Sistemas Informáticos
Curso 2013-2014

CALIDAD DEL SOFTWARE EN TECNOLOGÍAS MÓVILES

Adaptación y Actualización de aplicaciones móviles a los estándares de Calidad del Ayuntamiento de Madrid

HUGO GARCÍA RODRÍGUEZ

MIGUEL ROMÁN GARCÍA

PABLO MARTÍNEZ MOLINOS

Dirigido por:

Dra. Inmaculada Pardines Lence

Dra. Victoria López López

Facultad de Informática

Universidad Complutense de Madrid

AUTORIZACIÓN DE DIFUSIÓN

Hugo García Rodríguez, Miguel Román García y Pablo Martínez Molinos, 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 Calidad del Software en Tecnologías Móviles: Adaptación y Actualización de aplicaciones móviles a los estándares de Calidad del Ayuntamiento de Madrid, todo ello realizado durante el curso académico 2013-2014 bajo la dirección de María Victoria López López e Inmaculada Pardines Lence, ambas profesoras del Departamento de Arquitectura de Computadores y Automática de la Facultad de Informática.

Hugo García Rodríguez

Miguel Román García

Pablo Martínez Molinos

RESUMEN

Este proyecto engloba la actualización y adaptación a los estándares de trabajo del Ayuntamiento de Madrid de tres aplicaciones para móviles Android desarrolladas durante el curso pasado por compañeros de la UCM, siendo estas Itinerarios del Retiro, Recycla.me y Recyclate!.

El objetivo es la adaptación y actualización de las tecnologías utilizadas por las aplicaciones anteriormente citadas para que el Ayuntamiento de Madrid pudiera incorporarlas a sus servidores y ofertarlas desde la concejalía de Medio Ambiente de Madrid y proceder a la siguiente fase de desarrollo para dichas aplicaciones: su desarrollo para el resto de sistemas operativos móviles como iOS.

La organización y tecnologías de las que hacen uso las aplicaciones están relacionadas con los departamentos de Medio Ambiente y Movilidad de la ciudad de Madrid, así como del Departamento de Calidad del Software del Ayuntamiento. Estas directrices fueron suministradas por el servicio de Informática del Ayuntamiento de Madrid (IAM) y consensuadas durante las reuniones mantenidas entre nuestro equipo de desarrollo y los responsables de dicho departamento.

Para llevar a cabo el desarrollo de este proyecto hemos tenido que documentarnos acerca de las tecnologías propuestas como Enterprise Java Beans, Java Persistance Access o Maven (entre otras) y enfrentarnos al reto adicional de afrontar las tareas de mantenimiento y depuración de código de terceros. El objetivo de esta memoria es ofrecer una guía útil y clara de los pasos seguidos durante dicho proceso de estandarización con el fin de facilitar desarrollos posteriores similares.

En general las modificaciones han afectado a los servicios web de las tres aplicaciones, así como a sus sistemas de almacenamiento y recuperación de recursos, y por otro lado, a la actualización y mantenimiento de recursos y librerías de terceros que habían quedado obsoletas o desactualizadas.

De esta forma, se han tratado de aproximar lo más posible las tres aplicaciones de Itinerarios guiados Parque del Retiro, Recycla.me y Recyclate! a los estándares del IAM con el fin de que puedan ser auditadas por el departamento de calidad de esta entidad y estar disponibles

para el ciudadano en un breve período de tiempo.

PALABRAS CLAVE

Aplicación móvil, Android, Medio Ambiente, Movilidad, Itinerarios, Recycla.me, Recyclate!, reciclar, estandarización, web services.

ABSTRACT

This project comprises the updating of three apps for Android mobile phones, developed by UCM colleagues along the last year, which are Itinerarios del Retiro, Recycla.me and Recyclate!, and their subsequent adaptation to work standards of Ayuntamiento de Madrid (Madrid City Hall).

The aim of the adaptation and updating of the technologies used in the above mentioned apps, in order to may be included into the City Hall servers and to be offered by Consejería de Medio Ambiente de Madrid (Office of Natural Enviroment Affairs) as one of its services, proceeding after to their next development stage: their development for the rest of mobile operating systems as iOS.

The organization and technologies that these apps make use of are related to two departments: Departamento de Medio Ambiente y Movilidad de Madrid (Department of Natural Environment Affairs and Mobility) and Departamento de Calidad del Software del Ayuntamiento (City Hall Department for Software Quality). These guidelines were supplied by IAM, the Computer Engineering Department of Madrid City Hall and agreed along the meetings held by our development team/crew and the officers in charge of this department.

To accomplish the development of this project it was required to be acquainted with the technologies proposed, as Enterprise Java Beans, Java Persistence Access or Maven, among others, as well as to face tasks as the maintenance and depuration of codes rendered by third parties.

The intend of this report is to provide a guide compiling, usefully and clearly, every step taken during all this standardization process which allows to make later similar progresses easier.

On one hand, the modifications done have generally affected to web services of the three apps as well as to their storage systems and their resources recovery, and on the other hand, to the updating and the maintenance of resources and libraries carried out by third parties which have resulted obsolete or out-dated.

This way, to bring the three apps (Itinerarios del Retiro, Recycla.me and Recyclate!) as closer as possible to the standards of IAM has been the

foremost intention of the project, with the challenge of preparing them to be audited by the department of quality of this public institution and then become, subsequently, available to the citizen in the short term.

KEYWORDS

Mobile Application, Android, Environment, Mobility, Itinerarios, Recycla.me, Recyclate!, recycle, standarization, web services.

Dedicado a nuestra familia, amigos,

profesores y al Soportal.

Gracias

Índice

Capítulo 1. INTRODUCCIÓN

En este primer capítulo se presenta el tratamiento del proyecto, exponiendo las aplicaciones con las que se ha trabajado y justificando por qué es recomendable que el desarrollo cumpla siempre unos estándares de calidad.

1.1 Motivación y Objetivos

Nuestro proyecto surge de la necesidad de estandarizar, actualizar, y facilitar la escalabilidad de 6 aplicaciones móviles (y sus sistemas informáticos asociados) desarrolladas en el marco de un Convenio de Colaboración entre la Facultad de Informática de la UCM y el Área de Gobierno de Medio Ambiente y Movilidad del Ayuntamiento de Madrid. Estas aplicaciones fueron presentadas al público el 12 de Julio de 2013 y los ciudadanos pueden descargarlas para su uso desde una web específica (www.tecnologiaUCM.es). El objetivo final del convenio es poner dichas aplicaciones a disposición de los ciudadanos desde Google Play y las páginas oficiales del Ayuntamiento de Madrid. Al tratarse de un organismo público, Informática del Ayuntamiento de Madrid (IAM [1]) requiere a los desarrolladores (empresas o particulares) la estandarización del software de acuerdo con sus documentos de especificación para facilitar su mantenimiento futuro y garantizar un ciclo de vida correcto a los proyectos.

Nuestro proyecto, por lo tanto, se centra en las últimas fases de desarrollo, donde los ingenieros tienen que enfrentarse con proyectos abiertos y reutilización de código, lo que implica un nuevo reto muy ambicioso para estudiantes de Ingeniería Informática al trabajar con un proyecto real que va a instalarse definitivamente en los servidores oficiales de un organismo público.

La necesidad de estandarización del software es cada vez más frecuente especialmente en grandes organizaciones donde la estructuración es fundamental para garantizar el buen funcionamiento de los servicios que se ofrecen. Por otra parte, el auge de los dispositivos móviles hace que en la actualidad multitud de empresas y organizaciones de todo tipo se lancen a ofrecer servicios a través de estos dispositivos, con la intención de hacer la vida del usuario más fácil. Al tratarse de tecnologías novedosas y en continuo desarrollo, y ante el deseo de querer ofrecerlas al usuario lo antes posible con las máximas prestaciones, las compañías se enfrentan al problema de tener que manejar un gran número de aplicaciones

desarrolladas de manera muy diferente y que tratan la información de forma distinta.

En nuestro caso, nos encontramos con 6 aplicaciones para móviles Android desarrolladas por compañeros nuestros durante los cursos 2011-2012 y 2012-2013. Cada una de estas aplicaciones fue desarrollada de un modo diferente, sobre versiones diferentes de Android, como se puede observar en las respectivas memorias de estos proyectos [2-7].

Uno de nuestros objetivos principales ha sido estandarizar estas aplicaciones, atendiendo principalmente a la forma en que estas tratan los datos.

Esta meta, se torna mucho más ambiciosa cuando hablamos de un organismo como el Ayuntamiento de Madrid, como propietario de los datos que estas utilizan. Nuestras apps no solo deberán tratar los datos de una manera eficiente, sino que esta tiene que ser segura y de acuerdo a los requisitos de calidad propuestos por el IAM.

Además, hemos documentado la labor de actualización y estandarización que demanda el IAM para estas aplicaciones, con el fin de facilitar este trabajo en proyectos futuros. Para ello se han tenido en cuenta los criterios de calidad utilizando las estructuras y tecnologías de obligado cumplimiento en las herramientas informáticas desarrolladas en el Ayuntamiento de Madrid. Se ha puesto especial interés en la escalabilidad del software y la facilidad para introducir mejoras o nuevas necesidades funcionales que puedan surgir en el futuro, desligando su implementación del equipo de desarrolladores.

Tras la planificación de tareas asociadas a nuestro proyecto, finalmente se ha marcado como objetivo la implantación de tres de las aplicaciones referenciadas: Itinerarios Retiro, Recycla.me y Recyclate! [2-4].

1.2 Reutilización y actualización del software

En este proyecto, la estandarización del software es la clave fundamental, sin embargo, resulta imposible hablar de estandarización sin mencionar otros dos conceptos relacionados y no menos importantes en Ingeniería del Software: la reutilización y actualización del software.

Comenzaremos exponiendo brevemente los fundamentos sobre reutilización del software. Esta idea, que surge formalmente en 1968, aparece como una alternativa para desarrollar software de una manera más eficiente y productiva. La idea es reutilizar elementos y componentes software ya desarrollados y adaptarlos a nuestras necesidades, en lugar de tener que desarrollar nuestro software desde cero. Actualmente las empresas desarrollan grandes proyectos con equipos multidisciplinares, muchas veces distribuidos geográficamente por lo que es habitual que en un mismo proyecto se utilicen distintos lenguajes de programación y distintos entornos de desarrollo. Se justifica así el desarrollo de software como si fueran componentes electrónicos, de tal manera, que un mismo componente pueda ser utilizado en diferentes sistemas. Para conocer en más detalle qué es la reutilización de software, el artículo [8] de la Universidad de Pekín hace una muy interesante aproximación. El resumen de este artículo puede leerse a continuación.

"Software reuse offers a solution to eliminate repeated work and improve efficiency and quality in the software development. In the recent ten years, object oriented technology has appeared and become a mainstream technology, thereby providing fundamental technology support for software reuse. Software reuse regains more attention in software engineering research and is considered a practical and feasible approach to solving the software crisis. Software reuse is generally classified into two catalogues: product reuse and process reuse. Reuse based on software components is the important form of product reuse and is the major area of software reuse research. At the same time, software component technology plays an important role in distributed object research. Therefore, software component technology is regarded as a key factor of successful software reuse. The development and application of software reuse technology will facilitate the revolution of software development and reorganize software industry. As a result, the development of software components will become an independent and inseparable industry. The revolution offers a good chance for Chinese software development. This paper is a summarization on the development of software reuse technology. It presents

fundamental concepts and key techniques of software reuse. After introducing several successful research and practice in software reuse, including Jade Bird Project, a Chinese national key project supported by the government, it proposes some ideas on how to reinforce research and application of related techniques and facilitate the development of software industry in China."

(Ref.:http://en.cnki.com.cn/Article_en/CJFDTOTAL-DZXU902.018.htm)

Pese a que parece un concepto que puede resultar de utilidad para el desarrollo de software en todos los niveles, la realidad es que muchas compañías y equipos de trabajo deciden desarrollar todo su software desde cero. El siguiente artículo [9] publicado por la G'arnegie Mellon University de Pittsburgh, nos ayuda a entender por qué la reutilización del software no se ha extendido como se podía esperar, y a comprender como de eficiente puede ser seguir esta técnica.

"Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. This simple yet powerful vision was introduced in 1968. Software reuse has, however, failed to become a standard software engineering practice. In an attempt to understand why, researchers have renewed their interest in software reuse and in the obstacles to implementing it. This paper surveys the different approaches to software reuse found in the research literature. It uses a taxonomy to describe and compare the different approaches and make generalizations about the field of software reuse. The taxonomy characterizes each reuse approach in terms of its reusable artifacts and the way these artifacts are abstracted, selected, specialized, and integrated. Abstraction plays a central role in software reuse. Concise and expressive abstractions are essential if software artifacts are to be effectively reused. The effectiveness of a reuse technique can be evaluated in terms of cognitive distance—an intuitive gauge of the intellectual effort required

to use the technique. Cognitive distance is reduced in two ways: (1) Higher level abstractions in a reuse technique reduce the effort required to go from the initial concept of a software system to representations in the reuse technique, and (2) automation reduces the effort required to go from abstractions in a reuse technique to an executable implementation. This survey will help answer the following questions: What is software reuse? Why reuse software? What are the different approaches to reusing software? How effective are the different approaches? What is required to implement a software reuse technology? Why is software reuse difficult?

What are the open areas for research in software reuse?"

(Ref: http://dl.acm.org/citation.cfm?id=130856)

Para terminar con la reutilización del software, y dado que consideramos que es una técnica que puede resultar muy útil para cualquier tipo de desarrollo, exponemos el siguiente artículo [10], publicado por IEEE, en el que se exponen los factores clave a tener en cuenta si queremos reutilizar software de una manera eficiente.

"This paper aims at identifying some of the key factors in adopting or running a company-wide software reuse program. Key factors are derived from empirical evidence of reuse practices, as emerged from a survey of projects for the introduction of reuse in European companies: 24 such projects performed from 1994 to 1997 were analyzed using structured interviews. The projects were undertaken in both large and small companies, working in a variety of business domains, and using both object-oriented and procedural development approaches. Most of them produce software with high commonality between applications, and have at least reasonably mature processes. Despite that apparent potential for success, around one-third of the projects failed. Three main causes of failure were not introducing reuse-specific processes, not modifying nonreuse processes, and not considering human

factors. The root cause was a lack of commitment by top management, or nonawareness of the importance of those factors, often coupled with the belief that using the object-oriented approach or setting up a repository seamlessly is all that is necessary to achieve success in reuse. Conversely, successes were achieved when, given a potential for reuse because of commonality among applications, management committed to introducing reuse processes, modifying nonreuse processes, and addressing human factors"

(Ref:http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=995420&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D995420)

A continuación trataremos otro concepto muy a tener en cuenta, la actualización del software. Este concepto resulta clave, especialmente en un campo como es la tecnología móvil. Los dispositivos móviles evolucionan cada día, ofreciendo nuevas funcionalidades y los usuarios quieren sacar el máximo partido a cada de uno de los recursos que le ofrece su teléfono móvil. Una aplicación que saca el máximo partido de los recursos de un dispositivo tiene muchas posibilidades de triunfar. Además cuanto antes consiga implementar los cambios para utilizar la última mejora de un dispositivo en concreto, más diferencia sacará a sus competidores.

La actualización del software también es necesaria para solucionar diferentes problemas que puedan surgir, y que no se encuentran hasta que la aplicación no pase a producción. Esta es una parte fundamental, ya que uno de estos fallos no detectados puede suponer pérdidas muy importantes. Para ello el software tiene que ser flexible y estar preparado para ser actualizado de la manera más eficiente posible. Además, generalmente, no son los mismos equipos quienes se ocupan del primer desarrollo y las actualizaciones, por lo que el código debe ser lo más explicativo posible.

Respecto a este punto, uno de los temas más tratados en los últimos tiempos es el software actualizable dinámicamente, es decir programas que actualizan su código sin tener que parar su ejecución. Esto es especialmente

interesante para la actualización del software de los servidores. El siguiente artículo [11] realiza una aproximación muy interesante a este concepto:

> *"Dynamic software updating (DSU) enables running programs to be updated with new code and data without interrupting their execution. A number of DSU systems have been designed, but there is still little rigorous understanding of how to use DSU technology so that updates are safe. As a first step in this direction, we introduce a small update calculus with a precise mathematical semantics. The calculus is formulated as an extension of a typed lambda calculus, and supports updating technology similar to that of the programming language Erlang. Our goal is to provide a simple yet expressive foundation for reasoning about dynamically updateable software. In this paper, we present the details of the calculus, give some examples of its expressive power, and discuss how it might be used or extended to guarantee safety properties."*
>
> *(Ref: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.7176)*

1.3 Necesidad de estandarización en organismos públicos y grandes empresas

Como se ha comentado anteriormente, el objetivo del proyecto es conseguir que las aplicaciones se adapten a los estándares establecidos por el Ayuntamiento de Madrid. Sin embargo, este es un asunto que no sólo preocupa al Ayuntamiento, ya que hay muchos ejemplos de grandes empresas y organismos públicos que están intentando estandarizar todas sus herramientas, buscando así la reducción de costes y la mejora en otros aspectos como es la seguridad.

Con esta doble finalidad se ha ido desarrollando, en los últimos años, Java EE [12] (Java Enterprise Edition), el cual está basado en estándares y podría ser definido como una especificación "paraguas" que aglutina distintos JSRs [13] (Java Specification Request). Cabe preguntarse por qué los estándares son tan importantes, especialmente al comprobar que

algunos de los *frameworks* Java más exitosos (como Struts, Spring…) no están estandarizados.

A lo largo de la historia, el hombre ha creado estándares para facilitar las comunicaciones e intercambios. Algunos ejemplos notables son el lenguaje, las divisas, los sistemas horarios, las unidades de medidas, las líneas de ferrocarril, la electricidad, los teléfonos o los distintos protocolos y lenguajes de programación. En los inicios de Java, cuando alguien hacía cualquier tipo de desarrollo web o empresarial, debía crear sus propios *frameworks* o bien encadenarse a *frameworks* comerciales vendidos por otras compañías. Más adelante llegaron los días de los *frameworks* de código abierto, los cuales no siempre están basados en estándares libres. Se puede estar usando código libre y estar encadenado a una implementación, o utilizar código que implementa estándares y ser portable.

Java EE provee estándares abiertos implementados por varios *frameworks* comerciales (WebLogic, WebSphere, MQSeries...) o de código abierto [14] (GlassFish, JBoss, Hibernate, Open JPA, Jersey...) para manejar transacciones, seguridad, persistencia de objetos, etc. Hoy, más que nunca en la historia de Java EE, una aplicación es desplegable en cualquier servidor de aplicaciones compatibles con muy pocos cambios.

1.4 Estado del arte

Esta sección dedicada al Estado del Arte se divide en dos subapartados. En primer lugar, se exponen las aplicaciones con las que nos encontramos, su estado actual y cómo funcionan para, posteriormente, plantear las distintas alternativas de desarrollo de aplicaciones multiplataforma. Una de las ideas surgidas a la hora de afrontar este proyecto fue que las aplicaciones originales fueran capaces de funcionar sobre cualquier sistema operativo móvil pero, tras hablar con las coordinadoras del proyecto y con el IAM (Informática del Ayuntamiento de Madrid), se decidió que esta idea debía postergarse, ya que su implementación no tenía la misma prioridad que la preparación de la aplicaciones en términos de rápida disponibilidad para el público Android.

Aplicaciones que nos encontramos

En este punto se pretende describir las aplicaciones desarrolladas en cursos anteriores, atendiendo a sus funcionalidades, así como a las tecnologías y herramientas con las que fueron desarrolladas, entrando a analizar con

mayor detalle a las tres aplicaciones a las que este proyecto se dedica: Itinerarios guiados Parque del Retiro, Recycla.me y Recyclate!.

Además de estas tres aplicaciones existen otras tres, creadas gracias al convenio entre la Facultad de Informática de la UCM y el Área de Gobierno de Medio Ambiente y Movilidad del Ayuntamiento de Madrid. Estas son: Camino seguro al cole [5], Hábitat Madrid [6] y Mapa de recursos de la Comunidad de Madrid [7]. Las tareas relacionadas con Mapa de recursos de la Comunidad de Madrid han sido desarrolladas por compañeros del grupo "Sistemas OPEN-DATA Madrid: Aplicaciones móviles para medioambiente", con quienes se ha hecho necesaria una constante coordinación, compartiendo en más de una ocasión reuniones y pautas, para así lograr mirar en la misma dirección.

Es imprescindible, en primer lugar, presentar un pequeño resumen de la funcionalidad de cada una de estas aplicaciones.

Camino seguro al cole

Camino seguro al cole ha sido originalmente desarrollada en 2013 por Álvaro Bustillo Rebanal, Arturo Callejo Luis y Héctor A. Martos Gómez. Esta aplicación nace bajo el marco del proyecto "Madrid a pie: camino seguro al cole" [15] creado por el Ayuntamiento de Madrid en 2007. Apuesta por fomentar la seguridad y la autonomía de los niños en los desplazamientos a los centros educativos, así como por conocer cómo se mueven, qué necesidades pueden tener y qué les favorece. Está desarrollada para Android y fija los caminos que deben seguir los menores para ir seguros al colegio, gracias a las denominadas "arañas de movilidad". Además, a lo largo del trayecto se cuenta con la ayuda de comerciantes de la zona que se hayan inscrito en el proyecto. Mediante esta herramienta se pretende que los niños, con la colaboración de sus padres, sean capaces de trazar y visualizar una ruta segura en un mapa entre su domicilio y el colegio, con la posibilidad de recorrerla guiándose por sencillas instrucciones y pudiendo visualizar, en todo momento, la posición del usuario en el mapa. Con ella se pretende favorecer que, con la supervisión paterna, el niño se guíe y se familiarice con su entorno. La aplicación permite también el envío de notificaciones que adviertan de cualquier incidencia en el camino, haciendo así partícipes a los niños del mantenimiento de la ciudad. La *Figura 1.1* muestra algunas pantallas representativas de esta aplicación.

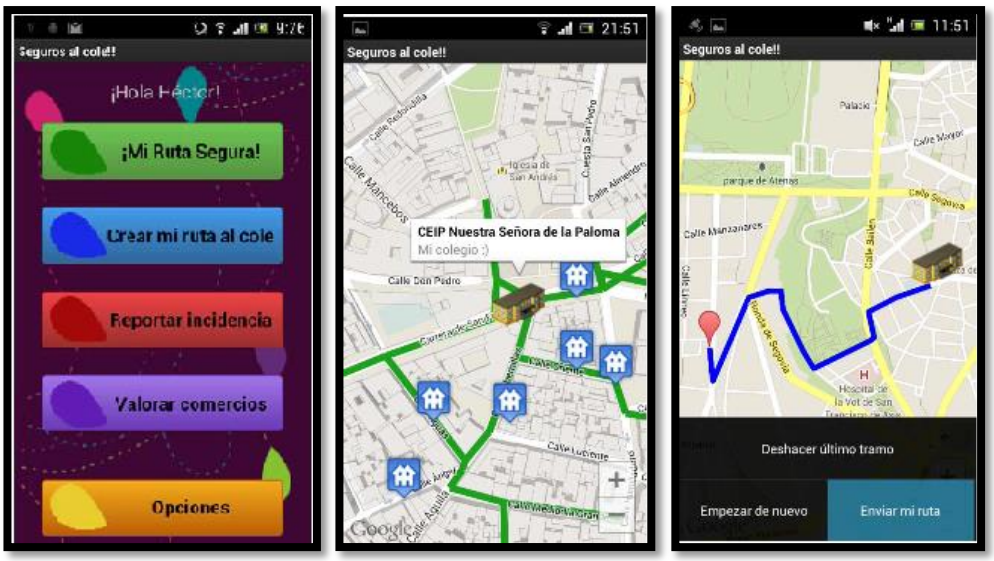


Figura 1.1.- Aplicación Camino Seguro al Cole

Hábitat Madrid

Hábitat Madrid ha sido originalmente desarrollada por Carlos Fernández Vázquez, Mª del Mar Octavio de Toledo Rodríguez y Antonio Sanmartín Sánchez durante el año 2013. Hábitat Madrid es un proyecto integral para la gestión de las escuelas temáticas del programa de Educación Ambiental del Ayuntamiento de Madrid [16]. Desarrollada para Android, permite al ciudadano de Madrid acceder a la información de los distintos talleres y escuelas que tengan lugar en la ciudad, con las opciones de poder reservar plaza en una o varias de estas actividades y de recibir notificaciones de todas las novedades relativas a estos ciclos. Este proyecto pretende agilizar el sistema de acceso a los diferentes talleres organizados por el ayuntamiento y para ello, no sólo se desarrolló la aplicación orientada al usuario, sino que también se añadieron los elementos necesarios para mantener y actualizar la base de datos, encargada de almacenar y gestionar la información concerniente tanto a los cursos como a los datos de los ciudadanos inscritos. En la *Figura 1.2* se puede apreciar el aspecto de la aplicación Hábitat Madrid.

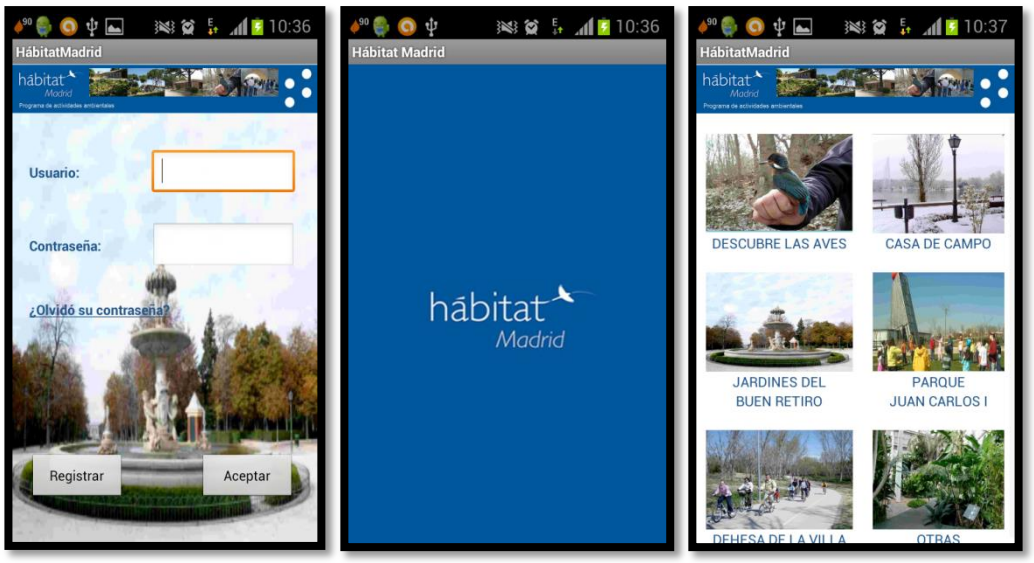


Figura 1.2.- Aplicación Hábitat Madrid

Mapa de recursos de la Ciudad de Madrid

Esta aplicación fue originalmente desarrollada por Ana Alfaro Orgaz, Sergio Ballesteros Navajas y Lidia Sesma Salgado en el curso 2012- 2013. 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.

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. La aplicación permite localizar, a través de un mapa y de manera sencilla, cualquiera de las infraestructuras antes mencionadas, ofreciendo a los madrileños y turistas los datos necesarios para disfrutar de los servicios que brinda la ciudad de Madrid. La *Figura 1.3* muestra algunas de las pantallas de la aplicación.

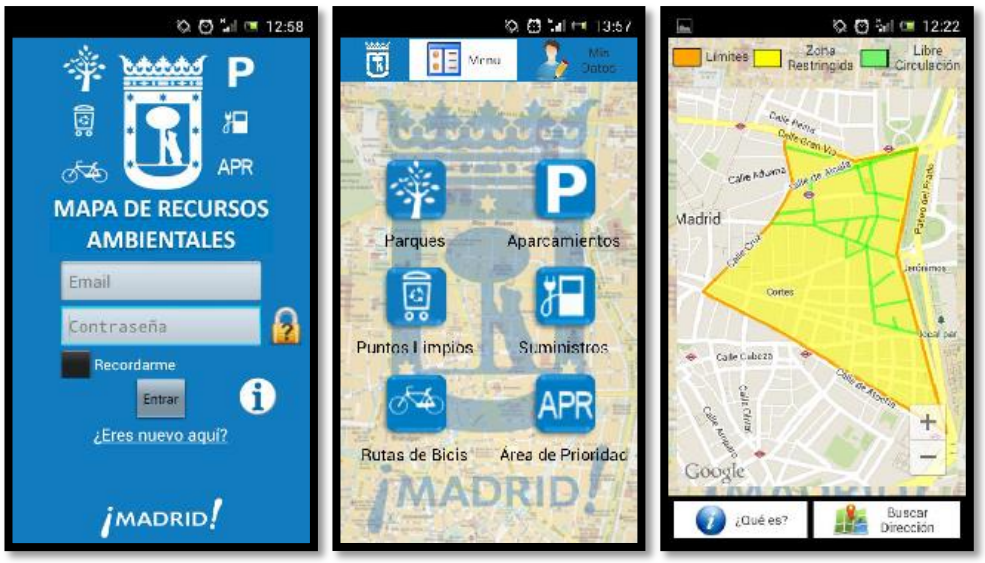


Figura 1.3.- Aplicación Mapa de Recursos

A continuación se describen, con mayor detalle y atendiendo a los aspectos más técnicos, las tres aplicaciones que conforman las competencias de este proyecto.

Itinerarios guiados Parque del Retiro

Esta aplicación fue creada en primera instancia por Miguel Bueno Gómez, Alberto Segovia Sanz y Víctor Torres González y durante el curso 2012-2013. Itinerarios guiados Parque del Retiro se crea con la finalidad de complementar las tradicionales visitas guiadas por los diferentes jardines del Parque del Buen Retiro y propone una solución virtual a la gran demanda de estas visitas. Su objetivo es procurar al usuario una ruta auto-guiada por los distintos estilos de jardinería del parque, con información pormenorizada sobre las distintas localizaciones que constituyen la riqueza paisajística e histórica del entorno del recorrido. Cualquier ciudadano puede realizar el recorrido de los jardines accediendo a la misma información sin restricciones de fechas ni horarios.

Figura 1.4- Aplicación Itinerarios Retiro

Requisitos funcionales

A grandes rasgos, la funcionalidad de esta aplicación es la de brindar itinerarios interactivos en el Parque del Buen Retiro, aportando información relativa a cada punto de interés del parque.

En su pantalla principal se muestra una "Introducción", un "¿Cómo usar?" (sencillos textos planos que facilitan la utilización de la herramienta al usuario), además de un "Acerca de" con enlaces relacionados con la aplicación.

Se ofrecen diferentes formas de hacer uso de esta aplicación:

- Seleccionando la opción Iniciar Ruta se muestra un listado con los puntos de interés del recorrido.
- Seleccionando la opción de acceder directamente a un punto de interés en concreto, se nos dirige específicamente a ese punto.
- Comprobando la localización del punto de interés más próximo a la posición actual del usuario, la aplicación indica cómo llegar a él.

Además, en cada punto de interés el usuario puede escuchar un audio con información relativa al lugar en el que se encuentra, información que también está disponible por escrito. La aplicación incluye, en cada uno de estos puntos de interés, una serie de imágenes que ilustran el paraje, para una mejor identificación. Por último, el botón "¿Sabías qué?" aporta curiosidades a cerca del lugar.

Herramientas

Al haber sido desarrollada para el sistema operativo Android, se utilizó el *Java Development Kit* (JDK) y el entorno de desarrollo integrado Eclipse. Fue necesaria, asimismo, la instalación de la SDK de Android para Eclipse, que permite desarrollar proyectos Android. La versión del sistema operativo elegido fue la 2.3.3 (*Gingerbread*), por los recursos que ofrece y por ser compatible con la mayoría de los dispositivos Android en la actualidad.

Las principales librerías utilizadas son:

- Librería de Google Maps, para trazar rutas de un punto a otro en el Parque del Retiro y para poder recalcular rutas.
- Librería Universal Image Loader, para poder obtener varias imágenes descargadas de Internet desde un punto específico y mostrarlas de la mejor manera posible.
- Librería Scale ImageView, para aplicar zoom a imágenes del proyecto.

Para la obtención de los recursos multimedia usa un servidor HTTP Apache (2.0) [17], así como servicios SSH [18] para acceder a él. Estos recursos se alojaron en servidores gratuitos.

La aplicación Itinerarios del Retiro original se apoyaba en una base de datos interna SQLite bastante sencilla, que atendía a un modelo entidad-relación. En esta base de datos se almacenaban las URLs de los recursos multimedia a los que fuese necesario acceder.

Recycla.me

Recycla.me fue desarrollada durante el curso 2011-2012 por Daniel Jesús Sanz Candilejo, Mariam Saucedo Alonso y Pilar Torralbo Fernández. El objetivo principal de Recycla.me es enseñar a los niños de 7 a 11 años a reciclar correctamente cualquier tipo de envase que potencialmente pueda tener asignado un código de barras. Los niños con ayuda de sus padres o un tutor, aprenden cómo reciclar los distintos tipos de envases que se puedan encontrar, utilizando el código de barras o especificando los materiales del envase. Además, incluye distintos consejos y trucos para motivar a los niños a reciclar. Cabe mencionar que Recycla.me recibió el premio SOPRA 2012 y una mención especial en los premios eMadrid en la edición de 2013.

Figura 1.5.- Aplicación Recycla.me

Requisitos funcionales

Estamos ante una aplicación que aspira a fomentar una rutina de reciclaje, por lo que las funcionalidades que ofrece pretenden abarcar la mayoría de los temas referentes a este proceso. El servicio fundamental que nos brinda esta herramienta es el de ayudarnos en la correcta separación y reciclaje de los productos. Existen dos posibilidades:

- Mediante un código de barras. La aplicación escanea el código de barras de un producto y a continuación muestra cómo debe separarse y reciclarse. Para esta función es imprescindible la conexión a Internet.

- Sin código de barras. El usuario dispone de una selección de imágenes para poder identificar el producto que más se asemeja al que queremos reciclar. Una vez seleccionada la imagen, la aplicación nos informa de los materiales de los que está compuesto el producto y cómo proceder a su reciclado.

Otro de los aspectos importantes en Recycla.me es la verificación del despiece del producto. Al estar destinada a un público infantil, debe ser un adulto el que valide el proceso de despiezado descrito anteriormente, por lo que la aplicación añade este paso adicional a dicho proceso.

Está enriquecida con diferentes consejos sobre reciclaje que se irán mostrando a los niños cuando estos realicen el proceso, dando gran importancia al componente visual para potenciar esta funcionalidad.

También se ofrece la posibilidad de visualizar en un mapa la ubicación del Punto Limpio más cercano, ya que puede que el reciclaje de algunos productos requiera acudir a este tipo de servicios.

Permite acceder a las páginas web de los colaboradores de la aplicación, dispone de una sección de ayuda al usuario y para determinadas acciones el sistema pregunta al usuario si se desea conectar el dispositivo mediante Wi-Fi.

Herramientas

Al igual que Itinerarios del Retiro esta aplicación ha sido desarrollada bajo el sistema operativo Android, para la versión del SDK 2.2 (*Froyo)*, por lo que ha sido realizada en lenguaje Java y en entorno de desarrollo Eclipse, instalando el ADT Plugin para poder crear proyectos Android.

Se hace uso de la API de Google Maps para la correcta implementación de los mapas.

El proyecto incluye alguna librería externa para cumplir con la funcionalidad y mostrar un mejor aspecto:

- Librería ZXing [19] para la lectura de códigos de barra, también ofrece la posibilidad de leer códigos QR. Se ha importado esta

librería como un .jar dentro del proyecto para que ocupe menos espacio.

- Librería Android ViewPagerIndicator [20], que permite el desplazamiento por los distintos materiales como si se pasaran las hojas de un libro deslizando el dedo por la pantalla de manera horizontal.

- Para el servidor se utilizó un servidor de aplicaciones gratuito Apache. Alojado en 000webhost.com, totalmente gratuito que proporciona hasta 1500 MB de almacenamiento, SQL para las consultas y PHP para la conexión entre la aplicación y los datos.

Recyclate!

Recyclate! fue originalmente desarrollada por Belén Abellanas Aguilera, Jaime Ramos Montero y L. Ignacio P. de Ziriza Sánchez durante el curso 2012-2013. La aplicación Recyclate! es una aplicación complementaria a Recycla.me, orientada a los adultos. Además de la posibilidad de saber en qué contenedor hay que desechar cada envase mediante su código de barras, también ofrece otras nuevas funcionalidades. Entre ellas, distintos artículos con consejos para reciclar y reutilizar, un listado de los puntos limpios de la Comunidad de Madrid y un juego que plantea preguntas relativas al reciclaje y a la ciudad de Madrid, ofreciendo la posibilidad de competir con amigos.

Figura 1.6.- Aplicación Recyclate!

Requisitos funcionales

Al ser una aplicación basada en Recycla.me la mayoría de las prestaciones que ofrece son iguales a la primera versión pero incluye además otras nuevas. Además de mejorar los consejos y trucos sobre reciclaje, que son renovados mensualmente, esta versión incluye curiosidades que pueden resultar no solo pedagógicas sino también lúdicas, pudiendo seleccionar el tema sobre el que recibir dicha información. El usuario tiene también la posibilidad de aportar, a través de correo electrónico, nuevos consejos que podrán ser posteriormente incluidos según el criterio del equipo de administradores.

Recyclate! agrega la opción de aprender jugando, mediante el juego *ReciclaDOS*, aunque cabe la posibilidad de añadir más juegos en el futuro. El juego permite establecer al usuario la configuración que desee: sonido, vibración, lectura de preguntas, etc.

El juego permite al usuario registrarse, teniendo así un perfil donde podrá comprobar sus datos relativos a los juegos. Si el usuario se ha registrado, la aplicación guarda la sesión abierta, para no tener que volver a acceder la próxima vez que se desee jugar. Si no se realiza el registro se puede jugar de forma anónima. Al finalizar el juego el usuario puede publicar su puntuación en Facebook, Twitter y Google+.

Herramientas

El servidor fue alojado en la Facultad de Matemáticas de la UCM, disponía de Windows server 2003, SQL Express 2005, PHP y Apache.

Se utilizaron diversas librerías para dotar de funcionalidad a la aplicación, hacerla más atractiva y fácil de usar:

- Librería ViewPager para añadir efectos visuales.
- Librería twitter4j-core-3.0.3 que permite crear Tweets que serán publicados en la cuenta personal del usuario.
- Otras librerías de las que Recycla.me hace uso.

Se hizo uso del SDK de Facebook para Android para gestionar la conexión de la aplicación con la red social. También se empleó Visual Estudio 2012 para la herramienta de escritorio en C# destinada a los administradores. Recyclate! es algo más compleja en lo que a la utilización de la base de datos se refiere, ya que se hace uso de dos bases de datos, ambas de tipo SQL. La primera es una base de datos externa, alojada en un servidor, y la segunda una interna alojada en el propio dispositivo del usuario.

Objetivos

En la situación inicial en la que se encontraba el proyecto era necesario realizar una serie de cambios en tres o cuatro de las aplicaciones, desarrolladas en años anteriores relativas al área de medio ambiente. Esta necesidad de modificar los proyectos pasados atendía a varios motivos:

1. La aparición de errores y fallos debido a que algunos de los recursos utilizados en el curso anterior estaban obsoletos o alojados en servidores que no permanecían activos, lo que obligaba a buscar alternativas más actuales y tener que migrar los recursos que no estaban disponibles para que fueran perfectamente accesibles.
2. Dos de las aplicaciones que se nos encomendó mejorar (Recycla.me y Recyclate!) realizan funcionalidades muy parecidas, pero tanto su modelos de datos como las capas de

presentación desarrolladas en Android tenían diferencias notables. Una de las metas a alcanzar fue la de unificar la estructura y los recursos que nutren estas dos aplicaciones, utilizando un único servidor y una única base de datos para ambas.

3. En las primeras reuniones mantenidas con el personal del IAM se nos informó de que estas aplicaciones no cumplían los estándares que precisaba la institución. Gran parte del cometido de este proyecto es el cumplimiento de estos requisitos, haciendo estos proyectos escalables y dejándolos listos para futuras mejoras.

Las medidas que se han ido tomando a lo largo del desarrollo del proyecto en aras de solventar estos problemas se detallan más adelante en esta memoria.

Alternativas multiplataforma

Como ya se ha comentado en la introducción de este apartado, uno de los objetivos finales del proyecto era conseguir que las aplicaciones estuvieran también disponibles para usuarios del sistema operativo iOS, pero ante la emergencia de nuevos sistemas operativos como Firefox OS y Windows 8, se decidió no limitar los desarrollos a sistemas operativos determinados. Nuestro objetivo es que estas sean accesibles, de igual manera, independientemente del sistema operativo.

En este punto estudiamos las posibilidades que se presentan. Hoy día, existen tres tipos de aplicaciones, si se clasifican según la forma en que se aborda su desarrollo. Estos tres tipos son:

- Nativas: íntegramente programadas en el entorno de desarrollo específico de cada sistema operativo.
- Web: completamente desarrolladas en HTML 5 y accesibles mediante un navegador web.
- Híbridas: una mezcla de ambas.

Cada una de ellas tiene sus aspectos positivos y negativos [21-22] los cuales fueron previamente estudiados antes de decantarnos por el tipo de aplicación que finalmente se ha desarrollado. La cualidad principal de las aplicaciones web es que son accesibles desde cualquier tipo de plataforma lo que supone una gran ventaja respecto a las aplicaciones nativas, que necesitan un desarrollo para cada sistema operativo. Por contra, dado que su funcionamiento se basa en recursos web, las aplicaciones web necesitan una muy buena conectividad para funcionar de forma óptima, e incluso algunas de ellas no podrán funcionar sin conexión. En nuestro caso, no podemos permitir que estas aplicaciones dejen de funcionar sin conexión, ya que una de las ventajas de Itinerarios guiados Parque del Retiro, Recycla.me y Recyclate! es que una buena parte de la funcionalidad que ofrecen está disponible cuando el dispositivo no tiene acceso a Internet. Otro punto a favor de las aplicaciones nativas es la compatibilidad con todas las funcionalidades que ofrecen los dispositivos. Una aplicación web podría encontrar dificultades (en algunos casos insalvables) a la hora de utilizar recursos como, por ejemplo, la cámara del móvil o la geolocalización GPS. Tras estudiar estas características, parece que las aplicaciones nativas se ajustan más a lo que el proyecto requiere. El problema es que en un futuro no se pueden limitar estas apps a solo uno o dos sistemas operativos, sino que se tendrán que adaptar a las distintas opciones que presente el mercado.

Por lo tanto, en un primer momento se decidió que el proyecto se decantaría por el modelo de aplicaciones híbridas, es decir, aplicaciones originalmente programadas en HTML5, que serán "empaquetadas" para ofrecer el 100% de su funcionalidad para los sistemas operativos Android e iOS. De esta manera, se ofrece la aplicación puramente web para usuarios de otros sistemas operativos, existiendo la posibilidad en un futuro de empaquetarlas para determinadas plataformas.

En conclusión, con esta decisión se pretende cumplir con los requisitos que fueron planteados y ofrecer unas aplicaciones adaptables a los cambios que se produzcan en el mercado en el futuro.

A continuación se muestran las diferentes opciones estudiadas para hacer aplicaciones multiplataforma:

Phonegap [23]: Permite empaquetar el código como aplicación nativa para distintas plataformas: iOS, Android, Blackberry, Windows Phone, etc, partiendo del mismo código base creado usando las tecnologías HTML5, CSS3 y

JavaScript. Este *framework* pone a nuestra disposición una serie de librerías que, incrustadas en las plataformas de desarrollo de cada uno de los fabricantes (SDK), permiten programar a través de su API funcionalidades del SDK nativo. Fue creado por Nitobi pero ahora es distribuido por Adobe Systems, siendo una de las alternativas de desarrollo más conocidas y que mejores críticas recibe. El *framework* soporta geolocalización, vibración, acelerómetro, cámara, cambio de orientación, magnetómetro y otras interesantes características.

Icenium [24]: Es una plataforma de desarrollo en la nube que permite desarrollar aplicaciones para iOS y Android mediante estándares web (HTML5, CSS3 y JavaScript). Una de sus facetas más interesantes es que hace que nos olvidemos de las SDK de cada una de las plataformas. Mediante su propio IDE de desarrollo se pueden crear las aplicaciones con las tecnologías mencionadas.

Appcelerator [25]: Al igual que Icenium, posee su propio IDE de desarrollo, mediante el cual podemos programar funcionalidades comunes a las distintas plataformas sin tener que crearlas de manera específica y en lenguajes diferentes. Se caracteriza por su sencilla forma de integrar la información de múltiples servicios en nuestras aplicaciones.

Capítulo 2. REQUISITOS SOFTWARE

En este capítulo se hace una descripción más detallada sobre las tecnologías que se han utilizado para desarrollar este proyecto. Como ya se mencionó anteriormente, la elección de estas herramientas está basada en las recomendaciones que nos hicieron los responsables del IAM (Informática del Ayuntamiento de Madrid), teniendo también en cuenta criterios de rendimiento y compatibilidad con el material que se recibió de años pasados. A continuación se muestran algunas de las características más importantes que presentan cada una de estas tecnologías.

2.1 Maven

La primera de las tecnologías que se usa es Maven [26-27], ya que es la encargada de gestionar todo el proyecto. Maven es una herramienta de Apache que surge de la idea de que multitud de proyectos Java se enfrentan a necesidades y dependencias muy similares entre sí, especialmente a la hora de compilar aplicaciones, como la generación del código y las fuentes, la compilación de las clases Java y de testeo, los empaquetamientos de código en archivos JAR (Java Archive), WAR (Web Archive), EAR (Enterprise Archive),…, además de la posible inclusión de librerías externas.

Una de las principales ventajas de Maven es que facilita bastante el proceso de construcción de proyectos Java, así como la gestión de sus dependencias y *plugin*s. No elimina la necesidad de conocer las tecnologías y mecanismos propios del proyecto, pero evita tener que enfrentarse a multitud de pormenores relacionados con estas. Para conseguirlo, Maven permite construir proyectos haciendo uso del POM (Project Object Model), un conjunto de *plugins* compartido por todos los proyectos Maven, que ofrece un sistema de construcción unificado y estandarizado.

En los proyectos Maven, las dependencias, las construcciones y los artefactos deben tener una descripción. Esta descripción se realiza a través de un archivo XML (Extensible Markup Language): el POM. Este se usa para indicar a Maven como debe realizar sus tareas y modificar su comportamiento por defecto. Mediante el uso del POM y de sus propiedades de herencia obtenemos una mayor reutilización tanto de configuraciones como de uso de *plugin*s. Podemos dividir un POM en cuatro apartados como se aprecia en la *Figura 2.1.*

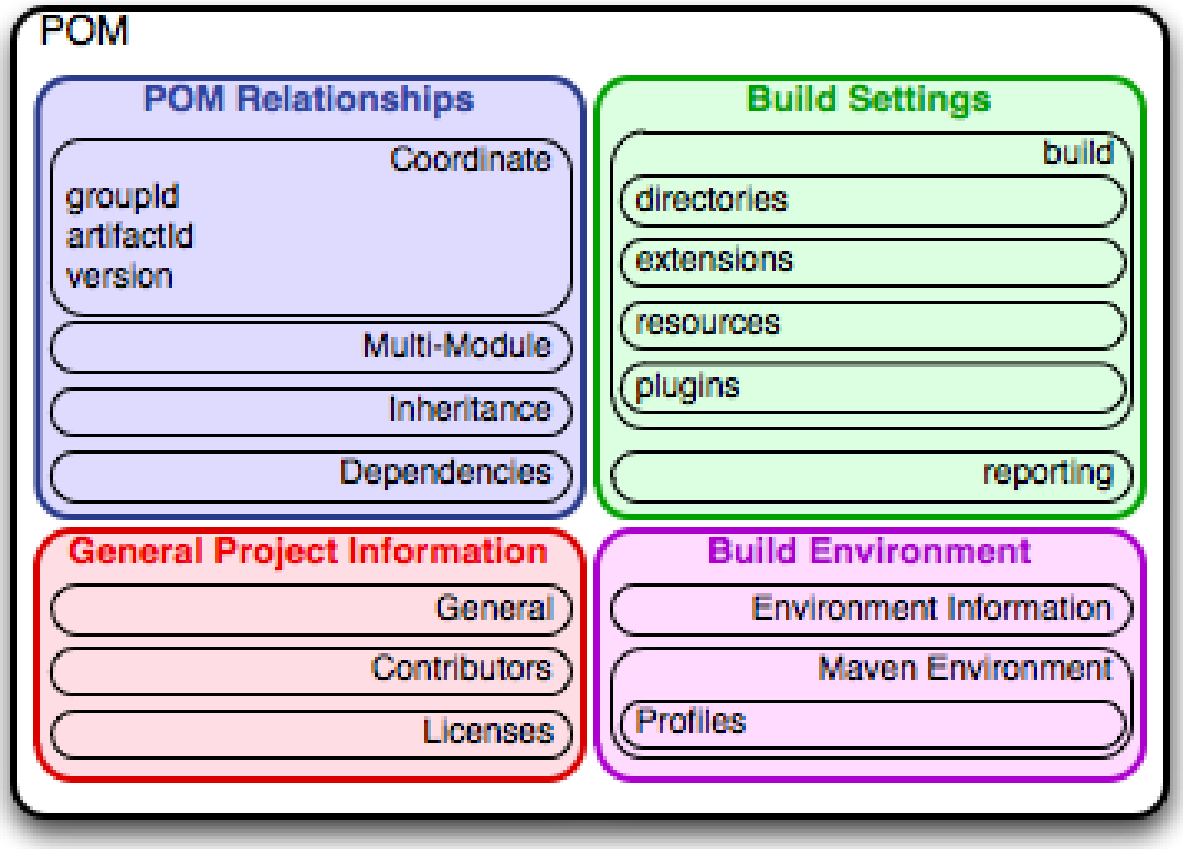


Figura 2.1.- Estructura de un POM

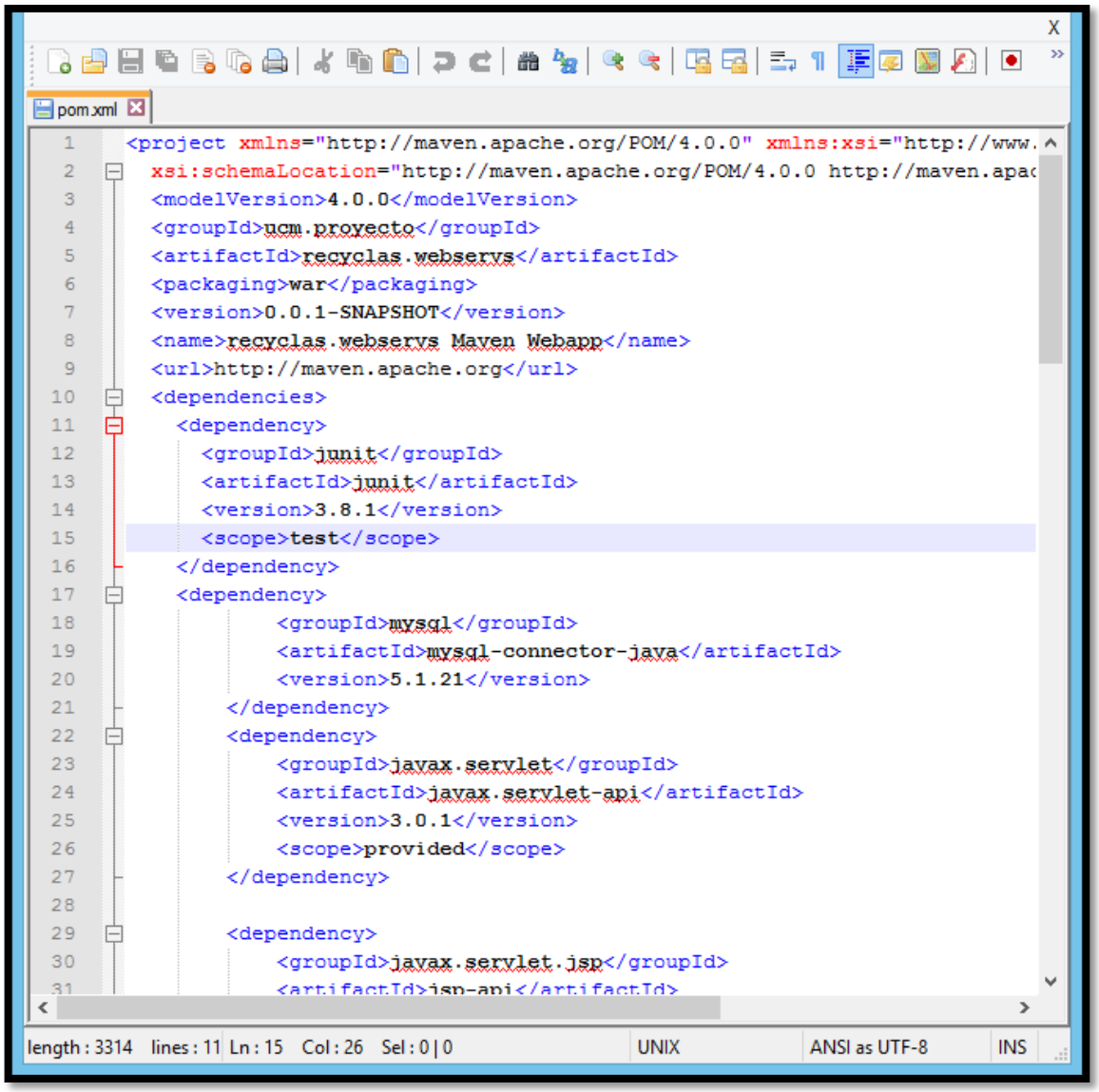


Figura 2.2.- Detalle del POM

POM Relationships: Este es el apartado de configuración de las relaciones y dependencias del módulo con otros proyectos. Aquí se indican los distintos módulos que pueden formar parte del proyecto, la información de herencia de POMs (en caso de que se trate de submódulos de un proyecto) y, por último, las distintas dependencias del proyecto, como se puede ver en la *Figura 2.2*. Las dependencias de un proyecto pueden tener distinto ámbito:

- *compile*: Dependencias requeridas en la compilación y que serán empaquetadas.
- *provided*: Necesarias para compilar pero no empaquetadas ya que son proporcionadas por el contenedor donde será desplegada la aplicación. Por ejemplo, si se quiere desplegar en un servidor de aplicaciones se necesita una dependencia con el Servlet API para compilar, pero esta debe ser proporcionada por el contenedor del servidor, así que se marcará como *provided*.
- *runtime*: No son empaquetadas pero son necesarias para la compilación y fase de pruebas.
- *test*: Necesarias para la fase de test del proyecto, no empaquetadas.

Build Settings: En este apartado se personaliza el comportamiento por defecto de Maven, pudiendo configurar los *plugin*s de control de las distintas fases del ciclo de vida o generar informes acerca del proyecto.

General Project Information: Este apartado incluye información general del proyecto como la descripción, los desarrolladores, los responsables, las licencias utilizadas para el desarrollo, etc.

Build Environment: En este apartado se configura el entorno para la ejecución de Maven, pudiendo configurar distintos perfiles que posteriormente podrán ser activados durante la ejecución de Maven. Aquí se definen los repositorios en los que Maven buscará tanto las dependencias que necesita el proyecto, como los artefactos (serán presentados más adelante), además de poder crear diferentes perfiles de configuración para su rápida reutilización.

Otra de las ventajas más importantes que ofrece Maven es la estandarización tanto de la estructura de directorios, como del ciclo de vida del proyecto. Hasta ahora cada proyecto tenía una estructura diferente según los estándares internos de la empresa o del equipo de desarrollo, haciendo

así que proyectos que hacían uso de las mismas tecnologías estuvieran organizados de formas completamente diferentes, con archivos comunes al tipo de proyecto en diferentes ficheros. Maven propone una misma estructura para todos los proyectos (pero permitiendo que pueda configurarse de forma personalizada en caso necesario) mediante el uso de sus arquetipos. Un arquetipo de Maven es una plantilla parametrizada que permite crear el esqueleto de un proyecto de una forma rápida y sencilla, creando su estructura de ficheros de forma estandarizada y obteniendo sus dependencias básicas y *plugin*s ya configurados. Todo esto es modificable, y tenemos la opción de crear nuestros propios repositorios y desarrollar nuestros propios arquetipos, además de los que ofrece Maven y muchas otras empresas que se dedican a este tipo de desarrollos (*Third-parties*).

También se encontraron grandes diferencias entre los ciclos de vida de los distintos proyectos, multitud de fases diferentes, con su nombre, orden y requisitos para cada uno de ellos. Un ciclo de vida es una secuencia ordenada de pasos para conseguir un objetivo, Maven nos proporciona tres:

- Clean (Limpieza): Este ciclo de vida es el encargado de borrar todos los artefactos generados durante la construcción del proyecto (por defecto ${basedir}/target). Este ciclo de vida es muy importante ya que nos asegura que tras ejecutarse, el proyecto vuelve a quedar en un estado limpio. Se ejecuta mediante el comando *mvn clean*.

- Build (Construcción): Este ciclo de vida es el encargado de construir el proyecto. Consta de numerosas fases por lo que solo vamos a describir las más comunes. La enumeración está ordenada desde las fases iniciales a las finales y la ejecución de una fase implica ejecutar todas las anteriores, así por ejemplo al ejecutar la fase *package* se ejecutará a su vez la fase compile pero no la *install*.

    - La fase *compile* se encarga de compilar el código fuente del proyecto. Para ejecutarla se debe invocar el comando *mvn compile*.

    - La fase *package* es la encargada de empaquetar los distintos artefactos del proyecto. Es decir, esta fase se encarga de construir los JAR, WAR y EAR. Se ejecuta mediante el comando *mvn package*.

- La fase *install* se encarga de instalar los artefactos generados dentro del repositorio local para poder ser usado como dependencia en otros proyectos locales. Para ejecutarla se debe invocar el comando *mvn install*.

- Site (Generación de informes): Este ciclo de vida es el encargado de generar la documentación e informes del proyecto (Javadoc, Checkstyle, etc.). La fase más común dentro de este ciclo de vida es *site-deploy*. Esta genera los informes de todos los módulos y además realiza el despliegue de la documentación. Se ejecuta mediante el comando *mvn site-deploy*.

Algunos de los inconvenientes que encontrados al usar Maven son:

- La dificultad intrínseca a su propio uso, es decir, la curva de aprendizaje de Maven es bastante alta, y su correcto uso y dominio requieren mucha dedicación y esfuerzo. Para proyectos simples en cuanto a dependencias y módulos o en los que no se busquen configuraciones complejas para los ciclos de vida no es excesivamente complicado, pero en nuestro caso, la correcta gestión de todo el proyecto implicaba bastante manejo de la herramienta.

- Cierta dificultad a la hora de depurar el código, ya que debido a la ocultación de pormenores de las fases de construcción, muchas veces es difícil conocer la causa del problema; aunque esto es fácilmente solventable gracias a la cantidad de documentación y soporte que hay en la red para esta comunidad.

- La dificultad a la hora de elegir la versión correcta para el proyecto, ya que ciertos *plugin*s y repositorios daban problemas dependiendo de la versión utilizada.

2.2 Modelo de tres capas

Antes de pasar a analizar el resto de tecnologías utilizadas, es conveniente ponerlas en un contexto adecuado. La elección de las herramientas que se describe a continuación se ha tomado con el fin de adaptar nuestros sistemas a una arquitectura basada en un modelo de tres capas [28-29]. El objetivo de este modelo es lograr la máxima abstracción e independencia

entre las mismas y hacer que distintos proyectos puedan llegar a compartir algunas de ellas.

La definición de los límites de cada capa permitirá determinar correctamente la colaboración que proporcionará cada una de ellas. Esto hace algo más complicado el desarrollo de las aplicaciones, pero dará lugar a una infraestructura robusta y lista para su extensión, y crecimiento como proveedora de servicios. Además, esta arquitectura facilita que el mantenimiento de los sistemas sea mucho más sencillo y modular.

Las 3 capas que se tratan son:

- **Capa de presentación**: Muestra la información al usuario final de la aplicación.
- **Capa de negocio**: Encargada de la lógica de negocio del sistema.
- **Capa de persistencia**: Interactúa con la base de datos.

En la *Figura 2.3* se muestra una visión general del modelo de tres capas. Además mostraremos que *framework* utilizamos para implementar cada una de las diferentes capas del sistema.

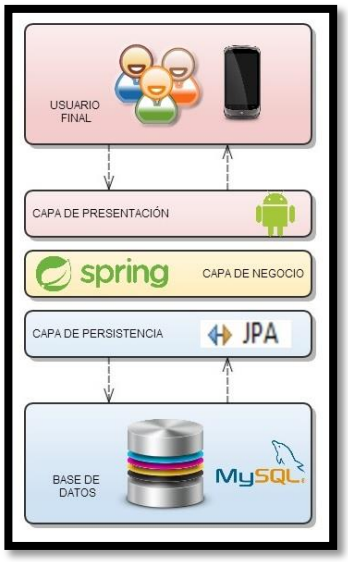


Figura 2.3.- Esquema de modelo de tres capas

En el diagrama de la *Figura 2.3* podemos ver representada una de las grandes ventajas de esta arquitectura: una capa solo se relaciona con su capa inmediatamente inferior o superior, impidiendo por ejemplo que desde la aplicación se pueda acceder a la base de datos, sin pasar por los controladores. Esto proporciona, además de las ventajas ya comentadas, mejoras muy importantes en todo lo relacionado a la seguridad de los datos y a la estabilidad de la base de datos.

A continuación se explica con detalle con cada una de las tecnologías utilizadas atendiendo a que capa de la aplicación corresponden.

Capa de presentación - Android

La capa de presentación es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura los datos introducidos por el usuario en un mínimo proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y debe tener la característica de ser amigable (comprensible y fácil de usar) para el usuario.

La capa de presentación en el proyecto se corresponde con las aplicaciones Android, una tecnología respecto a la cual el Departamento de Calidad del IAM está aún preparando los estándares que deberán cumplir los futuros desarrollos para dispositivos con este sistema operativo. Es por eso que en esta capa simplemente se han modificado las aplicaciones, desarrolladas por nuestros compañeros en años anteriores, en la forma que tienen estas de relacionarse con los servicios web, pero respetando sus diseños y funcionalidades. Además se han solucionado diferentes fallos detectados durante su funcionamiento.

En el próximo capítulo de la memoria entraremos a enumerar más detalladamente las modificaciones específicas realizadas para esta capa en cada una de las aplicaciones.

Capa de negocio – Spring Framework

La capa de negocio expone la lógica necesaria a la capa de presentación para que el usuario, a través de la interfaz, interactúe con las funcionalidades de la aplicación. La capa lógica de negocios ocupa un lugar preeminente en la construcción de una infraestructura de software, al permitir el crecimiento y la extensión de servicios para todas las aplicaciones existentes y futuras. En este proyecto esta capa está implementada con el *framework* Spring [30].

Spring es un *framework* para desarrollo de aplicaciones y contenedor de inversión de control, de código libre para Java. El núcleo de sus características puede ser usado por cualquier tipo de aplicación, pero existen extensiones, como la utilizada en este proyecto, para desarrollar aplicaciones web sobre la plataforma Java EE. Aunque el *framework* Spring no impone un modelo de programación específico, su uso se ha popularizado como una alternativa al modelo EJB (Enterprise JavaBean) [31] ya que nos permite crear una estructura de modelo de negocio de una manera muy sencilla a través de ficheros XML y anotaciones (en versiones actuales). Se encarga de instanciar los objetos que forman parte de nuestra lógica (los llamados JavaBeans) y de enlazar unos con otros, todo dentro de un contexto de ejecución, desde el cual son accesibles para el resto de la aplicación.

Simplificando mucho, se puede decir que existe una serie de clases que se encargan de iniciar el contexto de Spring en una aplicación JEE. Estas clases leen unos ficheros de configuración (por lo general en formato XML) en los que se define de qué están compuestas y cómo hay que instanciar las clases y ponerlas a disposición del cliente.

La arquitectura básica que define Spring para el acceso a datos se basa en:

- Una clase modelo que mapea los datos.
- Una clase DAO (Data Access Object) [32-33] que se comunica con la base de datos y realiza las operaciones sobre el modelo.
- Una clase servicio que ofrece los métodos funcionales de la lógica.

Para que Spring sepa a cuál de estos tres tipos corresponde cada clase del proyecto, se han de configurar dichas clases bien vía archivo XML, o bien (en las versiones más recientes) mediante anotaciones en la propia clase. En este caso se ha decidido optar por esta segunda posibilidad, ya que es mucho más intuitiva a la hora de programar y, sobretodo, a la hora de comprender el código para alguien que tenga que trabajar con la aplicación web en el futuro. Para ello, se hizo uso de la versión *3.2.8.release*.

Para cada una de las clases arriba listadas se usan las anotaciones *@Entity*, *@Repository* o *@Service*, respectivamente. Estas anotaciones se sitúan justo antes de la declaración de la clase, como se muestra en la *Figura 2.4* y hacen que el archivo applicationContext.xml quede mucho más simplificado como se puede ver en la *Figura 2.5*.

Figura 2.4.- Detalle de la capa DAO

Figura 2.5.- Detalle del archivo applicationContext.xml

En la *Figura 2.5* también se muestra la sentencia *<mvc:annotation-driven />*, esta no solo hace saber a Spring que ha de buscar en cada clase las

anotaciones que comienzan por @, sino que además indica que estamos siguiendo la plantilla Spring MVC.

Esencialmente, esto quiere decir que Spring permitirá que sea una nueva clase, los controladores, los que reciban y manejen las peticiones que le llegan al servidor. En la siguiente sección se explica este aspecto con más detalle.

Spring MVC

Spring MVC [34] es una implementación del patrón Modelo-Vista-Controlador [35] basada en Spring. El *framework* Spring MVC está diseñado alrededor de un *DispatcherServlet* (que se definirá en el archivo web.xml de nuestra aplicación web), clase que remite las distintas peticiones que recibe el servidor hacia la clase a la que le corresponde encargarse de las mismas. Estas clases que se encargan de manejar las peticiones son las denominadas controladores (*@Controller*), mediante la notación *@RequestMapping* el *DispatcherServlet* sabe a qué método de qué controlador debe invocar cada petición. Este método se encarga de llamar a los servicios necesarios para cumplir las tareas requeridas. Se puede ver un ejemplo aclarativo en la *Figura 2.6*.

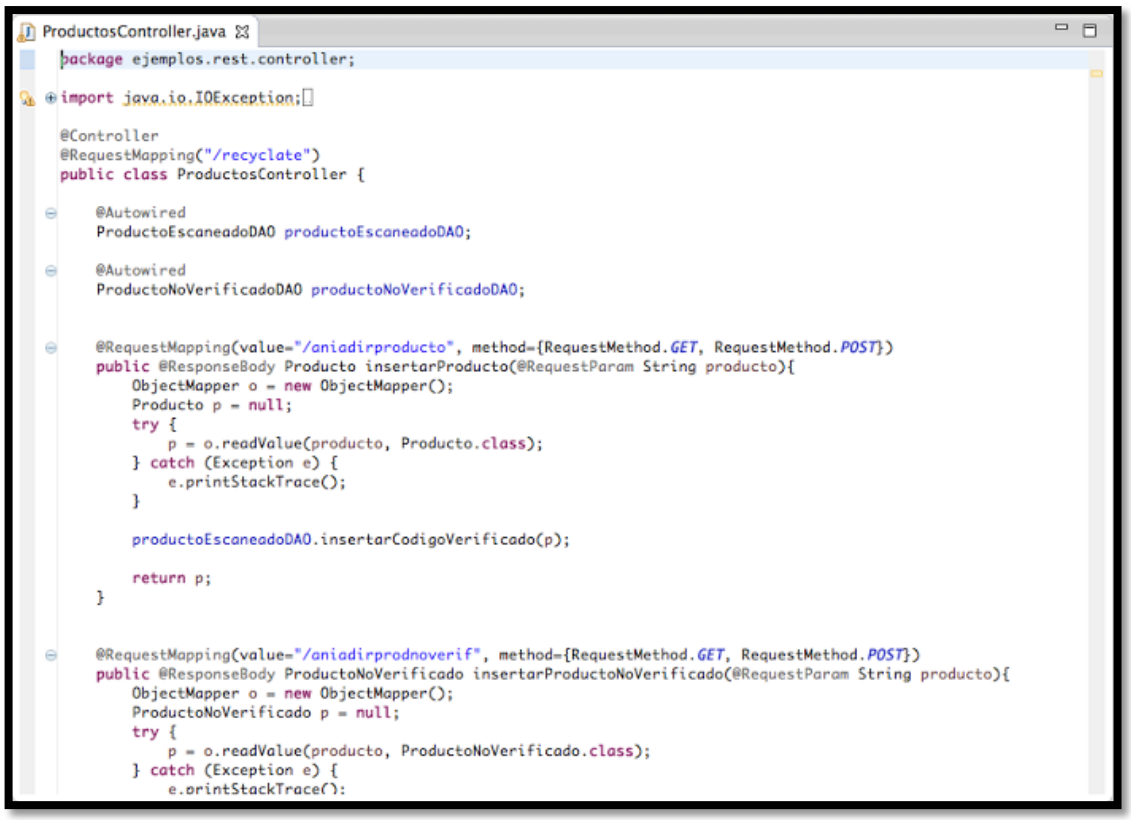


Figura 2.6.- Detalle de la clase controlador

En la *Figura 2.6* se puede observar que esta clase se trata de un controlador, ya que la anotación *@Controller* se encuentra encima de la cabecera de la clase. Así como varios *@RequestMapping*, que se encargarán de redireccionar las peticiones realizadas a las distintas URLs hacia sus correspondientes métodos dentro de la clase. Además, los métodos aceptan tanto peticiones GET, como POST, aunque podrían restringirse para que aceptaran solo una de las dos peticiones. Especial mención merecen dos nuevas anotaciones que resultan muy importantes para el *framework* Spring MVC: *@RequestParam* y *@ResponseBody*. Normalmente las peticiones no solo llaman a una URL, y esta ejecuta los pasos necesarios, siendo por eso que aparecen estas nuevas las anotaciones. *@RequestParam* se encarga de que solo se acepten peticiones que vengan acompañadas de unos atributos de entrada. Estos parámetros, en este caso, vienen en formato JSON y es el propio Spring (mediante la librería Jackson, incluida en el proyecto) el que los analizará para generar un objeto del tipo deseado.

De igual manera y como es normal, se espera una respuesta por parte del servidor. De eso se encarga *@ResponseBody*, que realiza la operación opuesta al proceso anteriormente mencionado, es decir, toma un objeto (o

lista de objetos) y lo trata, convirtiéndolo en un JSON que devolverá la información requerida a la capa de presentación.

Como también se puede apreciar en la *Figura 2.6* aparece la anotación *@Autowired*. Este es un claro ejemplo de la gestión de JavaBeans por parte del *framework* Spring. Lo que hará *@Autowired* será buscar si ya se ha creado una *bean* del tipo indicado (en el ejemplo *ProductoEscaneadaDAO*), y si es así lo igualará a ese atributo de la clase. En caso de que no haya ninguna *bean* de ese tipo creada en el momento de acceso a la clase, será el propio Spring quien se encargará de ejecutar las constructoras correspondientes. Si por otro lado se quisiera que la aplicación web maneje a la vez varias *Beans* del mismo tipo, además de usar *@Autowired*, se debe utilizar la notación *@Qualifier*("nombre") para diferenciar una de la otra.

En conclusión el *framework* Spring, y en particular el modelo Spring MVC, ofrece una manera sencilla de realizar aplicaciones web muy potentes, robustas y seguras. Es una tecnología que aunque ha llegado a España hace relativamente poco tiempo, en otros países lleva mucho tiempo siendo utilizada por un importante número de grandes empresas internacionales.

Capa de persistencia – JPA

La necesidad de vincular los datos guardados en una base de datos relacional, con los objetos de una aplicación orientada a objetos, determinó la aparición del concepto de persistencia de objetos. Siguiendo el estilo de desarrollo en tres capas, la persistencia queda recogida en su propia capa, separada de la lógica de negocio y de la interfaz de usuario.

Para esta capa se ha utilizado JPA (Java Persistence API) [36-38], API de persistencia desarrollado para la plataforma Java EE. El objetivo de utilizar este API es tener la posibilidad de mapear en objetos java los datos de un recurso y realizar tareas de obtención y persistencia de esos datos de una manera cómoda y sin tener que generar una gran cantidad de código.

Como JPA es un API es necesaria una implementación con la que trabajar. Las más utilizadas son EclipseLink, Hibernate, OpenJPA o

TopLink [39]. Después de estudiar distintas comparativas, y basándonos en algunas nociones previas, se optó por la plataforma Hibernate.

Hibernate [40] es probablemente el ORM (Object Relational Mapping) más utilizado, y su uso ha estado muy extendido también al margen de JPA. Las ventajas de esta API es que la configuración es bastante más sencilla que utilizando solo Hibernate, ya que solo tienen que configurarse las propiedades deseadas dentro del fichero *persistence.xml* (el fichero estándar de configuración para JPA).

Los desarrolladores de Hibernate incluyen varias capas de abstracción en su software para hacerlo compatible con JPA. Además, uno de los puntos fuertes de Hibernate respecto a otras implementaciones de JPA es que este software ha sido sobradamente probado en entornos reales y, por tanto, su rendimiento está mucho más optimizado.

En la *Figura 2.7* se puede ver cómo se ha configurado el fichero *persistence.xml* para utilizar la implementación Hibernate.

Figura 2.7.- Detalle del archivo persistence.xml

Se puede observar que es una configuración bastante sencilla. Este archivo define una unidad de persistencia (*persistence-unit*), que es utilizada durante toda la ejecución para realizar las transacciones con la base de datos. Para esta unidad de persistencia sólo es necesario definir un proveedor de la implementación JPA del manager de entidades (en este caso, Hibernate), definir las propiedades de la conexión con el servidor de

base de datos y especificar las clases a las que se mapearán las diferentes entidades de la base de datos (este último punto podría resolverse también solo con anotaciones). Una vez explicado que es JPA, resultará mucho más sencillo ver cómo funciona y cómo se integra con nuestra aplicación Spring.

El primer paso es definir las clases que representarán las entidades JPA. Esta parte del desarrollo no varía mucho según la implementación de JPA que se esté usando. Todas las entidades JPAs son POJOs (Plain Old Java Objects) [41], una "nueva palabra para definir algo viejo". Un POJO es un objeto Java simple, que no extiende ni implementa ninguna otra clase, y que solo cuenta con constructoras, sus atributos y sus métodos de obtención y establecimiento, aunque también pueden tener algunos métodos auxiliares. El objetivo es mapear las entidades de la base de datos, convirtiéndolas en uno de estos POJOs mediante el uso de JPA, el cual ofrece la posibilidad de realizar este proceso mediante anotaciones, facilitando mucho la labor de mapeo.

El propio JPA puede generar estas clases automáticamente, creando una por cada tabla de la base de datos referenciada y un atributo por cada columna de la tabla. Pese a esto, se decidió que hacer estas clases por nosotros mismos ayudaría a aumentar el control sobre las mismas aspirando a que el funcionamiento de la aplicación sea óptimo. En el proyecto, las clases que representan las entidades JPA son las que se encuentran en el paquete modelo, de las que la *Figura 2.8* muestra un ejemplo.

Figura 2.8.- Detalle de la clase ProductoVerificado.java

En la *Figura 2.8* se pueden ver las diferentes anotaciones propias de JPA que ayudan a relacionar esta clase con las entidades correspondientes. *@Entity* es la anotación más obvia, pero a la vez la más importante ya que hace saber tanto a Spring como al propio Hibernate que se está tratando con una entidad JPA. Justo a continuación se encuentra la anotación *@Table (name = "nombre")*, que relaciona a la clase en cuestión con la entidad correspondiente de la base de datos. El nombre deberá ser exactamente el que tenga la tabla referenciada en la base de datos. Una vez que el POJO está relacionado con una tabla de la base de datos, solo queda relacionar cada uno de los atributos del POJO con las columnas de la clase, lo cual se consigue mediante la anotación *@Column (name = "nombre")*, en la que el nombre deberá ser exactamente el que tenga la columna de la tabla deseada en la base de datos.

Por último, solo queda definir qué atributo, o atributos, de la clase se corresponde con la clave primaria de la entidad, siendo suficiente con anotar *@Id* en los atributos deseados. Además JPA (mediante la anotación *@GeneratedValue*) se encarga por sí mismo de autoincrementar el id de una

nueva entidad cuando se quiera introducir una nueva entrada en la base de datos.

Una vez definidas las clases que se relacionan con cada entidad, solo falta explicar cómo leer y dar persistencia a estas entidades en la base de datos. Este paso varía dependiendo de la implementación de JPA que se haya utilizado para la implementación (sobre todo a la hora de realizar las consultas de acceso a base de datos). A continuación se detallan los pasos a seguir utilizando la implementación Hibernate.

Siguiendo con la arquitectura básica que define el *framework* Spring, este acceso a base de datos se realizará en las clases DAO correspondientes. Cuando se hace referencia a clases DAO, se hace referencia a clases que implementan el patrón del mismo nombre. Este patrón de Objeto de Acceso a Datos (Data Access Object) se usa 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, de una manera eficiente y fácilmente adaptable a cambios en el origen de los datos.

En el Capítulo 3 de esta memoria describiremos más profundamente los desarrollos específicos de este proyecto, por lo que aquí se detallan sólo los principios básicos de acceso a la base de datos.

Antes de iniciar cualquier transacción es necesario definir un manager de entidades (clase *javax.persistence EntityManager*), al se le indicará a qué unidad de persistencia debe referenciar, como se muestra en la *Figura 2.9.*

```
private void inicializarEM(){
    entityManagerFactory = Persistence.createEntityManagerFactory("testunit");
    entityManager = entityManagerFactory.createEntityManager();
}
```

Figura 2.9.- Método para inicialización del Entity Manager

Una vez inicializado este EntityManager, simplemente hace falta llamar a los métodos propios de esta clase para modificar, leer o introducir un nuevo método en la clase de datos. Algunos de los métodos más representativos son:

- *getTransaction().begin()*: Se crea una nueva conexión para tratar con la BBDD.
- *getTransaction().commit():* Hace efectivos los cambios realizados en la base de datos.
- *persist(POJO)*: Introduce en la base de datos una entidad del tipo al que equivale el POJO.
- *getDelegate().get(CLASE POJO, clave_primaria):* Hace una búsqueda por *clave_primaria* en la BBDD y devuelve un objeto de la clase determinada.

Una vez se han realizado estos dos sencillos pasos ya se tiene en funcionamiento la conexión con la base de datos, por lo que, a modo de conclusión, se puede decir que utilizar JPA simplifica enormemente la programación de la lógica de persistencia. Su uso proporciona grandes beneficios como son la independencia de la base de datos, su bajo acoplamiento entre las capas de negocio y persistencia y un rápido desarrollo. Además, el código generado es muy comprensible y fácil de mantener.

Es fundamental conocer bien cómo funcionan las tecnologías que se utilizan ya que de ellas dependen gran parte del proceso de desarrollo, incluso puede llegar a afectar directamente al rendimiento de la aplicación. Valorado esto, en nuestra opinión el uso de JPA ha sido muy beneficioso ya que se ha conseguido una aplicación que se comunica de manera eficiente con la base de datos, además de ser muy robusta y modulable.

Base de datos - MySQL

Una vez explicado el funcionamiento de la capa de persistencia es el momento de hablar de la construcción y la gestión de la Base de Datos. En este caso.

Para construir dicha base de datos se ha seguido el modelo relacional, siendo este el modelo más extendido para su diseño y administración. Como se introdujo brevemente en el punto anterior, la representación gráfica de este modelo es una tabla. Una tabla (entidad JPA, POJO) se compone de filas (una instancia de ese POJO) y columnas (los atributos de ese POJO).

Las filas se corresponden con los registros y las columnas se corresponden con los campos, siendo estos la unidad mínima de información.

Para gestionar la base de datos hemos utilizado MySQL [42], un sistema de gestión de bases de datos relacional, multihilo y multiusuario. Sin duda es uno de los gestores de bases de datos con más usuarios en el mundo a día de hoy y su gran popularidad se debe, entre otros factores, a que se distribuye bajo licencia GNU GPL [43] (es decir, es software libre), a su facilidad de uso y de puesta de marcha, así como a su gran rendimiento en entornos de producción con alta carga de trabajo.

Además si lo que se pretende es diseñar una base de datos sencilla, MySQL pone a disposición de los desarrolladores una serie de entornos gráficos que facilitan esta tarea. Dos de los entornos gráficos más utilizados son MySQL Workbench (desarrollado por la propia MySQL AB) o PhpMyAdmin. El equipo de desarrollo de este proyecto se decantó por este último, PhpMyAdmin ya que se trata de la solución más eficiente a la hora de diseñar la base de datos y conectarla con la aplicación web.

PhpMyAdmin [44] es una herramienta escrita en PHP con el propósito de manejar la administración de MySQL a través de páginas web utilizando Internet. Actualmente puede crear y eliminar Bases de Datos; crear, eliminar y alterar tablas; borrar, editar y añadir campos; ejecutar cualquier sentencia SQL; administrar claves en campos y privilegios; exportar datos en varios formatos. Se encuentra disponible en 62 idiomas bajo la licencia GPL.

En el Capítulo 3 de esta memoria se detalla más concretamente el diseño de la base de datos de nuestro proyecto.

Capa de integración Cliente-Servidor – Servicios Web RESTful

Por último, es conveniente dedicar la última sección de este capítulo a lo que se puede denominar la cuarta capa de la arquitectura, la capa de integración [45]. Aunque no representa una capa como las tres expuestas anteriormente, la capa de integración se encargará de poner en contacto el cliente con el servidor. Hasta ahora se ha hablado de que la capa de

presentación y la de negocio se comunican mediante peticiones y respuestas, pero no se ha explicado cómo se realizan esas peticiones.

Para esta comunicación entre cliente y servidor se han utilizado servicios web REST (Representational State Transfer). Esta técnica ha ido ganando popularidad como una alternativa a los servicios web SOAP, y ya en 2008 grandes proveedores de web 2.0 como Google, Yahoo! o Facebook comenzaron a migrar hacia esta tecnología, consiguiendo desplazar a SOAP al ser su uso mucho más sencillo.

Los 4 principios de REST son:

- Utiliza los métodos HTTP de manera explícita. Esta es una de las características principales de los servicios web REST. Las aplicaciones cliente utilizan los métodos propios del protocolo HTTP como GET, POST, PUT o DELETE para ponerse en contacto con el servidor. En nuestro proyecto, todas las peticiones que realizan las aplicaciones Android son de tipo POST, este método es el más generalizado cuando se desarrollan servicios web para aplicaciones móviles.

- No mantiene estado. En los servicios REST es necesario que cualquier petición que envíen los clientes, incluya todos los datos necesarios para cumplir la transacción, de manera que el servidor puedan redireccionar y responder a la petición sin mantener el estado entre peticiones. En conclusión, todas las peticiones deben ser completas e independientes. Esto no impide que en el lado del servidor se puedan generar sesiones para requerir autenticación al usuario, o ahorrar tiempos en consultas a la base de datos.

- Expone URIs con forma de directorios. Como se explicó previamente en el apartado acerca de las clases *@Controller* de Spring, para enviar una petición a un servicio web REST se utiliza un URI del estilo *http*. Estas URIs tienen que ser lo más descriptivas posibles, y debe ser fácil e intuitivo adivinar la funcionalidad deseada a partir de la URI.

- Transfiere XML, JSON o ambos. Los datos que se transfieren en una petición deben reflejar el estado actual del recurso determinado, y sus atributos, en el momento en que el cliente de la aplicación realiza

la petición. Los XML o JSON son archivos que representan una entrada de la base, mostrando parejas de atributo-valor.

Como ya se explicó anteriormente, el servidor envía y recibe mensajes JSON. En estos mensajes siempre se representan todos los atributos del recurso que queremos enviar, aunque alguno de estos atributos no sea totalmente necesario para esa transacción, pero esto permite que la aplicación sea más flexible de cara al futuro, y al ser un texto plano no ralentiza la transacción.

La *Figura 2.10* muestra un ejemplo de cómo funciona el servicio web REST real de este proyecto en concreto. En dicha imagen se muestra un ejemplo muy sencillo compuesto por una petición POST al servidor web, que envía el número de código de barras y la respuesta del servidor que devuelve, si este número existe en la base de datos, un JSON analizando el objeto Producto con todos sus atributos; en caso de que no existiera devolvería un objeto vacío (*null*).

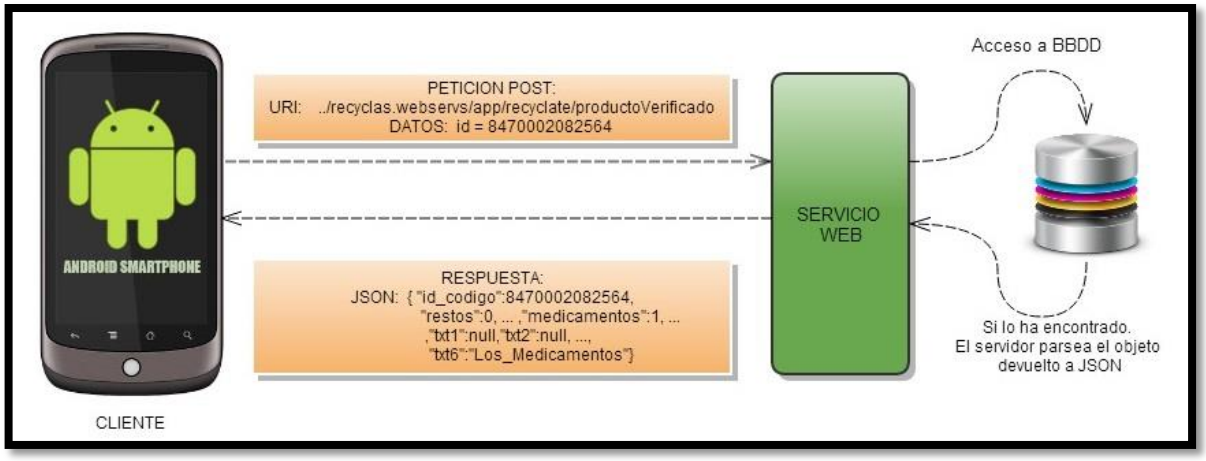


Figura 2.10.- Diagrama servicio REST

Una vez se tiene configurada toda la aplicación web y los servicios web REST definidos es necesario desplegar la aplicación web en un servidor de aplicaciones para que todo funcione.

Como se puede deducir por su nombre, un servidor de aplicaciones [46] es un dispositivo software que proporciona servicios de aplicación por parte de un servidor a los clientes. Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de

negocio y de acceso a los datos de la aplicación. Los principales beneficios de aplicar la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones.

En el IAM se utilizan servidores de aplicaciones WAS (WebSphere Application Server) [47], que es un producto de IBM que se distribuye bajo licencia de pago. Los requisitos del Ayuntamiento incluyen que la aplicación sea desplegable en dichos servidores web.

El equipo de desarrollo buscó la manera de conseguir una licencia que permitiera utilizar este software y poder así desplegar el sistema en un servidor WAS, pero tras descubrir que esto solo nos permitiría hacer pruebas a nivel local nos pusimos en contacto con los responsables del Ayuntamiento de Madrid. Su recomendación fue que si era necesario desplegar la aplicación en un entorno real, se hiciera uso de un servidor Apache Tomcat [48], ya que una vez dado este paso, basta con realizar unas ligeras modificaciones para poder desplegarlo en un servidor WAS.

Cabe aclarar que Tomcat no es un servidor de aplicaciones en sí mismo, sino un contenedor de *servlets*. Para el tipo de aplicación web que se está desarrollando es lo más adecuado, ya que además de permitir que se realicen las peticiones a un servidor real, funciona en cualquier sistema operativo que disponga de una máquina virtual de Java, al estar Tomcat desarrollado en dicho lenguaje de programación. Además esto hace que con unos pocos cambios las aplicaciones desplegadas en un Tomcat se puedan desplegar sin problemas en cualquier otro tipo de servidores de aplicaciones.

Con esto se da por concluido el segundo capítulo de esta memoria. En el siguiente capítulo se pasan a detallar en mayor profundidad los desarrollos realizados para cada una de las aplicaciones con las que se ha trabajado en el proyecto.

Capítulo 3. PLANIFICACIÓN Y DESARROLLO

En este capítulo se tratan en detalle, la planificación y desarrollo de cada una de las aplicaciones tratadas en este proyecto: Recycla.me, Recyclate! e Itinerarios guiados Parque del Retiro.

3.1 Planificación temporal del proyecto

En los primeros puntos de esta memoria ya se han mencionado las distintas decisiones de desarrollo del proyecto. A continuación se muestra un diagrama de Gantt que resume la planificación del trabajo realizado durante todo el curso académico. El diagrama de la *Figura 3.1* se realizó a finales del mes de Octubre, cuando tuvo lugar la primera reunión con los responsables del IAM. Por eso, lo relativo al primer mes no es tanto una planificación, como un simple reflejo del estudio de las alternativas que ofrecía el mercado para poder convertir las aplicaciones en multiplataforma, así como el comienzo de la familiarización con las aplicaciones ya existentes.

Figura 3.1.- Planificación temporal del proyecto. Diagrama de Gantt

Como se aprecia en el diagrama de Gantt, tras acordar con el IAM y las directoras del proyecto cual sería la línea de trabajo a seguir, realizamos una planificación lo más optimista y ambiciosa posible.

En un primer lugar se decidió dedicar el primer mes a terminar de realizar tareas como la redacción del primer capítulo de la memoria, el estudio del código y funcionalidad de las aplicaciones, y, sobretodo, el estudio de los estándares con los que trabaja el IAM.

Una vez pasado este primer mes, el siguiente paso que dimos fue comenzar con las modificaciones necesarias en la app Itinerarios del Retiro, ya que era la aplicación que, a priori, menos cambios iba a requerir. La planificación le asignaba a esta tarea aproximadamente un mes y medio, tras el cual se comenzaría tanto con las pruebas de dicha app, como con los desarrollos de la parte del servidor para Recyclate! y Recycla.me.

A estos últimos se les asignaron dos meses, aunque tras el primer mes ya se comenzó con las modificaciones necesarias en la aplicación Recycla.me. Una vez terminado con lo relativo a Recycla.me, se planificó un mes similar para Recyclate! y un último mes para desarrollar nuevas funcionalidades en la aplicación Hábitat. Mientras tanto se seguiría trabajando en la documentación y realizando pruebas sobre las aplicaciones terminadas.

Como ya hemos indicado se decidió hacer una planificación optimista, aunque una vez se empezaron a estudiar los requisitos expuestos por el Ayuntamiento de Madrid, se comprendió que estos eran unos objetivos muy ambiciosos y difíciles de lograr, por lo que resultaría más realista centrarse en intentar dejar las aplicaciones Itinerarios, Recycla.me y Recyclate! (así como su parte del servidor) completamente listas para su publicación y subida a los servidores del ayuntamiento, dejando para el trabajo futuro los desarrollos de Hábitat. Este objetivo se ha cumplido en un 95% ya que a fecha de entrega de esta memoria solo sería necesario el despliegue de la aplicación web en los servidores de aplicaciones IBM WebSphere.

Al margen de esta decisión, cabe destacar algunas alteraciones que se hicieron en el desarrollo del proyecto en relación a lo expuesto en el diagrama.

Una de las más remarcables fue el cambio en cuanto a la estrategia de corrección de errores de las aplicaciones. En principio se planificó centrarse en corregir los errores existentes durante un mes antes de empezar con los nuevos desarrollos. Se supuso que esto serviría para entender mejor como se habían desarrollado cada una de las aplicaciones, y así nos encontraríamos en una mejor situación a la hora de plantear los nuevos desarrollos. Se empezó siguiendo esta planificación y corrigiendo algunos de los errores más sencillos, pero llegó un momento en que para corregir los sucesivos errores era necesario tener un conocimiento mayor de cada aplicación y decidimos que sería más recomendable centrarse en irlos corrigiendo a la par que se desarrollaban las nuevas tareas para así no perder tiempo, evitando estancarnos en este punto.

Otra alteración respecto a la planificación se produjo con los desarrollos relativos a las aplicaciones Recyclate!, Recycla.me y sus correspondientes web services. Al principio se optó por comenzar por hacer todos los desarrollos de la parte del servidor, aplicando los estándares que previamente habían sido estudiados, en dos meses, para luego dedicar un mes a cada aplicación (primero Recycla.me, después Recyclate!, para que usasen dichos servicios web). Una vez estudiadas a fondo las tecnologías, se llegó a la conclusión de que sería mucho más eficiente realizar los desarrollos de la parte Android y de la parte del servidor de manera simultánea. Además, una vez conocidas las aplicaciones, se empezó trabajando con Recyclate! ya que estaba mejor adaptada al modelo de datos y necesitaba realizar cambios menos profundos. Así, una vez terminada esta, contaríamos con más experiencia para comenzar con las modificaciones de la aplicación Recycla.me, las cuales eran más complejas. Pese a estas modificaciones en el reparto del tiempo, la planificación final de cuatro meses para terminar con todo lo relativo a las dos aplicaciones de reciclaje no se vio apenas alterada, ya que comenzamos con ella la segunda semana de febrero para acabar a mediados del mes de mayo.

En lo que respecta a los demás puntos, no ha habido alteraciones remarcables, por lo que podemos considerar como bastante satisfactoria esta planificación, más teniendo en cuenta que se trataba de unos desarrollos con los que el equipo no estaba familiarizado y no sabía cuánto exactamente se podría tardar en comprender todos los requisitos necesarios para el desarrollo del proyecto.

A continuación analizamos los desarrollos realizados para cada una de las aplicaciones relativas a nuestro proyecto.

3.2 Itinerarios guiados Parque del Retiro

Se ha dividido este punto en cuatro subapartados. En los tres primeros se hace hincapié en el desarrollo de cada uno de los componentes de la aplicación, así como los cambios que se realizaron para alcanzar nuestros objetivos. En el último de estos subapartados analizamos la aplicación desde el punto de vista del rendimiento.

En las primeras fases del proyecto se tuvieron graves problemas de diseño causados por el desconocimiento de los requisitos del cliente (Ayuntamiento de Madrid y personal del Departamento de Calidad del IAM). Todas estas dudas fueron solventadas a lo largo del proceso de desarrollo por parte del equipo, aplicando su criterio en la toma decisiones basándose en lo que consideraban mejor para la aplicación y amparándose en el consejo de Inmaculada y Victoria.

Como se explica brevemente en el estado del arte, el estado de este sistema cuando el equipo de desarrollo se hizo cargo de él era bastante precario desde el punto de vista del alojamiento de datos, ya que muchas de sus funcionalidades hacían uso de recursos que se encontraban en servidores que ya no estaban activos. También se encontraron problemas con el posicionamiento GPS, debido a que la API de Google utilizada había quedado obsoleta tras la última versión, de lo que se habla en una sección posterior.

En una de las primeras reuniones con el IAM, se decidió que los recursos de los que hacía uso esta aplicación no tenían por qué ser alojados en un servidor externo y que podrían ser empaquetados dentro de la propia aplicación, ya que todos estos recursos son estáticos. Esta decisión se tomó al observar que sería más sencillo actualizar los contenidos de la aplicación cuando fuese necesario (es decir, cuando alguno de los recursos cambiase) que mantener y dar soporte a los servidores que contenían dichos recursos. Cabe añadir que en la actualidad las aplicaciones son cada vez más pesadas y los smartphones cada vez ofrecen más prestaciones en cuanto a capacidad de almacenamiento, por lo que tras estas consideraciones el IAM aprobó esta línea de acción en la aplicación Itinerarios guiados Parque del Retiro. Estos temas se abordarán en las siguientes secciones.

Base de Datos

Para detallar con claridad la disponibilidad de los recursos, primero se ha de comentar algún aspecto de la BBDD que usaba la aplicación original. Se utilizó para su implementación la BBDD nativa de Android SQLite, ya que el tamaño de los textos planos, los descripciones de los puntos de ruta y la información de cada uno de ellos eran perfectos para ser almacenados en la base de datos interna del teléfono. Pero el tamaño de los recursos era demasiado pesado para gestionarlo desde SQLite, así que se resolvió alojar los recursos en servidores online y almacenar las URLs de cada uno de ellos en la BBDD.

Debido a que la aplicación original hacía uso de diferentes servidores de hosting gratuito, utilizaba la base de datos para almacenar las URLs de todos los servidores que tenían contratados para alojar los recursos necesarios en la entidad Servidor, la única clase independiente al resto de entidades de la BBDD. Esto tenía su explicación, ya que en caso de que el servidor por defecto estuviera caído por cualquier motivo, lo más inmediato era la selección de otro servidor que también contuviera el recurso y descargarlo de este. Si ningún servidor lo poseía o todos estaban caídos, se intentaba descargar del servidor por defecto.

Hasta la reunión con el IAM anteriormente mencionada, el objetivo era migrar los recursos a los servidores del IAM y mantener la idea de la disponibilidad de servidores, almacenando los servicios web en varios de ellos (si fuese posible), por lo que la base de datos debía ser reconfigurada para apuntar a las nuevas URLs. Esta base de datos está estructurada de una manera muy eficiente y es bastante fácil de comprender una vez analizada.

En primer lugar existen cuatro clases gestoras, que se encargan de la gestión de la base de datos interna de la aplicación. Estas clases son:

- *BBDDAPI.java*: Sirve de interfaz común desde la base de datos a la aplicación que la invoca. Tiene métodos útiles de recuperación de información.

- *BBDDManager.java*: Es la piedra angular de la gestión de la base de datos.

    - Hereda de *SQLiteOpenHelper* e implementa la interfaz *BBDDAPI.java.*

- Trata la creación/apertura/actualización de la base de datos.
- Delega las consultas a la clase *BBDDQueryCompose.java*.
- Llama a *BBDDSlave.java* cuando se tienen que leer ficheros de script para la creación o borrado de tablas.

- *BBDDQueryCompose.java*: Ejecuta las consultas y devuelve los registros obtenidos en forma de vectores de entidades y se los sirve a BBDDManager.java.
- *BBDDSlave.java*: Se encarga de la lectura de ficheros. Los ficheros pueden ser cadenas de texto extremadamente largas y esto se soluciona almacenando la información en *buffers* de cadenas de caracteres.

A parte de estas cuatro clases gestoras se dispone de los archivos de texto para la carga de datos y la clase de constantes, que entre otros datos contiene el nombre de la base de datos y la versión de la misma que se está utilizando.

Tras la reunión y después de tomar la decisión de alojar los recursos dentro de la propia aplicación, la parte de gestión de los distintos servidores (la entidad Servidor y sus clases de control) quedó obsoleta, y se procedió a la migración de los recursos desde los servidores a la propia aplicación sin alterar el resto de la estructura de la BBDD, quedando su modelo entidad relación como se ve en la Figura 3.2. Ninguna de las clases de control de la BBDD ni sus archivos de carga o configuración para la gestión de los servidores de recursos ha sido eliminada del proyecto, ya que es una buena solución a la disponibilidad de servidores y si alguna de las posteriores versiones de la aplicación termina requiriendo el acceso a recursos online, podrán ser reutilizadas.

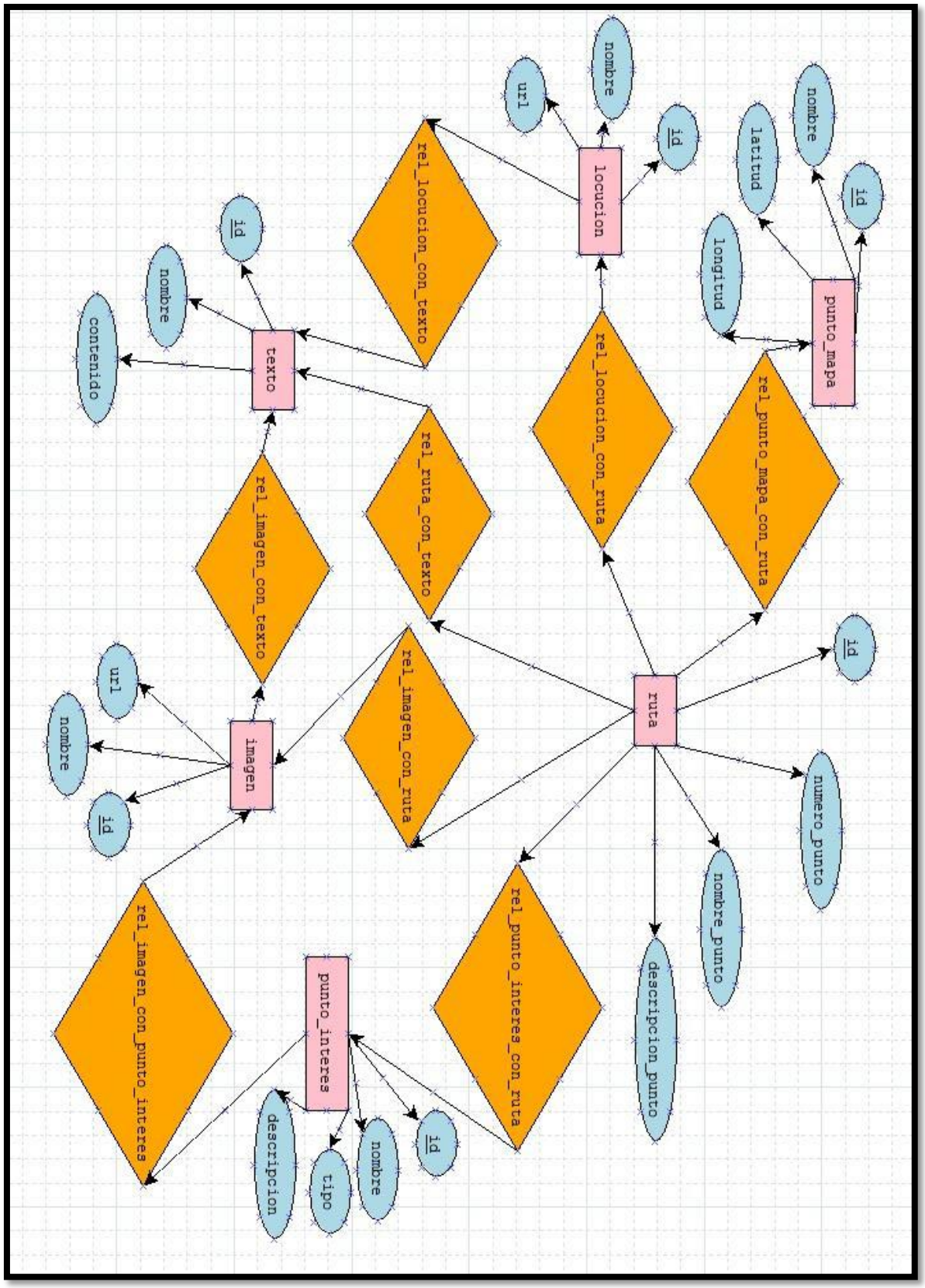


Figura 3.2.- Esquema Entidad-Relación de la base de datos

Servidor y Realojamiento de recursos

Al migrar los recursos, el peso (tamaño) de la aplicación creció mucho, debido a la cantidad de imágenes de las que se hacen uso y, sobretodo, a los archivos de audio.

Las imágenes se guardan en la carpeta /*assets/fotos*, ya que son demasiado pesadas para */res/drawable*; mientras que los archivos de audio son almacenados en la carpeta */res/raw*, directorio por defecto para los archivos de audio en Android.

Los problemas surgieron por la incapacidad de referenciar los recursos de audio desde local mediante el *plugin* utilizado en un principio (la librería nativa de Android MediaPlayer [49]) y de la posterior implementación orientada a obtener recursos en línea que se realizó. En un principio, se comprobaba la conexión a Internet cada vez que se requería un recurso, por lo que se procedió a comentar (por si en una futura versión fuesen necesarios) los métodos de la clase Utils *isOnline(Context ctx)* e *isAvaliable(String url)* y eliminar todas sus referencias en el proyecto, ya que, de ahora en adelante, la aplicación no iba a hacer uso este de recurso.

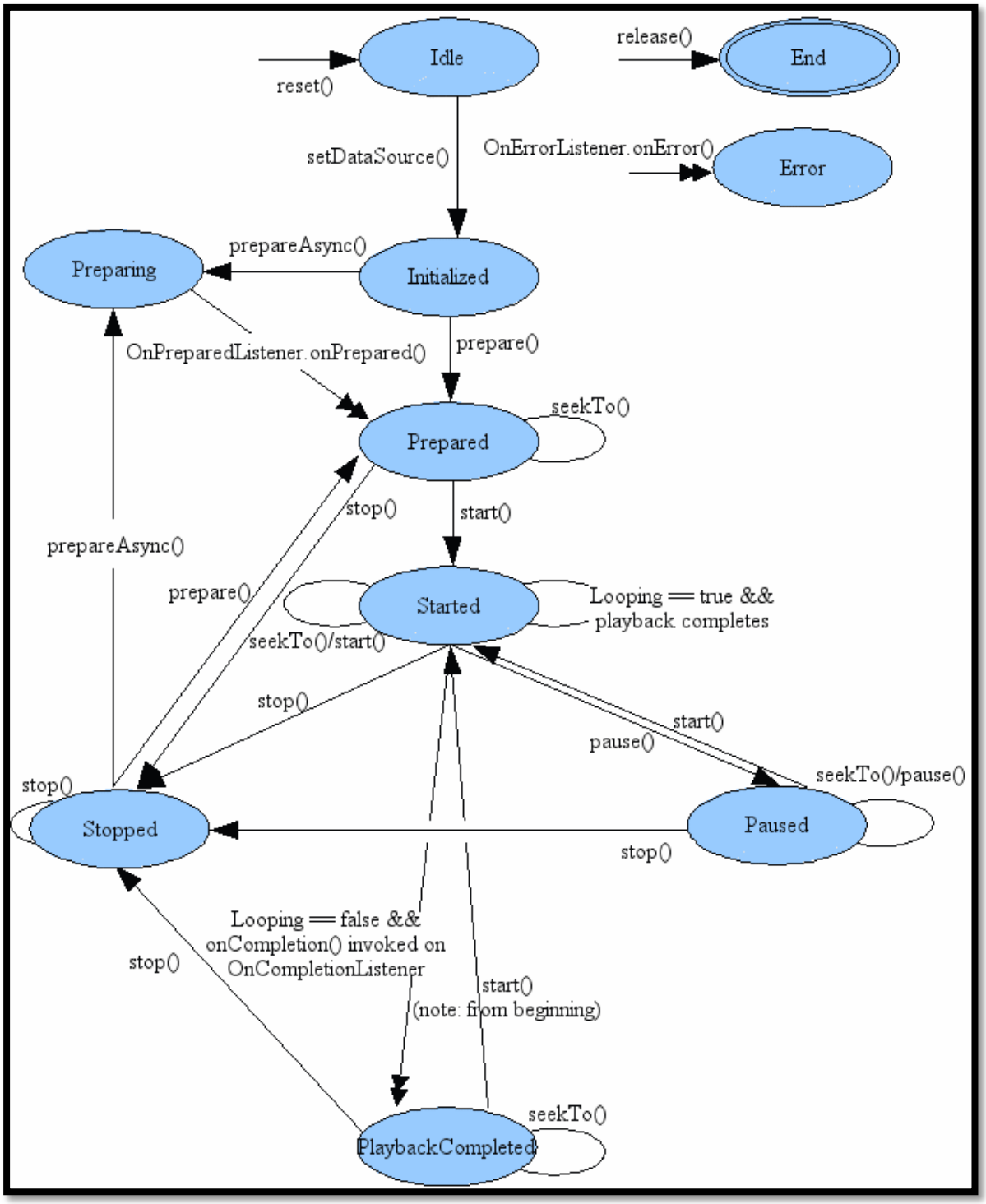


Figura 3.3.- Diagrama de Estados de la clase MediaPlayer

Una vez eliminadas estas dependencias, se procedió a cambiar las referencias a los antiguos servidores con los recursos del fichero de carga de información de la BBDD *generadorDatos.txt* por referencias a la carpeta

/assets/fotos y */res/raw* pertinentes para realizar la carga de recursos desde local.

Se procedió entonces a cambiar el formato de llamada a los recursos de audio dentro de la clase MediaPlayer, lo cual resultó ser un dificultoso proceso de ingeniería inversa, ya que el diagrama de estados de dicha clase es bastante complejo, como se puede apreciar en la *Figura 3.3*, y a que su implementación original dependía en muchos aspectos de la conexión a Internet y accesos a las URLs. Lo más dificultoso fue conseguir que la clase MediaPlayer accediera a los recursos localizados en */res/raw* en el formato Uri, pero tras documentarnos a fondo sobre las distintas maneras de referenciación de recursos en local, encontramos con la instrucción correcta:

```
Uri rutaUri = Uri.parse("android.resource://"
+getPackageName()+"/raw/" +urls.get(siguiente));
```

El caso de las fotos no resultó tan complicado como el de los audios, pero el uso de librerías de terceros como ImageLoader [50] incrementó la dificultad. Tras depurar las comprobaciones de disponibilidad web en la clase ImageLoader.java no hizo falta alterar el método de muestra de imágenes, ya que esta librería acepta rutas locales de la forma:

```
String imageUri = "assets://Fotos/image.png";
```

Aplicación

Por último se abordó el problema de la geolocalización de la aplicación, ya que, como se comentó antes, en un principio se utilizaba la API Google Maps v1, la cual quedó obsoleta poco tiempo después. Fue reemplazada por la API Google Maps v2, cuya implementación resultó levemente complicada ya que los nuevos mapas utilizaban clases Fragment y no Acivitiy como en la anterior versión. Para solucionar este primer escollo,

simplemente se añadió a nuestro proyecto la librería de retrocompatibilidad *android.support.v4* y extendimos nuestra clase *ComoLlegar.activity* desde *android.support.v4.app.FragmentActivity*, permitiendo así el uso de *Fragments* para todas las versiones de Android previas a la 3.0 (API 11).

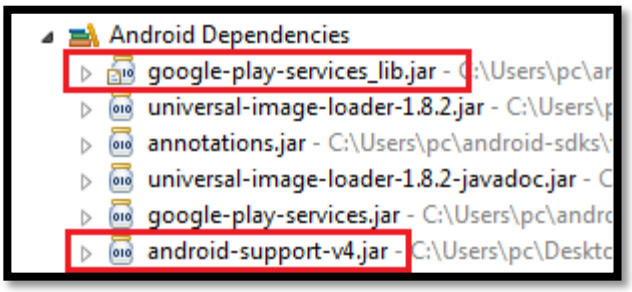


Figura 3.4.- Bibliotecas utilizadas

La configuración de la API de Google Maps está bastante bien detallada en la web de Google Developers [51], aun así, a continuación se detallan los pasos más importantes a seguir.

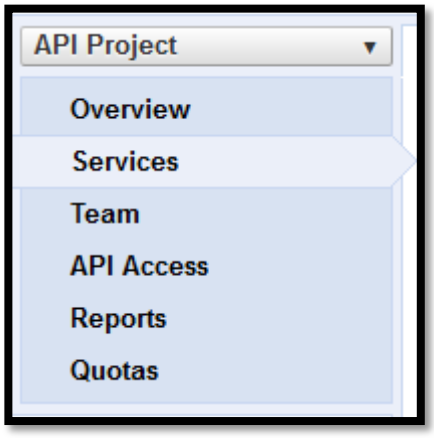
En primer lugar, descargamos el *Google Play Services SDK* mediante el *Android SDK Manager* y lo añadimos a nuestro proyecto como *Android Library Project* referenciándolo. Por último definimos la versión de la API en uso y los permisos necesarios para hacer uso de dicha librería en el fichero *AndroidManifest.xml*:

```
<meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version"/>
```

Los pasos necesarios para integrar esta librería al proyecto comienzan por la creación de una API key que es un código de desarrollador para que Google tenga controlado el número de accesos. Para la obtención de la API key es necesario visitar la consola de APIs de Google (previo registro como desarrollador), en este enlace:

https://code.google.com/apis/console/?pli=1#project:462708371760.

Una vez dentro, hay que dar el nombre de la aplicación y hacer clic sobre la pestaña *Services* que se encuentra en el panel de la izquierda como se puede ver en la *Figura 3.5*.



3.5.-Pestaña de servicios

A continuación se activará el servicio de Google Maps como se muestra en la *Figura 3.6.*

Figura 3.6.- Servicio de Google Maps que hay que activar

Posteriormente, marcando la pestaña de API Access, se puede solicitar una API key pulsando en el botón correspondiente y proporcionándole a la consola el nombre del paquete en el que va a ser introducida la API y la

huella digital con la que se firmará la aplicación (*SHA1*). Para hallar la huella digital se busca el certificado de pruebas del proyecto desde el mismo Eclipse (en Preferences -> Android -> Build). Una vez hallado el directorio, se accede a él a través del símbolo del sistema en Windows y se ejecutan los siguientes comandos desde la terminal:

```
C:\Users\...\.android>"C:\ProgramFiles\...\keytool.exe" -list -v
-keystore debug.keystore -alias androiddebugkey -storepass
android -keypass android
```

Donde *C:\Users…\.android* es la ruta en la que se encuentra el certificado de pruebas *y C:\ProgramFiles\...\keytool.exe* la ruta donde se encuentra la "Java Virtual Machine". Introduciendo esta directiva, encontramos la huella digital o SHA1, con lo que ya se está en condiciones de solicitar la "API key". Google la proporciona y hay que incluirla en el *AndroidManifest.xml* con el fragmento de código de la *Figura 3.6* sustituyendo el valor por el suministrado por Google.

```
1  ...
2  <application>
3  ...
4      <meta-data android:name="com.google.android.maps.v2.API_KEY"
5                 android:value="api_key"/>
6  ...
7  </application>
```

Figura 3.7.- Detalle AndroidManifest.xml

Una vez se tiene la "API key", y se han importado las librerías necesarias se definen los permisos requeridos en el *AndroidManifest.xml*:

```
<permission android:name="com.google.android.providers.gsf.permission.MAPS_RECEIVE" android:protectionLevel="signature" />

<uses-permission android:name=
"com.interfaz_retiro.permission.MAPS_RECEIVE" />
```

Una vez hecho esto ya está incluido el mapa en la interfaz. Sólo queda darle funcionalidad. Para ello basta con modificar la clase asociada a ese archivo *xml*. Solo hay que poner la siguiente línea de código para tener pleno acceso al mapa:

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

Rendimiento

En esta última sección se hace hincapié en las mejoras de rendimiento de la aplicación tras los cambios aplicados.

En primer lugar, el espacio que ocupa la aplicación ha crecido mucho debido a la reubicación de recursos en local tal y como se vio en las secciones anteriores. Este aumento de tamaño no supone una desventaja a día de hoy, ya que la capacidad de almacenamiento de los nuevos dispositivos es muy elevada y muchas de las aplicaciones que circulan por el mercado tienen tamaños similares al de nuestra aplicación. Es más, gracias a la reubicación de los recursos, el rendimiento en cuanto a tiempos de carga y respuesta ha crecido muchísimo, dado que no es necesario acceder a la red para obtener las distintas imágenes y audio que contiene la aplicación, siendo este uno de los mayores motivos por los que se decidió eliminar toda la infraestructura de servidores web para el almacenamiento de los recursos.

En segundo lugar, pero no por ello menos importante, gracias a la nueva versión de la API de GoogleMaps v2 los tiempos de respuesta de la geolocalización y la precisión de la ubicación de la misma han aumentado

considerablemente como se comprobó durante el plan de pruebas realizado en los Jardines del Retiro. Además, las nuevas aplicaciones que funcionen con esta versión de la API podrán usar los mapas de Google Maps basados en vectores 2D y 3D con mejores tiempos de carga y menor consumo de datos, obteniendo unos rendimientos por encima de la media respecto al resto de sistemas de navegación con mapas disponibles.

3.3 Recycla.me y Recyclate!

Este punto se divide en cuatro subapartados en los que se detallan los distintos desarrollos realizados y su puesta en marcha (servidor propio o Amazon). El motivo de englobar las dos aplicaciones en un mismo apartado se debe a una de las decisiones más relevantes que se tomaron durante el desarrollo de nuestro proyecto, la cual mencionamos a continuación.

Como ya se expuso en el estado del arte, al comenzar con este proyecto se encontraron dos aplicaciones (Recyclate! y Recycla.me) que ofrecían unas funcionalidades muy similares pero que a su vez eran completamente independientes la una de la otra, ya que fueron desarrolladas en cursos, y por equipos de trabajo, diferentes. Cada una de ellas utilizaba una base de datos distinta, además la estructura de cada una de ellas tenía poco en común con la otra. Tablas que tenían objetivos equivalentes (almacenar los datos relativos a cada producto y donde debía reciclarse) tenían columnas diferentes y guardaban información de manera distinta. Cada una tenía una estrategia para almacenar los datos, por ejemplo, Recyclate! controlaba que la información de los artículos introducidos fuera veraz mientras que Recycla.me no lo hacía.

Por lo tanto el objetivo fue conseguir que estas dos aplicaciones funcionaran de la forma más similar posible y, sobre todo, que pudieran utilizar el mismo servidor y la misma base de datos, lo cual multiplicaría por dos la utilidad de ambas aplicaciones. Por este motivo, a día de hoy, no tiene sentido hablar de una aplicación sin mencionar la otra.

En primer lugar se describe la base de datos; a continuación, el desarrollo de la parte del servidor; y por último las dos aplicaciones. Se trata primero Recyclate!, ya que a pesar de que es una aplicación más reciente, es conveniente seguir este orden a la hora de exponer ambas

aplicaciones. Una vez vistos los distintos desarrollos realizados, se analiza cómo se ha realizado el despliegue en el servidor.

Base de Datos

En primer lugar hablamos de la base de datos externa, a la que ambas aplicaciones accederán, y que se encuentra alojada en el mismo servidor que la aplicación web. Atendiendo a su estructura, cuenta con cinco tablas. Dos de ellas (*códigos_materiales* y *códigos_filtrados*) son utilizadas por ambas aplicaciones, mientras que las otras tres tablas están destinadas a satisfacer funcionalidades exclusivas de la aplicación Recyclate!.

A continuación se describe para qué sirve cada tabla y cada una de sus columnas:

- *códigos_materiales*: Esta tabla almacena todas los códigos de barras escaneados por el usuario. Cada entrada tendrá un identificador autogenerado que hará de clave primaria. Este id existe porque esta tabla almacenará más de una vez información relativa a un mismo código de barras. Está formada por los campos:
    - id_scan (Clave Primaria): El id que comentábamos nos permitirá tener más de una entrada en la tabla para el mismo código, y así filtrar los mismos.
    - id_codigo: El código de barras del producto escaneado.
    - restos/vidrios/envases/cartón/punto_limpio/medicamentos: Estos seis campos (de tipo numérico) almacenan un 1 si el producto escaneado contiene ese material y 0 en caso contrario.
    - txt1/txt2/.../txt5/txt6: Seis campos (de tipo *varchar*) correspondientes a cada tipo de material, contienen una descripción del material si el producto lo contiene y *null* en caso contrario.

- *codigos_filtrados*: En esta tabla se almacenan los códigos una vez filtrados. Es decir, si en la tabla anterior hay tres entradas para el mismo código de barras, y la descripción de estas contienen los mismos materiales, al escanear desde la aplicación otra vez el mismo código, si la descripción de materiales coincide, se considerará esa descripción como válida. Una vez sabemos que la descripción de un producto es válida, lo introducimos en esta tabla. Consta de los siguientes campos:

    - id_codigo (Clave Primaria): El código de barras del producto escaneado. En este caso será la clave primaria, ya que sabemos que en esta tabla solo puede haber una entrada por producto.

    - restos/vidrios/envases/cartón/punto_limpio/medicamentos: Los mismos seis campos con la misma función que sus homónimos en la tabla *códigos_materiales*.

    - txt1/txt2/.../txt5/txt6: Los mismos seis campos con la misma función que sus homónimos en la tabla *códigos_materiales*.

- *trivial*: Esta tabla incluye todas las preguntas, y sus respectivas respuestas, del juego "*ReciclaDOS*". Cuenta con los siguientes campos:

    - id_pregunta (Clave Primaria): Identificador único de cada pregunta.

    - tematica: De tipo palabra (*varchar*), nos indica a qué categoría corresponde cada pregunta.

    - respuesta1/2/3/4: Representan las diferentes respuestas de cada pregunta.

    - respuesta_correcta: Nos indica que número de respuesta es la correcta.

    - dificultad: Contiene un número que nos indica el nivel de dificultad de la pregunta (1-fácil, 2-intermedio, 3-difícil).

- *usuarios*: Contiene un listado con los e-mails de todos los usuarios que se han registrado para jugar en la aplicación, así como la puntuación obtenida por cada uno. Sus campos parecen obvios:

    - usuario (Clave Primaria): De tipo palabra (*varchar*), almacena el e-mail del usuario, solo permitiremos un usuario por cada dirección de e-mail por lo que se utiliza como clave primaria.

    - puntuación: De tipo numérico, almacena la puntuación de cada jugador, se va incrementando al finalizar cada partida.

- *consejos*: Esta tabla almacena los consejos sobre reciclaje de la aplicación Recyclate!. En la actualidad estos consejos sólo se almacenan en la base de datos local de la aplicación, por lo que esta tabla está contemplada de cara a una posible ampliación del proyecto. Esta tabla contiene los siguientes campos:

    - id_consejo (Clave Primaria): Identificador único de cada consejo.

    - consejo: Este campo contiene el título de cada consejo de la tabla.

    - material: Guarda el tipo de material que se está utilizando en el consejo o truco (restos, vidrio, envases o cartón).

    - imagen_lista: Guarda la URL de la imagen que se mostrará como icono en la lista.

    - texto: Muestra la descripción del consejo. Las recomendaciones a seguir por parte de los usuarios.

    - imagen_texto: Contiene la URL de la imagen que irá acompañando al texto del consejo.

    - tags: Nube de etiquetas relacionadas con el consejo. Será de utilidad a la hora de hacer el filtrado de consejos en la búsqueda.

Como ya se ha mencionado, además de la base de datos externa, la aplicación Recyclate! utiliza una base de datos interna SQLite. Esta contiene sólo dos tablas, una que almacena los consejos y otra con las

preguntas del trivial. Ambas contienen los mismos campos que las tablas con el mismo nombre en la base de datos externa. La lista con todos los consejos se almacena en esta tabla y para actualizarlos habrá que hacerlo de manera local. Por el contrario, la base de datos local de preguntas de trivial se actualiza una vez al mes. En esa ocasión la aplicación lanzará una petición al servidor, descargando a la base de datos local todas las preguntas que hayan sido introducidas en los últimos 30 días. De esta manera, no se tiene que acceder a la base de datos externa cada vez que se juegue al trivial, lo que hará que el rendimiento de la aplicación sea mucho más alto.

Servidor

En este apartado se desglosa el desarrollo relativo a la parte del servidor, los servicios web que se encargan de nutrir a las dos aplicaciones de reciclaje y la forma en que estos están implementados. Del mismo modo que las aplicaciones Recycla.me y Recyclate! acceden a la misma base de datos, el mismo proyecto ofrecerá servicios a ambas aplicaciones. De hecho hay servicios web a los que hacen peticiones tanto una como otra.

Estructura del proyecto

Como se explicó en el capítulo anterior, para desarrollar este proyecto se siguió la arquitectura básica que define Spring y más en concreto Spring MVC. Aunque cabe destacar una pequeña modificación, normalmente una aplicación web Spring sigue la siguiente estructura:

- Controller: Se encargan de manejar las peticiones recibidas en el servidor.
- Servicios: Llamados por los controladores, se encargan de llamar a los DAOs (Data Access Object) y realizar las operaciones necesarias para devolver la información deseada.
- DAOs: Se encargan de leer, grabar y modificar datos en el BBDD.
- Clases modelo: Representan la información almacenada en la base de datos.

En concreto, en este proyecto, se ha optado por desarrollar unos controladores sencillos, que llamen directamente a los DAOs que interactúan con la base de datos, en vez de desarrollar controladores más complejos, que dependiendo de los datos recibidos llamen a unos servicios determinados u otros.

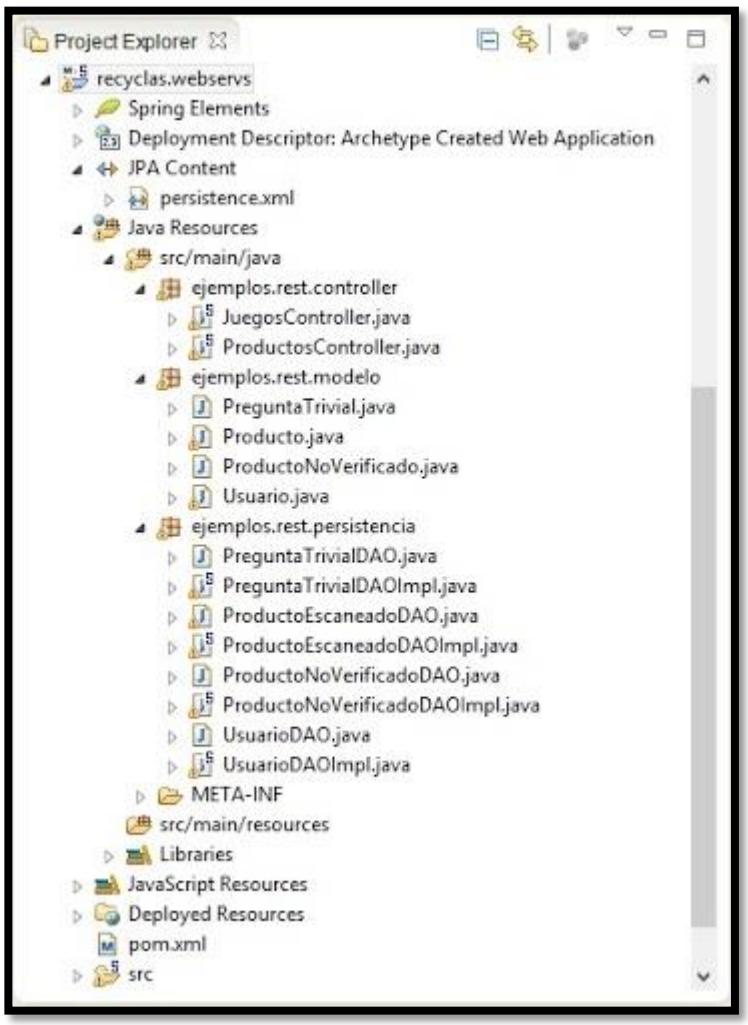
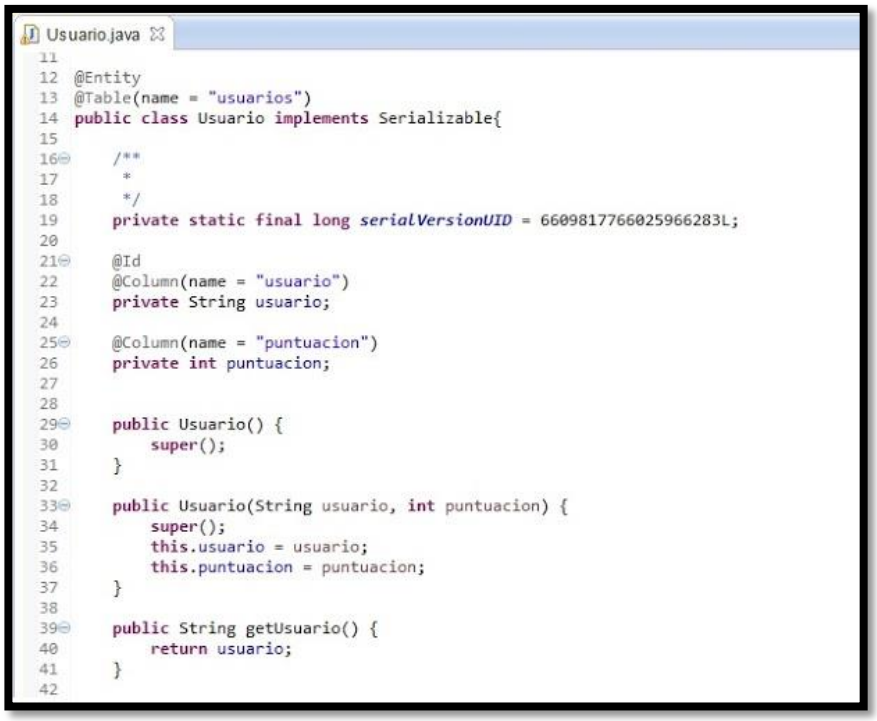


Figura 3.8.- Estructura de los Webservices de Recycla.me y Recyclate!

Como se aprecia en la *Figura 3.8*, cada una de las clases está bien diferenciada en su propio paquete. En el paquete *controller*, se encuentran dos clases controladoras: *ProductosController*, que se encarga de tratar todos los servicios web relativos a la administración de los productos

escaneados, y *JuegosController*, que contiene todos los servicios web necesarios para el correcto desarrollo de los juegos.

Respecto a cada uno de los servicios web desarrollados, en el paquete *modelo* se encuentran los POJOs de los que ya se habló en el capítulo anterior. Es decir, cada clase de este paquete representa cada una de las tablas de la base de datos y sus atributos los diferentes campos de cada una. Como se puede comprobar en la *Figura 3.9*, todas estas clases implementan la interfaz Java *Serializable*. Esto es necesario ya que JPA necesita convertir estos objetos en bytes, para así poder dar persistencia en la base de datos. Para poder transformar un objeto en un mapa de bytes, es necesario que el mismo sea *Serializable*, y además toda clase que implemente esta interfaz deberá incluir el atributo *serialVersionUID*. Este será el identificador universal de toda clase *Serializable* (el entorno Eclipse lo define automáticamente) y servirá para que Java pueda saber a qué clase tiene que convertir un objeto serializado determinado. Si no encontrara ninguna clase con el *serialVersionUID* buscado, Java lanzaría la excepción *InvalidClassException*.


Usuario.java

```
11
12 @Entity
13 @Table(name = "usuarios")
14 public class Usuario implements Serializable{
15
16     /**
17      *
18      */
19     private static final long serialVersionUID = 6609817766025966283L;
20
21     @Id
22     @Column(name = "usuario")
23     private String usuario;
24
25     @Column(name = "puntuacion")
26     private int puntuacion;
27
28
29     public Usuario() {
30         super();
31     }
32
33     public Usuario(String usuario, int puntuacion) {
34         super();
35         this.usuario = usuario;
36         this.puntuacion = puntuacion;
37     }
38
39     public String getUsuario() {
40         return usuario;
41     }
42
```

Figura 3.9.- Detalle clase Usuario.java

Por último, se describe el paquete que contiene las clases más complejas, el paquete persistencia, que aglutina todos los DAOs. Estas clases, que implementan el patrón DAO, son las encargadas de realizar las comunicaciones con la base de datos. Es por eso que tenemos también una clase DAO por cada tabla de la base de datos con la que interactuamos. Siguiendo el patrón, se ha desarrollado una interfaz en la que se definen los métodos propios de cada DAO y otra clase que implementa dichos métodos. Esto hace mucho más escalable esta aplicación, ya que se pueden realizar distintas implementaciones de la misma interfaz dependiendo del tipo de base de datos al que queramos acceder a lo largo de la vida de la aplicación web.
En cada método de estas clases DAO se inicia una sesión en el servidor de la base de datos, a continuación se realizan las operaciones necesarias (lectura, escritura o modificación) y se cierra la sesión (haciendo efectivos los cambios necesarios) devolviendo en cada caso el objeto deseado.

El siguiente diagrama de la *Figura 3.10* resume de una manera gráfica la forma en que hemos estructurado esta aplicación web.

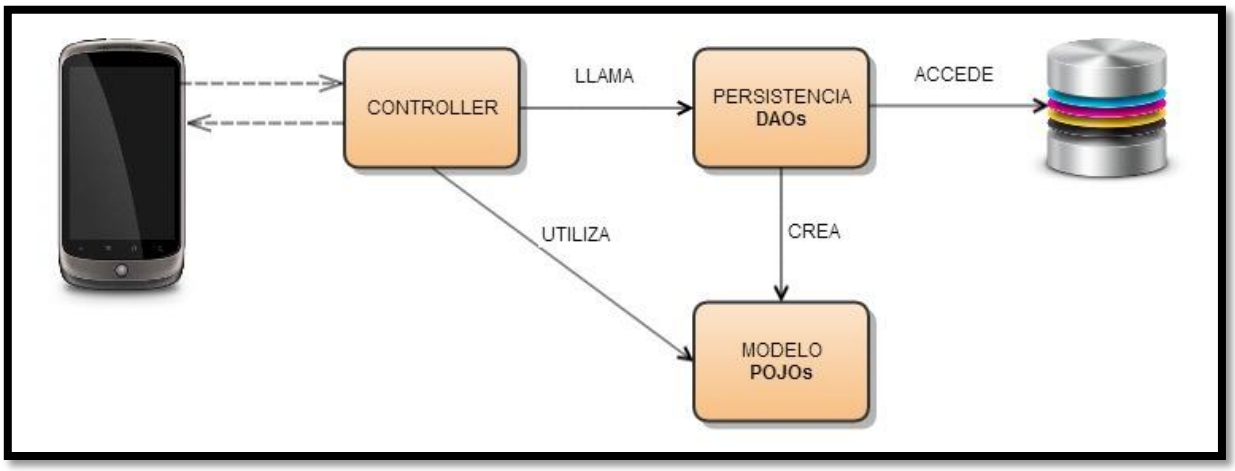


Figura 3.10.- Diagrama de la estructura de la aplicación web

Servicios web

A continuación detallamos los diferentes servicios web que ofrece nuestra aplicación. En primer lugar, se describen los comprendidos en la clase *ProductosController*:

- ***insertarProducto***:
    - *URI: ../recyclas.webservs/app/recyclate/aniadirproducto*
    - Recibe un producto con su código de barras y la descripción de los materiales que contiene. Da persistencia al producto en la tabla de productos verificados. El cliente se ha de encargar previamente de verificar que la descripción del producto era válida. El controlador devuelve la descripción del propio producto a la aplicación para que esta sepa que todo ha ido bien.
- ***insertarProductoNoVerificado***:
    - *URI: ../recyclas.webservs/app/recyclate/aniadirprodnoverif*
    - Este servicio web realiza la misma operación que el anterior, con la diferencia de que la inserción se produce en la tabla de productos no verificados.
- ***buscarProducto***:
    - *URI: ../recyclas.webservs/app/recyclate/productoVerificado*
    - Este servicio web es el invocado al escanear el código de barras de un producto. El controlador recibe el código de barras, y busca en la tabla de productos verificados de la base de datos la descripción del mismo. Si existe el producto, devuelve un JSON con todos los atributos que contiene la descripción de los materiales del producto buscado. En caso de que este no exista devuelve un valor nulo a la aplicación.
- ***buscarNoVerificados***
    - *URI:../recyclas.webservs/app/recyclate/productosNoVerificados*
    - A raíz del parámetro código de barras enviado desde el cliente, este servicio devolverá un JSON con la lista de todos los productos con este código que existen en la tabla

de productos no verificados. La aplicación utilizará esta lista para verificar si la nueva descripción de un producto es válida.

A continuación enumeramos los servicios web que maneja el controlador *juegosController*:

- ***insertarUsuario***
  - *URI: ../recyclas.webservs/app/recyclate/aniadirUsuario*
  - Se inserta el e-mail recibido como parámetro en la tabla de usuarios con la puntuación inicializada a 0.
- ***puntosUsuario***
  - *URI: ../recyclas.webservs/app/recyclate/puntosUsuario*
  - Recibido el e-mail del usuario devuelve su puntuación actual.
- ***actualizaPuntos***
  - *URI: ../recyclas.webservs/app/recyclate/actualizaPuntos*
  - Este servicio web recibe un usuario y la nueva puntuación que ha conseguido. Se encarga de actualizar la puntuación del mismo en la base de datos.
- ***actualizaPreguntas***
  - *URI: ../recyclas.webservs/app/recyclate/actualizaPreguntas*
  - Como ya se ha comentado, cada mes la aplicación actualizará las preguntas almacenadas a nivel interno con las nuevas preguntas introducidas en la base de datos. Este servicio web recibe un id de pregunta, y devuelve un JSON con la lista de todas las preguntas cuyo id (su clave primaria) sea mayor que el de la pregunta recibida.
- ***guardaPregunta***
  - *URI: ../recyclas.webservs/app/recyclate/guardaPregunta*

- Este servicio web recibe un JSON con una lista de preguntas y se encarga de darles persistencia en la base de datos. Actualmente nuestras aplicaciones nunca llaman a este servicio web, pero nosotros lo desarrollamos para hacer una copia de todas las preguntas desde la base de datos antigua (de la que no teníamos clave de acceso) hacia la nueva que implementamos. Nos pareció buena idea mantenerlo dentro del proyecto, como un buen ejemplo de servicio web de administración que facilitará que en futuro se puedan desarrollar otro tipo de aplicaciones cliente que se encarguen de actualizar la base de datos de una manera más cómoda.

Recyclate!

En cuanto al desarrollo de aplicaciones Android, el Ayuntamiento de Madrid no tiene definidos todavía (se están redactando durante la escritura de esta memoria) ningún estándar a la hora de trabajar con dicha tecnología. Por este motivo el trabajo realizado se ha basado principalmente, en solucionar los problemas existentes y sobretodo en diseñar la manera de realizar las peticiones a los nuevos servicios web.

Atendiendo a la corrección de fallos, lo primero que se encontró fue que al intentar escanear un código, la aplicación no conectaba correctamente con la librería *ZXing* (librería de código abierto y gratuito para lectura de códigos de barras y QR), por lo que tuvimos que volver a enlazar la aplicación con la librería, incluyéndola en el *build path* de la aplicación de igual manera que hace Recycla.me. Ahora tanto Recyclate! como Recycla.me utilizan la misma librería, por lo que al hacer una actualización de la misma el trabajo a realizar se vería simplificado. Al margen de esta problemática, se han detectado diferentes errores que se fueron arreglando a medida que se encontraron.

Antes de comentar las modificaciones realizadas en la aplicación, explicamos primero cómo funcionaba la misma cuando se trató por primera vez, centrándonos en su comunicación con la base de datos externa. Recyclate! interactuaba con la base de datos mediante peticiones *HttpPost* con unos archivos PHP almacenados en un servidor gratuito. Estas peticiones se llevaban a cabo mediante diferentes objetos HttpClient. A este objeto se le iban añadiendo parámetros en forma de *array* simple de pares Atributo-Valor.

A continuación se encapsulaba la petición en un paquete POST que referenciaba a la URL del servidor donde se encontraba alojado el archivo PHP deseado, almacenando así la respuesta del servidor en un objeto *HttpResponse* el cual se procesaba y se almacenaba en un *String*. Una vez recogida esta respuesta, se enviaba a las clases *Solucion.java* y *SolucionUsuario.java*, que transformaban los datos desde su formato en base de datos a la forma en que consumen los mismos las distintas actividades.

Los nuevos desarrollos han conseguido que el proceso de comunicación con la base de datos se simplifique mucho, haciéndolo mucho más comprensible para el programador que se vea en la situación de modificar la aplicación en el futuro. Se crearon dos paquetes, que contienen las clases necesarias para hacer más comprensible la comunicación con el servidor. Estos son el paquete *modelo*, que realiza la misma función que su homónimo en el proyecto del servidor, es decir, representar los objetos de la forma en la que se almacenan en la base de datos, y el paquete *connectionUtils* que incluye la clase que implementa todos los métodos necesarios para realizar las peticiones POST a nuestro servidor.

Figura 3.11.- Estructura del proyecto Recyclate!

Para hacer una petición al servidor, simplemente creamos un objeto del tipo *PeticionPost.java*, se añaden los datos necesarios y se llama a la URI correspondiente. Es un proceso realmente sencillo como se aprecia en la *Figura 3.12*.

```
ObjectMapper o = new ObjectMapper();

PeticionPost post = new PeticionPost (PeticionPost.urlBase + "/recyclas.webservs/app/recyclate/productosNoVerificados");
post.add("id",Long.toString(producto.getId_codigo()));
String lista = post.getRespueta();

List<ProductoNoVerificadoDato> listaProds = o.readValue(lista, new TypeReference<List<ProductoNoVerificadoDato>>(){});
```

Figura 3.12.- Método para la creación de un objeto del tipo PeticionPost

En estas cuatro líneas de código están los pasos necesarios para realizar una petición al servidor y procesar la respuesta. El método *getRespuesta()* de la clase *PeticionPost* genera un objeto de la clase Java *URLConnection*, inicia una conexión con el servidor y mediante un *BufferReader* convierte la respuesta del servidor en una cadena de caracteres.

Este *String* devuelto por el servidor es el que se trata para conseguir una instancia del objeto deseado. En el caso de una imagen, lo que se recibe es un *ArrayList* con todos los productos que tengan el número de código de barras indicado. Este análisis se realiza mediante la librería *Jackson*. *Jackson* es una librería de Java, la misma que Spring utiliza en el proyecto servidor, que mediante su clase *ObjectMapper* permite en una sola línea crear un objeto de la clase que se especifica a partir de un JSON con la información necesaria para ello. *Jackson* también ofrece la posibilidad de realizar esta operación a la inversa, es decir, a partir de un objeto genera un JSON, lo que ha sido de especial ayuda a la hora de enviar información al servidor para que este la almacene en la base de datos del sistema.

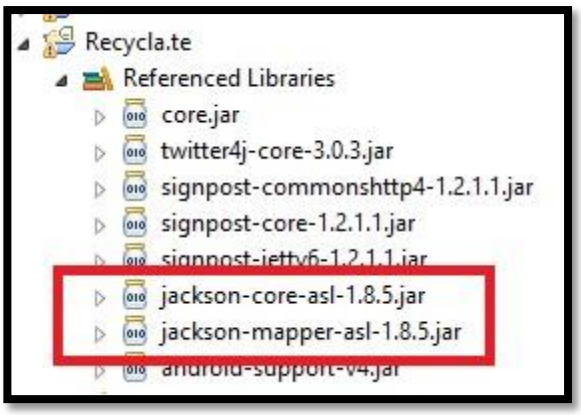


Figura 3.13.- Librería Jackson

Son varias las clases que interactúan con la base de datos externa, y que por lo tanto, han sido modificadas por completo en cuanto a funcionamiento, como por ejemplo: *CargandoMateriales.java*, que se encarga de consultar la tabla de códigos filtrados y devolver los materiales, o las clases *GuardandoCodFiltrados.java* y *GuardandoCodMateriales.java*, que guardan en la base de datos la información que recogen del usuario. Por otra lado están las relativas a los juegos: *GuardandoUsuarios.java*, *SponsorIdentifyingActivity.java*, *Questions.java*, *GameSelectors.java* o *PostingScores.java*.

Debido a la complejidad a la hora de crear las peticiones, estas clases eran realmente largas y difíciles de comprender. Gracias a la nueva implementación de la comunicación con los servicios web de la aplicación, no solo se han podido hacer más sino que incluso se han eliminado aquellas que eran redundantes, aglutinando su funcionalidad en una sola clase.

Más concretamente, ahora la clase *GuardandoCodMateriales.java*, no solo se encarga de enviar la petición a la base de datos para guardar un producto no verificado; tras estos nuevos desarrollos, esta clase también carga todos los productos con el número de código de barras deseado, chequea si la descripción del nuevo objeto es válida (siguiendo los pasos explicados anteriormente) y guarda el mismo en la tabla de la base de datos que corresponda. De este modo la clase *GuardandoCodigosFiltrados.java* no es necesaria, ya que *GuardandoCodMateriales.java* aglutina toda la funcionalidad necesaria, ofreciendo un código muy sencillo.
De igual modo, ahora la clase *SponsorIdentifyingActivity* es capaz de, recibiendo un e-mail de usuario, comprobar si este ya está registrado en la

base de datos (cargando, en este caso su puntuación); y en caso contrario, hacer la inserción del mismo asignándole la puntuación inicial, dejando obsoleta la clase *GuardandoUsuario.java*.

Por último cabe mentar las modificaciones realizadas en las clases *Solucion.java y SolucionUsuario.java*, que son las encargadas de presentar la información que ofrecen nuestros objetos del modelo al resto de actividades. Esto hace mucho más flexible esta aplicación, ya que un cambio en el modelo de datos no afectaría al funcionamiento de ninguna de las diferentes actividades Android.

Recycla.me

Al encarar esta aplicación se encontraron dos problemas importantes. El primero vino a la hora de intentar comprender el código existente, todas las clases se encontraban en un mismo paquete y era muy difícil entender la funcionalidad de algunas de ellas. Además al ser una aplicación desarrollada hace más tiempo, los desarrollos Android no seguían algunos de los patrones que hoy (estando mucho más extendido el desarrollo para este SO) son utilizados por un gran porcentaje de desarrolladores, además algunos de los métodos propios de Android estaban obsoletos, por lo que se trató en la medida de lo posible de actualizarlos.

El segundo gran problema que nos encontramos fue que el modelo de datos que se iba a utilizar era el empleado en la aplicación Recyclate!, y este tenía poco que ver con el que utilizaba Recycla.me, lo que hizo que el proceso de adaptación de la aplicación para consumir los nuevos servicios web fuera más complejo que en el caso anterior, ya que los objetos que leía de la base de datos no eran los que consumía la clase *Solucion.java*. Esta clase, al igual que en Recyclate!, era la encargada de presentar la información de los productos a las diferentes actividades Android. Por eso se decidió empezar desarrollando la aplicación Recyclate!, lo que hizo que una vez adquirida la experiencia necesaria con esta, nos resultara más sencillo el desarrollo de Recycla.me.

Con el fin de ganar en claridad, y hacer más fácil el trabajo a los posibles desarrolladores que traten con la aplicación en el futuro, lo primero que se hizo fue organizar el proyecto. Generamos una estructura de paquetes similar a la que ya conocíamos de Recyclate! y añadimos los

paquetes necesarios para la conexión con el servidor. En la *Figura 3.14* podemos ver la estructura inicial de la aplicación y en la *Figura 3.15* su nueva estructura tras la reorganización.

Figura 3.14.- Estructura del proyecto al comienzo

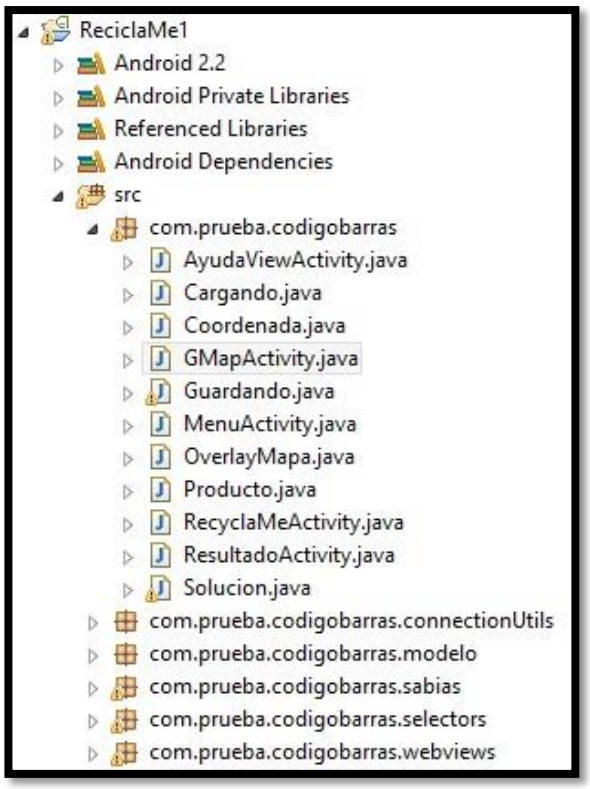


Figura 3.15.- Estructura final del proyecto

Como se puede apreciar en *la Figura 3.15*, la estructura del código queda mucho más clara, y de un solo vistazo se pueden diferenciar los distintos tipos de actividades que existen en la nueva versión.

Respecto a los cambios necesarios para comunicarse con el servidor, se han realizado siguiendo el mismo esquema utilizado en Recyclate!. Se crearon los objetos modelo respectivos a *Producto* y *ProductoNoVerificado*, que son los que se envían y reciben en formato JSON al servidor y la clase *PeticionPost* que se utilizará para realizar la conexión con el mismo, además de las modificaciones necesarias en la clase *Solucion.java*.

Al contrario que en Recyclate! aquí hubo muchas otras clases que fueron modificadas para que la información contenida en los nuevos objetos pudiera mostrarse a los usuarios. En concreto cambiamos todas las actividades incluidas en el paquete *selectors*, que se encargan de crear el texto a mostrar para cada material del producto escaneado, concatenando si el producto contiene más de un material distinto.

```
cb12P.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
         if (cb12P.isChecked()) {
            cat1+=getResources().getString(R.string.catenvases);
            n2+=getResources().getString(R.string.cosmeticon2);
            n3+=getResources().getString(R.string.cosmeticon3);
        }

        else if(!cb12P.isChecked()){
            cat1 =quitaPalabra(cat1, getResources().getString(R.string.catenvases));
            n2 =quitaPalabra(n2, getResources().getString(R.string.cosmeticon2));
            n3 =quitaPalabra(n3, getResources().getString(R.string.cosmeticon3));
        }
    }
});
```

Figura 3.16.- Detalle de la clase Solucion.java

Además, como se puede comprobar en la *Figura 3.16*, se han adaptado todas estas clases, once en total, para que muestren los textos en diferentes idiomas dependiendo del idioma del dispositivo del usuario. Para ello se han incluido en los recursos de la aplicación la traducción realizada por nuestros compañeros de Recyclate! el curso pasado.

Despliegue en servidor

Con el objetivo de que las aplicaciones, con sus nuevos desarrollos, estén disponibles para todo aquel que decidiera comenzar a probarlas, y dado el largo proceso de subida a los servidores del IAM, se optó por desplegarlas en los servidores de Amazon antes de la entrega final del presente proyecto.

Fueron las tutoras del mismo quienes recomendaron la utilización de estos servidores, ya que ofrecen un rendimiento muy alto, permitiendo mantener unidos el servidor de aplicaciones y el de base de datos, y todo ello sin tener que pagar cantidades iniciales de puesta en marcha, ni realizar trámites interminables para poner en marcha un servicio.

Amazon Web Services (AWS) [52] ofrece una gran cantidad de servicios diferentes, pero probablemente los más extendidos (y el que se ha elegido para este caso) son sus instancias *EC2*. Amazon describe este servicio de la siguiente manera:

> *Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona capacidad informática con tamaño modificable en la nube. Está diseñado para facilitar a los desarrolladores recursos informáticos escalables basados en web.*
> *La sencilla interfaz de servicios web de Amazon EC2 permite obtener y configurar su capacidad con una fricción mínima. Proporciona un control completo sobre sus recursos informáticos y permite ejecutarse en el entorno informático acreditado de Amazon. Amazon EC2 reduce el tiempo necesario para obtener y arrancar nuevas instancias de servidor en minutos, lo que permite escalar rápidamente la capacidad, ya sea aumentándola o reduciéndola, según cambien sus necesidades. Amazon EC2 cambia el modelo económico de la informática, al permitir pagar solo por la capacidad que utiliza realmente. Amazon EC2 proporciona a los desarrolladores las herramientas necesarias para crear aplicaciones resistentes a errores y para aislarse de los casos de error más comunes. [53]*

A grandes rasgos, se puede decir que lo que Amazon ofrece son unas máquinas Linux (con distintas distribuciones) en la nube, en las que permite instalar el software que deseemos para así poder ser utilizadas como servidor. Para instalar el software requerido, se ha de conectar a la máquina Linux mediante el protocolo *SSH* (Secure Shell; en español, "intérprete de órdenes segura"), utilizando un certificado (un archivo *.pem*) que Amazon ofrece para asegurar que solo pueden acceder a la instancia EC2 las personas que posean este archivo, utilizándolo como firma electrónica.

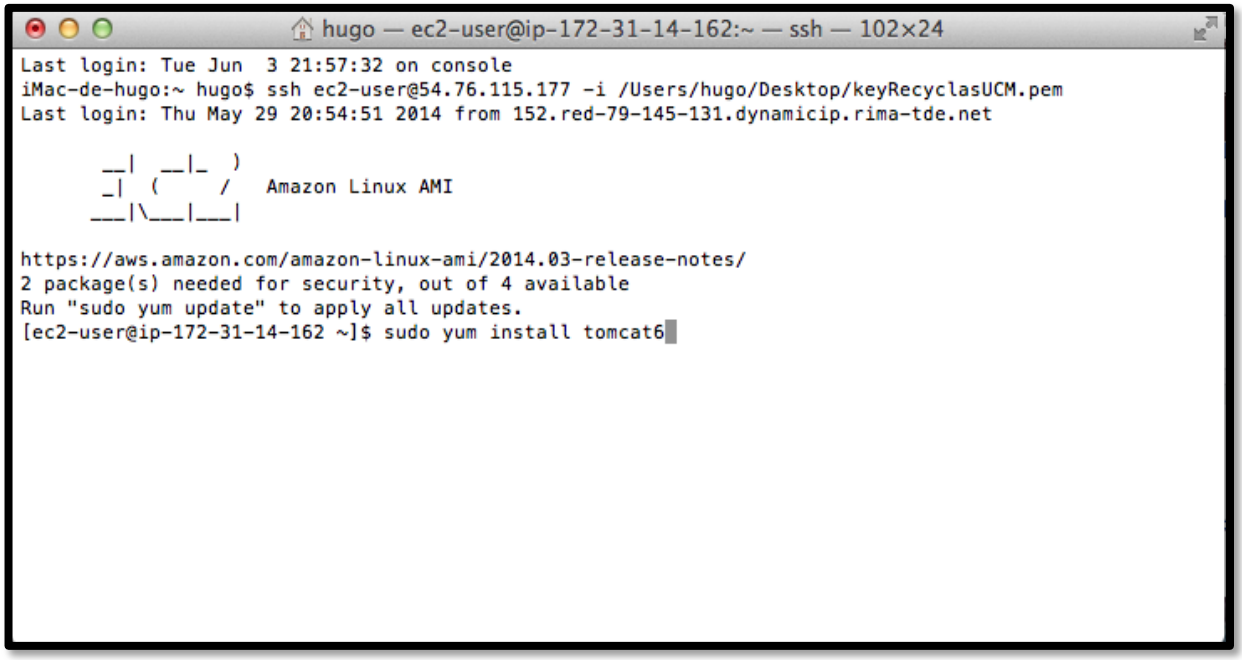


Figura 3.17.- Ejemplo de conexión con instancia EC2

Se eligió una máquina con el sistema operativo *Amazon Linux* (el recomendado desde AWS) y se conectó a la misma desde un terminal de nuestro ordenador Linux. Una vez conectados, mediantes diferentes comandos procedimos a instalar el software necesario para desplegar nuestra aplicación, en este caso:

- El contenedor de servlets *Tomcat 6*
- El servidor de base de datos *MySQL Server*
- *PhpMyAdmin* para gestionar nuestra base de datos vía web
- *PHP* para poder utilizar dicho gestor

Este software se descarga desde el repositorio *YUM* (Yellow dog Updater, Modified) que es una herramienta libre de gestión de paquetes para sistemas Linux y una vez instalados correctamente funcionan de una manera similar a como lo hacen en un sistema local. En la *Figura 3.17* se puede ver un ejemplo de cómo realizar la conexión con el servidor y de cómo instalar *Tomcat 6* desde el mencionado repositorio.

Una vez instalado el software se procede a subir los archivos necesarios para el despliegue de la aplicación. En primer lugar, se sube el archivo *.war* que empaqueta los servicios web a la carpeta correspondiente dentro del sistema de archivos de *Tomcat*, utilizando para ello el cliente *FTP Filezilla* [54]. Una vez realizado este paso, solo es necesario subir mediante *PhpMyAdmin* el archivo *.sql* que ejecutará las sentencias necesarias para crear las tablas de la base de datos.

Por último, solo resta iniciar los servidores *Tomcat* y *MySQLServer* en la instancia *EC2* y la aplicación está disponible, haciendo que las aplicaciones móviles puedan acceder a estos nuevos servicios web.

Rendimiento

Como ya se hizo con la aplicación Itinerarios Guiados Parque del Retiro, en esta última sección se hace hincapié en las mejoras de rendimiento de la aplicación tras los cambios aplicados.

Spring ha demostrado ser un *framework* que optimiza los recursos con los que cuenta, creando aplicaciones web ágiles, y al ser de código abierto, siempre está a la última en cuanto a las novedades que puedan aparecer para mejorar su rendimiento. Además *Tomcat*, es un contenedor de *servlets* muy ligero y veloz, y ha demostrado que da perfectamente la talla en entornos de producción con un gran número de usuarios haciendo uso simultáneo de sus servicios. Por otra parte, es posible que JPA no ofrezca un rendimiento excelente a la hora de acceder a base de datos, pero su facilidad de uso y su escalabilidad hacen que merezca la pena perder algo de eficiencia al realizar consultas con un gran número de resultados.

Pero lo que realmente potencia el rendimiento de nuestra aplicación es la utilización de los servicios web *REST*. Como ya se explicó en capítulos anteriores, esta arquitectura se centra en el uso del estándar *HTTP* para la

transmisión de datos, por lo que no es necesario contar con una capa adicional que ralentice cada transacción. Además el uso de *JSON* en vez de *XML* como contenedor de la información, hace que las comunicaciones sean más veloces. Si a esto le unimos el rendimiento de los servidores de Amazon, cuyas instancias *EC2* se adaptan al tráfico que van recibiendo, se puede concluir que esta aplicación no tendrá problemas de rendimiento aunque el número de usuarios de la aplicación crezca durante los próximos años.

Capítulo 4. INTEGRACIÓN DE LAS APLICACIONES EN EL IAM. RESULTADOS

Se dejan, por tanto, las tres aplicaciones dispuestas de la siguiente manera: las dos relativas al reciclaje de productos (aplicaciones que requieren servicios web) alojadas en servidores de Amazon, y pendientes de unas pequeñas modificaciones para poder ser subidas a WebSphere, una vez ahí estarían en condiciones de implantarse en los servidores del Ayuntamiento; Itinerarios Guiados Parque del Retiro queda completamente finalizada y no requiere modificación alguna.

Con el fin de poder integrar estas aplicaciones dentro del marco del IAM en un futuro, se detallan, a continuación, los pasos pertinentes y los requisitos que esta institución demanda. Dichos requisitos son responsabilidad del Comité de Estándares del IAM, órgano encargado de la redacción de estos documentos, así como de la atención en caso de que surja cualquier duda o sugerencia, de la gestión de las peticiones de reunión, del estudio de los informes de incompatibilidad de los diferentes proyectos con las directrices de la casa y del estudio de las peticiones de adaptación de algunos estándares ya definidos en casos muy concretos.

4.1 Infraestructura tecnológica

Se describe aquí la infraestructura tecnológica estándar sobre la que se deben construir los servicios para este Organismo. Los desarrollos realizados sobre entornos distintos a los descritos a continuación deberán ser presentados al Comité de Estándares como se ha mencionado anteriormente [55].

Los Sistemas Operativos con los que se debe trabajar son:

- Windows Server 2003 R2 SP2
- Red Hat Enterprise Linux V5.4

En cuanto a los Servidores de Aplicaciones (entorno Windows):

- IBM WebSphere Application Server Network Deployment V6.1.0.33 en un entorno *cluster*.
- JRE 1. 5. 0 IBM en Windows Server 2003
- J2EE 1.4
- IBM WebSphere Application Server Network Deployment V8.0.0.3 en un entorno *cluster*.
- JRE 1.6.0 IBM en Windows Server 2003
- Java EE 6

Para la gestión de la Bases de Datos se ha de hacer uso de Microsoft SQL Server 2005 SP2. De manera excepcional, si es necesario utilizar Oracle en su versión 10.2.0.4 y habría que pedir para ello autorización al Comité de Estándares.

Herramientas

- Rational Software Architect (de IBM) v8: Esta herramienta permite el diseño de modelos de UML tanto para despliegues, como para casos de uso y análisis. También es posible la codificación y el empaquetado de aplicaciones. Es necesario incluir en local las versiones que se demandan del servidor de aplicaciones WAS que antes mencionamos.
- Maven 3.0.4, para la construcción de aplicaciones.
- Subversion como gestor de versiones.
- TortoiseSVN como software de acceso al repositorio desde el explorador de archivos.

4.2 Ejecución y seguimiento del proyecto

Todo proyecto vinculado al IAM requiere una fase previa de propuesta y autorización que se lleva a cabo con los responsables de la corporación. Acto seguido, se mantiene una reunión de inicio del proyecto, a la que asisten representantes de las distintas unidades de trabajo del IAM (Calidad,

Sistemas, jefes de proyectos, etc.), durante la que se acuerda qué tipo de proyecto corresponde a lo que se pretende desarrollar, se determinan los criterios de calidad a aplicar y las características del proyecto (servidor de aplicaciones, plan de prueba, componentes comunes, etc.), quedando todo reflejado en el acta de la reunión.

Si es preciso se realiza un análisis del proyecto para definir si contiene subsistemas. Existen subsistemas siempre que se estima que hay partes independientes atendiendo a su despliegue, pero con código compartido. Esto facilita la organización del proyecto si este adquiere cierta complejidad, dividiéndolo en módulos de entrega individual, lo que permite avanzar más rápidamente al poder desarrollar cada una de las partes a diferente ritmo.

Esto último responde al Modelo de tres capas [56] antes mencionado, ya que cabe la posibilidad de que algunas aplicaciones compartan varias partes del negocio, pero su capa de presentación sea completamente distinta. Concretamente, esto sucede en nuestro proyecto con las aplicaciones Recycla.me y Recyclate!, un tema que se trató a fondo en el Capítulo 3 de la presente memoria. La característica más interesante de este análisis por subsistemas es la posible evolución y mejora de cada uno de los módulos de forma independiente, atendiendo a sus ciclos de vida y a las necesidades que puedan surgir en el futuro.

Para poder cumplir con los criterios del Departamento de Calidad es preciso hacer uso de las librerías y los arquetipos especificados por el IAM, los cuales serán revisados por este departamento en cada una de las subidas hasta que la entrega sea satisfactoria.

4.3 Política de certificados

Tanto para acceder a los repositorios de artefactos, como para efectuar las subidas de las sucesivas versiones de los proyectos es necesario obtener un certificado cliente expedido por las autoridades emisoras reconocidas por el Ayuntamiento de Madrid, aquí listadas:

- FNMT.
- DNIe.

- Firma Profesional.
- Asociación Nacional de Fabricantes - ANF.
- Agencia Notarial de Certificación – ANCERT.
- Camerfirma.
- Agencia de Tecnología y Certificación Electrónica - ACCV.
- Colegio de Registradores.
- APE.

El certificado que emite la Fábrica Nacional de la Moneda y Timbre (FNMT) es el más común y sencillo de conseguir, en documento *Solicitud de un certificado de Usuario* [57] se explican detalladamente los pasos a seguir para hacerse con este tipo de certificado.

4.4 Acceso a los repositorios de arquetipos

Para gestionar sus repositorios el IAM hace uso de la herramienta Artifactory [58]. Artifactory es un repositorio de artefactos en el que se aglutinan todas las bibliotecas que el IAM permite utilizar para desarrollar aplicaciones Java.

A cada proyecto se le asigna un usuario y una contraseña, de este modo, al acceder al repositorio con las claves de un proyecto sólo se podrán observar aquellas bibliotecas accesibles para ese usuario.

Se puede navegar por los distintos repositorios a los que se tiene permiso de acceso y realizar búsquedas en ellos. Existe la posibilidad de realizar una navegación en formato árbol o hacer una navegación directa, así como llevar a cabo búsquedas según el nombre del artefacto (*Quick Search*), una clase concreta de dicho artefacto (*Class Search*), las coordenadas del archivo POM de Maven (GAVC *Search*) o según la codificación md5 del artefacto (*CheckSum Search*).

Una vez realizada la búsqueda, simplemente con posicionarse mediante el cursor encima del artefacto aparece un desplegable con la opción

"*Download*", al pulsar sobre ese botón el artefacto se descarga en nuestro local.

En la actualidad existen los siguientes repositorios de carácter general:

- *ext-libs-desa*: En este repositorio se almacenan las librerías externas que se encuentran dentro del estándar del IAM para aplicaciones de desarrollo y a él tienen acceso todas las aplicaciones del IAM.

- *ext-libs-legacy*: En este repositorio se almacenan las librerías externas que se encuentran fuera del estándar del IAM para aplicaciones de desarrollo y a él sólo tienen acceso las aplicaciones del IAM antiguas, es decir, que históricamente usan librerías fuera del estándar.

- *iam-libs-desa*: En este repositorio se almacenan las librerías desarrolladas por el IAM que se encuentran dentro de su estándar para aplicaciones de desarrollo y a él tienen acceso todas las aplicaciones del IAM.

- *iam-libs-legacy*: En este repositorio se almacenan las librerías desarrolladas por el IAM que se encuentran fuera de su estándar para aplicaciones de desarrollo y a él sólo tienen acceso las aplicaciones del IAM antiguas, es decir, que históricamente usan librerías fuera del estándar.

- *ext-plugin-desa*: En este repositorio se almacenan las librerías externas necesarias para ejecutar los *plugins* de Maven.

- *iam-plugin-desa*: En este repositorio se almacenan las librerías desarrolladas en el IAM necesarias para ejecutar los *plugins* de Maven.

- *ext-libs-was61*: En este repositorio se incluirán las bibliotecas de terceros que están aprobadas para su uso solo en WAS 6.1.

- *iam-libs-was61*: En este repositorio se incluirán las bibliotecas desarrolladas por el IAM que están aprobadas para su uso solo en WAS 6.1.

- *ext-libs-was85*: En este repositorio se incluirán las bibliotecas que están aprobadas para su uso solo en WAS 8.5.

- *ext-libs-jboss*: En este repositorio se incluirán las bibliotecas que están aprobadas para su uso solo en JBoss.

4.5 Criterios de entrega y subidas

Como ya se ha mencionado con anterioridad, para dar por finalizada una entrega es necesario situarla en la herramienta corporativa Subversion [59] y esperar a que la Unidad de Calidad del IAM verifique el correcto cumplimiento de los estándares. Una vez depositado el proyecto en los repositorios, se comunica a esta unidad la ruta del entregable y el nombre de la etiqueta con la que este se introdujo. Tras esta petición de entrega, el Departamento de Calidad verifica que los requisitos se cumplen y notifica a los solicitantes el estado en el que se haya el entregable, asignando un porcentaje de cumplimiento de estos estándares. Las auditorías que realiza la Unidad de Calidad no son bloqueantes para realizar subidas, excepto en casos de carácter grave.

Los estados posibles de los entregables incluidos en el Informe de la Unidad de Calidad son:

- PENDIENTE: El entregable no ha sido revisado por la Unidad de Calidad.
- NO APLICA: El entregable no se adecúa a los requisitos del proyecto en cuestión.
- NAACORD: (NO APLICA ACORDADO) El entregable sí que debería realizarse, pero se acuerda no realizarlo con la jefatura de Proyecto del IAM.
- Valor numérico: Porcentaje de cumplimiento del entregable respecto a la metodología definida en la *Guía de Estándares del IAM*.

Cada proyecto tiene su propio repositorio dentro de SVN, atendiendo a la siguiente estructura:

[Acrónimo del proyecto]

Trunk

Tags

Branches

En cada una de las carpetas del repositorio el proyecto ha de alojar lo siguiente:

- *Trunk* (tronco): La línea principal del desarrollo. Es en esta carpeta en la que normalmente se actualizan los cambios.
- *Branches* (ramas): Las copias/ramas. En el caso en que se deban mantener varias versiones en paralelo, se utilizarán las "ramas" para gestionar las distintas líneas de desarrollo concurrentes.
- *Tags* (etiquetas): Las etiquetas, versiones de entrega al IAM tanto parciales como totales.

Todas ellas tienen la misma estructura interna: Fuentes, Documentación y Despliegues. Los anexos hablan del uso del SVN para RSA no de RAD.

Capítulo 5. CONCLUSIONES Y TRABAJO FUTURO

5.1 Conclusiones

Tras finalizar este proyecto nos sentimos muy satisfechos con los resultados conseguidos durante este curso.

Consideramos que el estado de las tres aplicaciones con las que hemos trabajado en nuestro proyecto, es mejor, en general, que el que nos encontramos, y sobretodo cumplen con los estrictos requerimientos propuestos por el Ayuntamiento de Madrid. Dejamos unas aplicaciones que están completamente preparadas para su subida a los servidores del IAM y su publicación oficial en un futuro próximo, además consideramos que ahora mismo estás aplicaciones son muchos más escalables y fáciles de actualizar por nuevos desarrolladores que trabajen con ellas en un futuro.

Respecto a la parte del servidor, también devolvemos una aplicación web totalmente escalable y preparada para nuevas modificaciones, y que ofrece un rendimiento a la altura de aplicaciones web profesionales.

En el aspecto personal ha sido una satisfacción para todos nosotros trabajar en este proyecto de Sistemas Informáticos. Al comenzar este año académico, no estábamos familiarizados con prácticamente ninguna de las tecnologías con las que hemos desarrollado nuestro proyecto, y el reto de trabajar con una institución como el Ayuntamiento de Madrid suponía una motivación muy grande, pero nos hacía darnos cuenta de la gran cantidad de trabajo que teníamos por delante. Por otra parte, era la primera vez que trabajamos para un cliente que nos marcaba unos objetivos y un camino a seguir, esto nos ha hecho ganar una experiencia impagable. Además tratar con estándares tan altos como con los que trabaja el IAM, han supuesto otro gran reto que nos ha hecho superarnos y ponernos en contacto con tecnologías punteras en España, lo que creemos que nos abrirá puertas en nuestro futuro en el mercado laboral.

Tener la posibilidad de desarrollar tanto la parte del servidor, como modificar las propias aplicaciones Android nos ha permitido comprender y dominar cada una de las partes, sintiéndonos así capaces de desarrollar nuestras propias aplicaciones en un futuro de una manera totalmente independiente, obteniendo resultados profesionales.

Por todos los motivos expuestos y al ver que hemos alcanzado los objetivos marcados sentimos que hemos dado un gran paso adelante en nuestra evolución como futuros ingenieros.

5.2 Trabajo Futuro

En cuanto al trabajo futuro, respecto a las aplicaciones Itinerarios del Retiro, Recycla.me y Recyclate! solo quedaría subirlas a los servidores del IAM, realizando alguna mínima modificación.

Por otra parte, y aunque el IAM este año nos recomendó centrarnos en el desarrollo de la parte servidora y la adaptación de las aplicaciones Android, parece que el siguiente paso a dar en el futuro es desarrollar estas aplicaciones para otros sistemas operativos móviles, para lo que en esta memoria ya hemos dado alguna idea acerca de cómo plantear esa tarea.

Parece claro que la aplicación a la que se le deben dedicar más esfuerzos es Hábitat, que podría ser ampliada en diferentes aspectos:

- Desarrollo de la parte servidora y los servicios web necesarios cumpliendo los estándares expuesto en el punto 2 de esta memoria
- Implementación de un sistema de recomendación de actividades
- Implementación de un sistema de alertas para el usuario
- Por último, y no menos importante, habría que solucionar los distintos fallos que tiene la aplicación y completar las funcionalidades ya existentes.

Capítulo 6. BIBLIOGRAFÍA

[1] IAM - http://www.madrid.es/portales/munimadrid/es/Inicio/Buscador-Avanzado/ANM-2004-37-Organismo-Autonomo-Informatica-del-Ayuntamiento-de-Madrid-I-A-M-?vgnextfmt=default&vgnextoid=3a927b3cb0e4f010VgnVCM1000009b25680aRCRD&vgnextchannel=102f43db40317010VgnVCM100000dc0ca8c0RCRD&idioma=es&idiomaPrevio=es

[2] Itinerarios Guiados Parque del Retiro. Memoria del proyecto Itinerarios Guiados Parque del Retiro.

[3] Recycla.me. Memoria del proyecto Recycla.me.

[4] Recyclate!. Memoria del proyecto Recyclate!.

[5] Camino seguro al cole. Memoria del proyecto Camino seguro al cole.

[6] Hábitat Madrid. Memoria del proyecto Hábitat Madrid.

[7] Mapa de recursos de la Comunidad de Madrid. Memoria del proyecto Mapa de recursos de la Comunidad de Madrid.

[8] Software Reuse and Software Component Technology. Yang Fuqing, Mei Hong, Li Keqin (Dept.of Computer Science & Technology, Peking University, Beijing) 1999 - http://en.cnki.com.cn/Article_en/CJFDTOTAL-DZXU902.018.htm

[9] Software Reuse. Charles W. Krueger (School of Computer Science, G'arnegie Mellon University, Pittsburgh, Pennsylvania) 1992 - http://dl.acm.org/citation.cfm?id=130856

[10] Success and failure factors in software reuse. IEEE Transactions on Software Engineering 2002 - http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=995420&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D995420

[11] Formalizing Dynamic Software Updating. Gavin Bierman, Michael Hicks, Peter Sewell, Gareth Stoyle 2003 - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.7176

[12] Java EE - Beginning Java™ EE 6 Platform with GlassFish™ 3, Second Edition. Antonio Goncalves.

[13] JSR - https://jcp.org/en/jsr/overview

[14] Open Source J2EE Frameworks - http://java-source.net/open-source/j2ee-frameworks

[15] Proyecto “Madrid a pie: camino seguro al cole” - http://www.madrid.es/portales/munimadrid/es/Inicio/Ayuntamiento/Medio-Ambiente/Educacion-ambiental/Madrid-a-pie--camino-seguro-al-cole?vgnextfmt=default&vgnextoid=173b3b3e1598b310VgnVCM1000000b205a0aRCRD&vgnextchannel=378c9ad016e07010VgnVCM100000dc0ca8c0RCRD

[16] Educación Ambiental del Ayuntamiento de Madrid - http://www.madrid.es/portales/munimadrid/es/Inicio/Ayuntamiento/Medio-Ambiente/Educacion-ambiental?vgnextfmt=default&vgnextchannel=378c9ad016e07010VgnVCM100000dc0ca8c0RCRD

[17] Servidor HTTP Apache (2.0)- http://httpd.apache.org/docs/2.0/es/

[18] Servicios SSH - http://es.wikipedia.org/wiki/Secure_Shell

[19] Librería Zxing - zxing.org

[20] Librería Android ViewPagerIndicator - http://viewpagerindicator.com/

[21] APLICACIONES MÓVILES: ¿NATIVO, WEB, HÍBRIDO? - http://www.pixmatstudios.com/blog/aplicaciones-moviles-nativo-web-hibrido/#.U5h-kvl_sSo

[22] TIPOS DE APPS: NATIVAS, HÍBRIDAS Y WEB APPS - http://www.appio.es/tipos-de-apps/

[23] Phonegap - http://phonegap.com/ - http://es.wikipedia.org/wiki/PhoneGap

[24] What's the Difference Between Icenium and PhoneGap Build? - http://blogs.telerik.com/appbuilder/posts/13-05-01/what's-the-difference-between-icenium-and-phonegap-build-

[25] Appcelerator - http://www.appcelerator.com/

[26] Anexo 15. Manual de uso de Artifactory y Maven. IAM.

[27] Pagina de Apache sobre los requisitos software de Maven http://maven.apache.org/what-is-maven.html

[28] Memoria del proyecto: Mapa de Recursos de la Comunidad de Madrid.

[29] Modelo 3 capas - http://www.jtentor.com.ar/post/Arquitectura-de-N-Capas-y-N-Niveles.aspx

[30] Spring Framework http://en.wikipedia.org/wiki/Spring_Framework y http://spring.io/

[31] EJB http://es.wikipedia.org/wiki/Enterprise_JavaBeans y http://www.arquitecturajava.com/introduccion-a-ejb-3-1-i/

[32] DAO - http://es.wikipedia.org/wiki/Data_Access_Object

[33] Spring Framework: El patrón DAO - http://www.genbetadev.com/java-j2ee/spring-framework-el-patrn-dao

[34] Spring Web MVC framework - http://docs.spring.io/spring-framework/docs/2.0.x/reference/mvc.html

[35] Modelo Vista Controlador - es.wikipedia.org/wiki/Modelo–vista–controlador

[36] JPA Persistencia - http://es.wikipedia.org/wiki/Java_Persistence_API

[37] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[38] http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/96

[39] Comparativa de implementaciones de JPA: Toplink, EclipseLink, Hibernate y OpenJPA http://terrazadearavaca.blogspot.com.es/2008/12/comparativa-de-implementaciones-de-jpa.html

[40] Hibernate + JPA http://docs.jboss.org/hibernate/ogm/4.1/reference/en-US/html/ogm-configuration.html

[41] POJO http://es.wikipedia.org/wiki/Plain_Old_Java_Object

[42] MySQL http://es.wikipedia.org/wiki/MySQL

[43] Licencia GNU http://es.wikipedia.org/wiki/Licencia_p%C3%BAblica_general_de_GNU

[44] PhpMyAdmin http://es.wikipedia.org/wiki/PhpMyAdmin y http://www.PhpMyAdmin.net/home_page/index.php

[45] Servicios web RESTfull - http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-RESTful.html

[46] Servidor de aplicaciones http://es.wikipedia.org/wiki/Servidor_de_aplicaciones

[47] WebSphere Application Server - http://es.wikipedia.org/wiki/WebSphere_Application_Server y http://www-03.ibm.com/software/products/es/appserv-was

[48] Servidor Apache Tomcat - http://www.fdi.ucm.es/profesor/jpavon/web/42-tomcat.pdf y http://tomcat.apache.org/

[49] Android Mediaplayer - http://developer.android.com/reference/android/media/MediaPlayer.html

[50] Image Loader https://github.com/nostra13/Android-Universal-Image-Loader

[51] API google maps https://developers.google.com/maps/documentation/android/start#installing_the_google_maps_android_v2_api

[52] Amazon Web Services - http://aws.amazon.com/es/

[53] EC2 - http://aws.amazon.com/es/ec2/

[54] FTP Filezilla - https://filezilla-project.org/

[55] Guía de Estándares. Informática del Ayuntamiento de Madrid.

[56] Anexo 12. Gestión de Subproyectos - Informática del Ayuntamiento de Madrid.

[57] Solicitud de un Certificado de Usuario. Soporte Técnico, Fábrica Nacional de Moneda y Timbre.

[58] Anexo 15. Manual de uso de artifactory y Maven - Informática del Ayuntamiento de Madrid.

[59] Anexo 10. Uso SVN y Anexo 8. Normas de empaquetado y entrega - Informática del Ayuntamiento de Madrid.