRD&vgnextchannel=2b199ad016e07010VgnVCM100000dc0ca8c0RCRD

[3] Combustibles alternativos - http://www.madrid.es/portales/munimadrid/es/Inicio/Ayuntamiento/Medio-Ambiente/Educacion-ambiental/Energia-y-cambio-climatico/Vehiculos-menos-contaminantes-/Promocion-de-vehiculos-menos-contaminantes/Combustibles-alternativos?vgnextfmt=detNavegacion&vgnextoid=9798ad0d45f9e210VgnVCM2000000c205a0aRCRD&vgnextchannel=c770f91ca8d9e210VgnVCM2000000c205a0aRCRD

[4] Red de vías ciclistas - http://www.madrid.es/portales/munimadrid/es/Inicio/Ayuntamiento/Movilidad-y-Transportes/Oficina-de-la-bici/Madrid-en-bici/Red-de-vias-ciclistas/Vias-urbanas/Vias-urbanas?vgnextfmt=detNavegacion&vgnextoid=aa25caafc7fc8210VgnVCM2000000c205a0aRCRD&vgnextchannel=6f82d8f887e79210VgnVCM2000000c205a0aRCRD

[5] Puntos Limpios - http://www.madrid.es/portales/munimadrid/es/Inicio/Ayuntamiento/Medio-Ambiente/Residuos-y-limpieza-urbana/Puntos-Limpios?vgnextfmt=default&vgnextoid=a30815e08aa62110VgnVCM1000000b205a0aRCRD&vgnextchannel=6f5c9ad016e07010VgnVCM100000dc0ca8c0RCRD

[6] Fundación Humana -http://www.humana-spain.org/que-hacemos/medio-ambiente/?lang=es

[7] Parques de Madrid - http://www.madrid.es/portales/munimadrid/es/Inicio/Ayuntamiento/Medio-Ambiente/Parques-y-jardines/Patrimonio-Verde/Parques-en-Madrid?vgnextfmt=default&vgnextchannel=38bb1914e7d4e210VgnVCM1000000b205a0aRCRD

[8] Fundación ecolec - http://www.ecolec.es/index.asp

[9] Aplicación “Punto Limpio” - http://www.puntolimpio.info/descargalo_para_movil

[10] Aplicación “Dónde Reciclar” - https://play.google.com/store/apps/details?id=com.geekool.dondereciclar&feature=search_result#?t=W251bGwsMSwxLDEsImNvbS5nZWVrb29sLmRvbmRlcmVjaWNsYXIiXQ

[11] Desarrolladores de aplicaciones de puntos limpios - http://www.valoranetwork.com/

[12] Aplicación "Parques en Nantes - https://play.google.com/store/apps/details?id=jsmc.opendata.parcsanantes&feature=search_result#?t=W251bGwsMSwxLDEsImpzbWMub3BlbmRhdGEucGFyY3NhbmFudGVzIl0

[13] Aplicación "Parkopedia" -http://www.parkopedia.mx/

[14] Aplicación "Aparcamientos España"- http://aparcamientoses.compoundeyes.com/

[15] Aplicación "Spotcycle" - http://www.spotcycle.net/index_en.html

[16] Aplicación "Cycle Hire Widget" - http://www.littlefluffytoys.mobi/?p=350

[17] Aplicación "bici bike Valencia" - http://www.cpsingenieros.net/es/APPs/APPs%20Publicas/Bici%20Bike%20Valencia/

[18] Carril bici Barcelona - https://play.google.com/store/apps/details?id=net.rocboronat.android.carrilsbici.barcelona&feature=search_result#?t=W251bGwsMSwxLDEsIm5ldC5yb2Nib3JvbmF0LmFuZHJvaWQuY2Fycmlsc2JpY2kuYmFyY2Vsb25hIl0

[19] Aplicación "Electromaps" -http://www.electromaps.com/

[20] Aplicación "Recargo" - http://www.recargo.com/

[21] Aplicación "ElectriCar" - https://play.google.com/store/apps/details?id=es.sitvalencia.puntos_de_recarga&feature=search_result#?t=W251bGwsMSwxLDEsImVzLnNpdHZhbGVuY2lhLnB1bnRvc19kZV9yZWNhcmdhIl0

[22] Aplicación "Charge Locator Iberia" - http://www.chargelocator.com/Web/mapa.php?direccion=espa%F1a&zoom=5

[23] Informe Google uso de versiones Android noviembre 2012 - http://www.xatakamovil.com/sistemas-operativos/android-4-x-crece-lentamente-esperando-la-llegada-de-los-nuevos-nexus

[24] Informe Google uso de versiones Android junio 2013 - http://www.xatakamovil.com/sistemas-operativos/jelly-bean-ya-esta-en-el-33-de-los-dispositivos-android

[25] API Google Maps - https://developers.google.com/maps/?hl=es

[26] Xampp - http://www.apachefriends.org/es/xampp.html
[27] Apache Tomcat - http://tomcat.apache.org/

[28] Archivo .war - http://www.osmosislatina.com/tomcat/wars.htm

[29] SoapUI - http://www.soapui.org/

[30] SoapUI, jugando con web services - http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=introduccion-soapUI

[31] KSoap2-Android - http://code.google.com/p/ksoap2-android/

[32] Sommerville, Ian: *Ingeniería del Software*. Pearson (2004). Páginas 226-227 y 245-249.

[33] Servicios Web: http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

[34] Modelo 3 capas - http://www.jtentor.com.ar/post/Arquitectura-de-N-Capas-y-N-Niveles.aspx

[35] MySQL: http://dev.mysql.com/doc/refman/5.0/es/

[36] Licencia Pública General de GNU - http://www.gnu.org/licenses/licenses.es.html

[37] SQL Manager para MySQL - http://www.sqlmanager.net/en/products/mysql/manager?gclid=CNDi7vyh6LcCFSfLtAoddHAAmg

[38] Hibernate - http://docs.jboss.org/hibernate/core/3.5/reference/es-ES/html/

[39] Interview with Garvin King, Hibernate - http://www.javaperformancetuning.com/news/interview041.shtml

[40] Spring – http://www.springsource.org/

[41] Interview with Rod Johnson, Spring - http://howsoftwareisbuilt.com/2007/09/10/interview-with-rod-johnson-ceo-interface21/

[42] Enterprise JavaBeans - http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/ejb/sesion01-apuntes.htm

[43] Inversión de Control, Spring - http://www.javatutoriales.com/2010/12/contenedores-de-ioc-e-inyeccion-de.html

[44] Ciclo de vida de los Beans, Spring - http://www.javatutoriales.com/2011/02/spring-3-parte-4-ciclo-de-vida-de-los.html

[45] Patrón DAO, Spring - http://www.genbetadev.com/java-j2ee/spring-framework-el-patrn-dao

[46] Hibernate Query Languaje - http://docs.jboss.org/hibernate/core/3.5/reference/es-ES/html/queryhql.html

[47] Anaya Multimedia: *Programación Android*. Ed Burnette (2011). Capítulo2

[48] Historia y versions Android - https://androidos.readthedocs.org/en/latest/data/historia/

[49] Axis2 - http://axis.apache.org/axis2/java/core/

[50] Servicios Web. Conceptos básicos - http://www.emagister.com/curso-programacion-servicios-web/servicios-web-conceptos-basicos-1-2

[51] Servicios Web SOAP - http://www.emagister.com/curso-programacion-servicios-web/arquitectura-servicios-web-soap-1-3

[52] Cómo hace Axis2 su trabajo - http://www.elclubdelprogramador.com/2012/01/16/apache-axis2-el-contenedor-de-servicios-web-mas-usado/

[53] Desarrollar un web service desde WSDL con Axis2 - http://jcesarperez.blogsome.com/2007/07/24/desarrollar-un-webservice-desde-wsdl-con-axis2/

[54] Cloudbees - http://www.cloudbees.com

[55] PaaS, Plataforma como Servicio - http://alfonsogu.com/2008/08/14/definicion-de-paas/

[56] Continuous Integration, Martin Fowler - http://martinfowler.com/articles/originalContinuousIntegration.html

[57] Jenkins - https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

[58] Interview Jenkins creator, Kohsuke Kawaguchi - http://architects.dzone.com/articles/kohsuke-kawaguchi-interview

[59] The MIT License - https://github.com/jenkinsci/jenkins/blob/master/LICENSE.txt

[60] Apache Maven - http://maven.apache.org/

[61] Introducción a Maven - http://www.genbetadev.com/java-j2ee/introduccion-a-maven

[62] Introducción a Maven II, Project Object Model - http://www.genbetadev.com/java-j2ee/introduccion-a-maven-ii-project-object-model

[63] Apache Ant - http://ant.apache.org/

[64] Arquitectura Orientada a Servicios (SOA) - http://www.kybele.etsii.urjc.es/docencia/IS_LADE/2012-2013/Material/CAR%20Accenture%20-%20SOA%5B1%5D.pdf

[65] Java Reflection - http://www.arumeinformatica.es/blog/java-reflection-parte-1/

[66] jUnit, primeros pasos - http://www.dc.fi.udc.es/ai/tp/practica/junit/ppasos.html

[67] Log4j - http://adeideas.com/paquito/archives/161/Manual_corto_Log4j.htm

[68] Anaya Multimedia: *Programación Android*. Ed Burnette (2011). Capítulo 3

[69] Ben Shneiderman, Catherine Plaisant: Diseño de interfaces de usuario. Pearson (2005)

[70] Consumir web service SOAP con Android - http://androideity.com/2011/11/16/consumiendo-web-service-soap-json-con-android-i/

[71] Quicksort y su creador Charles Antony Richard Hoare - http://metodoquicksort.blogspot.com.es/2012_05_01_archive.html

[72] La proyección UTM - http://www.cartesia.org/data/apuntes/cartografia/cartografia-utm.pdf

[73] JScience - http://jscience.org/

[74] Uso de la pantalla táctil - http://support.google.com/android/bin/answer.py?hl=es&answer=168446

[75] La Catedral Innova - http://www.lacatedralonline.es

[76] Palacio de Telecomunicaciones - http://www.centrocentro.org/centro/home

Anexo - Publicaciones y difusión

❖ La Catedral Innova

El 17 de mayo de 2013 acudimos a nuestro primer acto público de presentación de la aplicación en La catedral Innova

¡MADRID! INNOVA — El programa del Ayuntamiento de Madrid para impulsar la innovación, las nuevas tecnologías y la I+D+i

En ella pudimos exponer nuestro trabajo y mostrar una demo a todos los asistentes del funcionamiento de la aplicación.

La catedral Innova es una Red de Centros de Innovación que pertenece al Ayuntamiento de Madrid. Se creó transformando la antigua fábrica industrial de "Boeticher y Navarro" y contempla toda la actualidad sobre innovación en el mundo empresarial.

Además es un centro de formación online sobre innovación compuesto por una red social de innovadores y se encarga de un programa de colaboración con empresas de innovación y nuevas tecnologías [75].

❖ Palacio de Telecomunicaciones

El día 22 de mayo expusimos nuevamente nuestro trabajo en el Centro del Palacio de Telecomunicaciones de Cibeles.

A esta nueva presentación de la aplicación acudieron representantes de la dirección de los distintos departamentos del Ayuntamiento de Madrid que se han visto implicados en el desarrollo de la aplicación [76].