Facultad
de
Ciencias

DESARROLLO DE UNA APLICACIÓN MÓVIL PARA LA GESTIÓN DE LA ITINERANCIA EN DISPOSITIVOS ANDROID

(Development of a mobile application for roaming management in Android devices)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Rebeca Elena Bárcena Orero
Director: Marta Elena Zorrilla Pantaleón
Junio - 2016

Índice de contenidos

Índice de tablas

Índice de figuras

Agradecimientos

En primer lugar, mi agradecimiento a toda mi familia y amigos, por animarme siempre y haber sido mi mayor apoyo.

En segundo lugar, a ConMymo, por haber confiado en mí para llevar a cabo este proyecto y haberme aportado sus conocimientos.

Y, por último, pero no menos importante, a todos los profesores que han sabido motivarme y enseñarme para llegar hasta aquí, y en especial a la profesora Marta Zorrilla, por haberme guiado y orientado en el desarrollo de este proyecto.

Resumen

En la actualidad, la mayoría de los directivos y comerciales de las empresas disponen de *Smartphones* para llevar a cabo sus funciones. Además, muchos de ellos deben viajar al extranjero, por lo que conviene a las empresas comprar bonos de itinerancia (*roaming*) para que los costes relativos a comunicación sean contenidos. Dos días en el extranjero utilizando el móvil de manera usual puede suponer miles de euros en la factura. Por ello, cada vez que un trabajador viaje al extranjero, es necesario que active el bono de *roaming*, así como que lo desactive cuando vuelve a su país de origen. En las grandes organizaciones, en general, suele haber un encargado al que hay que comunicarle el bono que se quiere activar o desactivar, en función al país al que se viaje, para que, a su vez, se lo solicite a la empresa de telecomunicaciones suministradora del bono.

La realidad es que a muchos trabajadores se les olvida avisar al encargado de que tiene que activar o desactivar un bono o que el propio encargado se olvide hacerlo lo que provoca gastos elevados innecesarios. Con objeto de solucionar o, al menos, minimizar estos costes, se propone desarrollar una aplicación móvil para que los empleados planifiquen sus viajes, avisando por SMS o correo electrónico al encargado. Además, en caso de que el trabajador se le olvidase informar de su viaje, la aplicación lo avisaría en el momento en el que saliese de su país, permitiendo establecer ajustes de seguridad para evitar gastos innecesarios, incluyendo entre ellos, desactivar la itinerancia de datos.

La aplicación se desarrollará en Android nativo siguiendo un patrón de 3 capas.

Palabras clave: itinerancia de datos, aplicación móvil, Android

Abstract

Actually, most managers and commercials from companies have Smartphones in order to carry out their duties. Many of them must travel abroad, so buying roaming bonds suit companies so as to costs related to communication are restrained. Two days abroad using the mobile quite often can mean thousands of euros on the bill. Therefore, each time a worker travels abroad, it is necessary to activate the roaming bond, and disable it when he returns to his country of origin. In large organizations, generally, there is an attendant who must be informed when the bond should be turned on or off, depending on the country the worker is travelling to, in turn he must request the telecommunications company which supplies the bond.

The reality is that many workers forget to tell the attendant that he has to enable or disable the bond, or even the attendant himself forgets to do so, causing unnecessary high costs. In order to solve or, at least, minimize these costs, it is proposed to develop a mobile Application for Android Operating System to plan their trips, warning the attendant by SMS or email. In addition, if the worker forgets to plan the trip, the application will warn at the time he leaves his country, allowing to establish security settings to avoid unnecessary costs, including disable data roaming.

The application will run in native Android using a three layer pattern.

Keywords: roaming, mobile app, Android

1. Introducción

1.1 Antecedentes y objetivos

Hoy en día el uso de *Smartphones* está muy extendido, llegando a casi igualar el número de estos dispositivos al de personas en el mundo.

Los *Smartphone*, hoy en día, son pequeños ordenadores que podemos llevar en nuestro bolsillo, teniendo una enorme cantidad de aplicaciones para facilitarnos nuestra rutina diaria. Como han demostrado muchos estudios [1], las aplicaciones más usadas siguen siendo las de mensajería instantánea, por lo que la mayoría de los usos de los teléfonos inteligentes requieren de una conexión a Internet.

Es por esto que, cuando se viaja, uno debe estar pendiente de apagar la itinerancia de datos para evitar gastos elevados y quedarse sin internet, u optar por contratar tarifas o bonos más baratos que las propias teleoperadoras ofrecen.

La utilización de bonos de *roaming* está, prácticamente, generalizado en las empresas que dan a sus empleados móviles y en las cuales se viaja al extranjero con frecuencia, siendo un empleado de la empresa quién ha de encargarse de gestionar la activación y desactivación de estos bonos cuando los trabajadores le comunican sus viajes.

Al no tener un sistema que permita gestionar fácilmente estas peticiones y en caso de olvido, disponer de alarmas que avisen al usuario de este hecho, se hace necesario programar un aplicativo móvil, que evite a las empresas este tipo de gastos, además de facilitar a los empleados la labor de la activación y desactivación de los bonos de *roaming*.

Este proyecto se enmarca como un producto de la empresa ConMymo. Esta empresa ya cuenta con su servidor y base de datos para procesar y almacenar la información con la que trabaja, por lo que la app tendrá que trabajar con ambos para realizar algunas de sus funcionalidades. La aplicación se desarrolló sobre el sistema operativo Android, puesto que tiene una cuota de mercado cercana al 85% [2], eligiendo Android Studio para su desarrollo por ser el IDE oficial de Android.

1.2 Metodología

En este proyecto se contaba con dos clientes, uno de ellos actuaba a su vez como jefe de proyecto, y una sola persona encargada de realizar la aplicación móvil. Puesto que la metodología seguida en la empresa era Scrum, se decidió que este proyecto también se apoyaría en dicha metodología, pero con algunos cambios debido al pequeño tamaño del equipo de trabajo.

El proceso de esta metodología, el cual se muestra en la Figura 1.1, se explica brevemente a continuación.

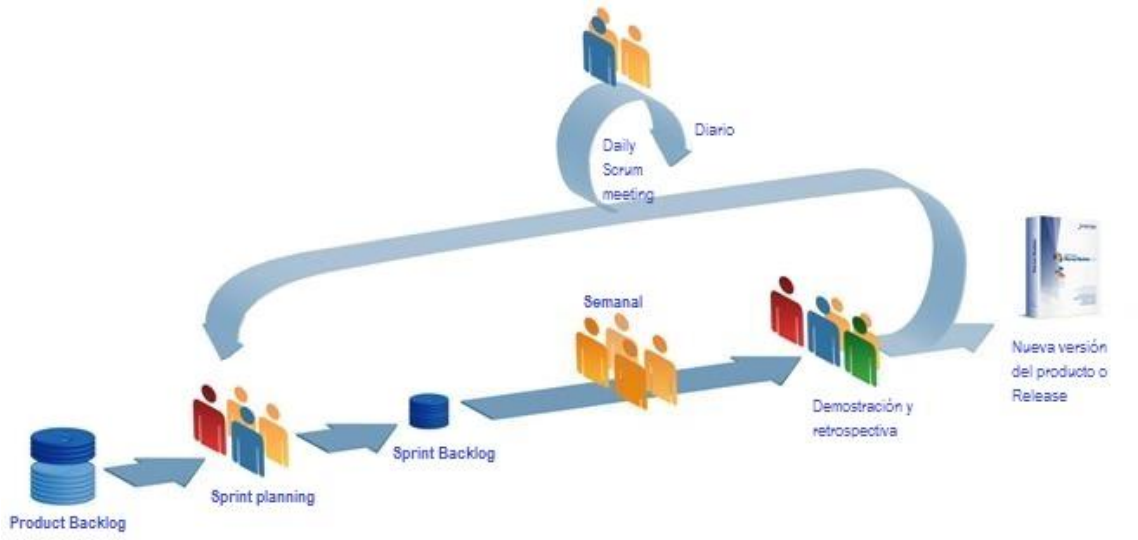


Figura 1.1. Proceso de la metodología Scrum

Partiendo de un *Product Backlog*, que contiene todos los requisitos del producto que se deben implementar, semanalmente se realizaba un *Sprint planning,* en el cuál se decidían los requisitos y tareas que se iban a llevar a cabo dicha semana (*Sprint backlog*).

Como ya se ha mencionado, hubo una programadora y dos clientes, los cuales participaban a veces en el propio proyecto, encargándose de la programación de los artefactos del servidor que fuesen necesarios para el correcto funcionamiento de la app. Además, el resto del tiempo, ellos se encargaban de sus labores comerciales dentro de su propia empresa. Por ello, los *Daily Scrum Meetings* consistieron en reuniones breves diarias en las que cada cual comentaba lo realizado el día anterior, los problemas encontrados y los propósitos para esa jornada. De las veinte semanas que ha supuesto el desarrollo del producto, cinco se realizaron en el entorno de la empresa y el resto fuera de ella. Como consecuencia de esto, los *Daily Scrum Meetings* se convirtieron en una "retrospectiva" para la desarrolladora, en la que listaba qué tareas habían quedado pendientes, organizaba el trabajo de ese día y solucionaba los problemas que hubiesen surgido en la jornada anterior.

Durante la estancia en la empresa, se mostraba semanalmente a los clientes el trabajo realizado y se comprobaba qué partes del trabajo semanal habían sido cumplidas satisfactoriamente, cuáles había que mejorar, los problemas surgidos y qué se había aprendido. Durante el tiempo de desarrollo fuera de la empresa esta tarea fue realizada únicamente por la desarrolladora.

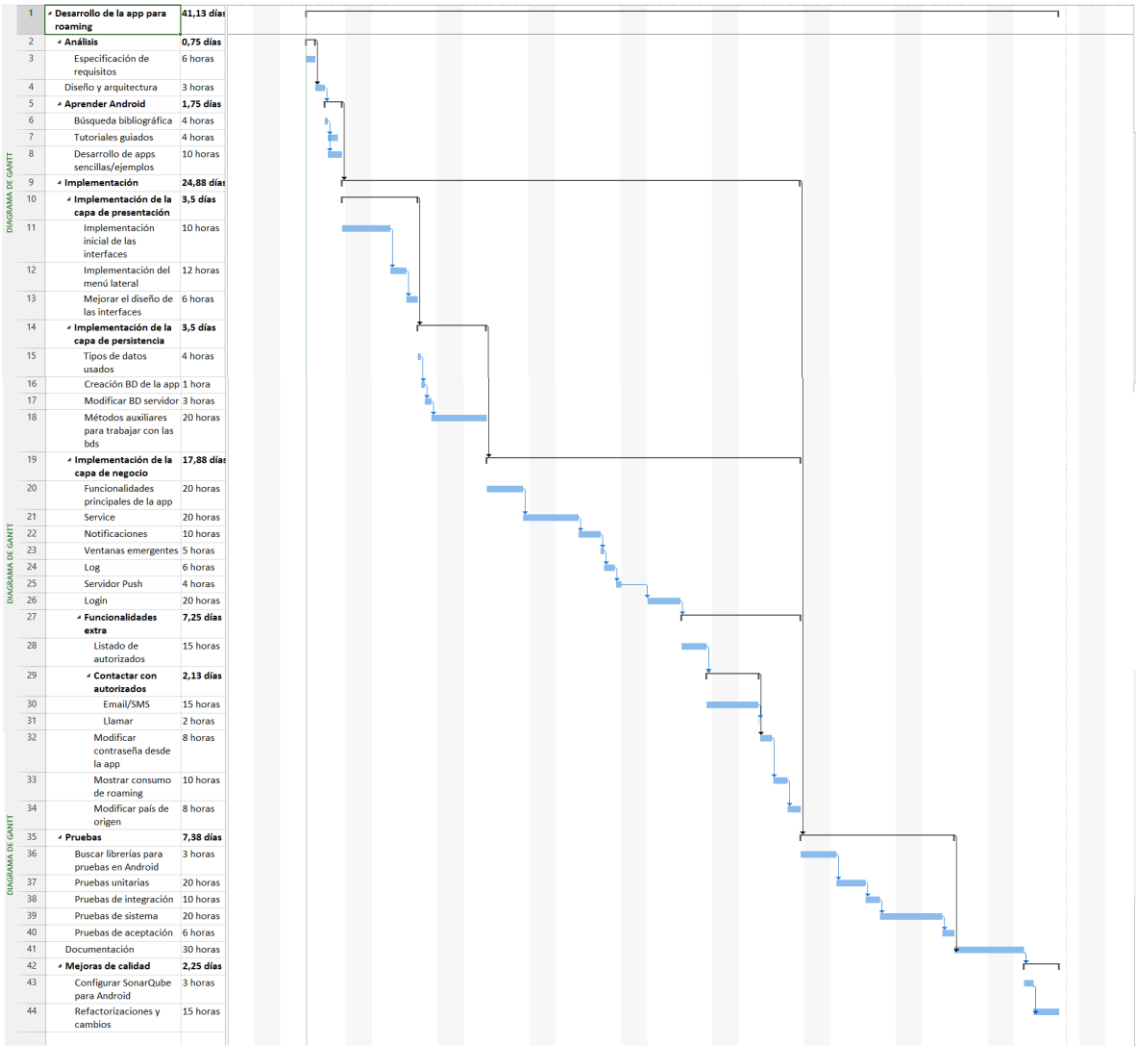
La primera semana de junio se hizo entrega de la última *release* a los clientes, con la aplicación completada. Sobre esta *release* se han llevado a cabo las pruebas de aceptación de los clientes.

Por último, hay que destacar que la lista de requisitos iniciales ha sufrido modificaciones, bien porque la empresa ConMymo consideró necesario incorporar nuevos requisitos o bien porque así lo opinaba la desarrolladora y lo aceptaban los clientes. En general, estos han sido añadidos y/o mejoras, nunca una modificación del núcleo principal.

1.3 Plan de trabajo y presupuesto

A continuación, se muestra el plan de trabajo que se diseñó para llevar a cabo este proyecto. El plan de trabajo se estableció en 300h, como se recoge en el diagrama de Gantt de la Figura 1.2, en el que también se pueden observar las tareas desglosadas y su estimación en horas.

Dado que las tareas las realizó una única persona, a veces se paralelizaron tareas o cambiaron de orden. El plan previsto se cumplió en la mayoría de los casos, habiendo ligeras variaciones de un par de horas como mucho.



1	**Desarrollo de la app para roaming**	**41,13 días**
2	**Análisis**	**0,75 días**
3	Especificación de requisitos	6 horas
4	Diseño y arquitectura	3 horas
5	**Aprender Android**	**1,75 días**
6	Búsqueda bibliográfica	4 horas
7	Tutoriales guiados	4 horas
8	Desarrollo de apps sencillas/ejemplos	10 horas
9	**Implementación**	**24,88 días**
10	**Implementación de la capa de presentación**	**3,5 días**
11	Implementación inicial de las interfaces	10 horas
12	Implementación del menú lateral	12 horas
13	Mejorar el diseño de las interfaces	6 horas
14	**Implementación de la capa de persistencia**	**3,5 días**
15	Tipos de datos usados	4 horas
16	Creación BD de la app	1 hora
17	Modificar BD servidor	3 horas
18	Métodos auxiliares para trabajar con las bds	20 horas
19	**Implementación de la capa de negocio**	**17,88 días**
20	Funcionalidades principales de la app	20 horas
21	Service	20 horas
22	Notificaciones	10 horas
23	Ventanas emergentes	5 horas
24	Log	6 horas
25	Servidor Push	4 horas
26	Login	20 horas
27	**Funcionalidades extra**	**7,25 días**
28	Listado de autorizados	15 horas
29	**Contactar con autorizados**	**2,13 días**
30	Email/SMS	15 horas
31	Llamar	2 horas
32	Modificar contraseña desde la app	8 horas
33	Mostrar consumo de roaming	10 horas
34	Modificar país de origen	8 horas
35	**Pruebas**	**7,38 días**
36	Buscar librerías para pruebas en Android	3 horas
37	Pruebas unitarias	20 horas
38	Pruebas de integración	10 horas
39	Pruebas de sistema	20 horas
40	Pruebas de aceptación	6 horas
41	Documentación	30 horas
42	**Mejoras de calidad**	**2,25 días**
43	Configurar SonarQube para Android	3 horas
44	Refactorizaciones y cambios	15 horas

Figura 1.2. Diagrama de Gantt del proyecto

En cuanto al presupuesto, este trabajo se ha realizado disfrutando de la Beca de prácticas Santander CRUE-CEPYME.

2. Tecnologías y herramientas usadas

En este capítulo se describirán las tecnologías, herramientas y metodologías usadas a lo largo de todo el proyecto.

2.1 Tecnologías

2.1.1 Wi-Fi

Wi-Fi, o *Wireless Fidelity*, es la tecnología usada en una red o conexión inalámbrica para comunicar datos entre equipos situados dentro de una misma área de cobertura. La diferencia respecto a una red cableada es que los datos se transmiten y reciben a través de ondas electromagnéticas. En el caso de Android, permite a los *Smartphones* conectarse a Internet a través de un punto de acceso de red inalámbrica. En este proyecto se hace uso del Wi-Fi sólo para permitir, en ciertos casos, el uso de Internet cuando el *Smartphone* está en el extranjero.

2.1.2 Itinerancia de datos o Roaming

La itinerancia de datos o *Roaming* hace referencia al uso de datos de internet en un dispositivo móvil ajenos al operador de la SIM, lo que se traduce en poder usar Internet y realizar y recibir llamadas y mensajes cuando se está en el extranjero usando puntos de acceso pertenecientes a otras operadoras diferentes a la contratada [3].

En este proyecto, la itinerancia de datos es vital; se activará o desactivará en función del país en el que se esté y de si hay o no bonos contratados, con el fin de minimizar los costes.

2.1.3 Android

Android es un sistema operativo basado en el núcleo de Linux, diseñado para dispositivos móviles con pantalla táctil. En su estructura, la cual se muestra en la Figura 2.1, se distinguen cuatro bloques [4].

- Aplicaciones: esta es la capa superior, la cual utiliza la capa "*Application Framework*" para acceder a los servicios del dispositivo. En esta capa se encuentran las aplicaciones que vienen por defecto, como puede ser un cliente de correo electrónico, calendario, mapas, un navegador, etc. Todas las aplicaciones son escritas en lenguaje Java.
- *Application Framework* o marco de trabajo de aplicaciones: proporciona los servicios necesarios a las aplicaciones para que puedan funcionar correctamente y con todas sus funcionalidades. Los desarrolladores tienen acceso completo a las mismas APIs del framework usadas por las aplicaciones base.
- Librerías: conjunto de librerías escritas en C o C++ usadas por varios componentes del sistema. Algunas de estas librerías son bibliotecas de medios, de gráficos, 3D o SQLite.
- *Android Runtime*: es un conjunto de librerías con las que cuenta Android que proveen la mayor parte de las funcionalidades disponibles en las librerías del lenguaje Java.
- Núcleo de Linux: es la capa más baja y la más básica, y se corresponde al núcleo del sistema operativo. Actúa como una capa de abstracción entre el hardware y el resto de

la pila de software. Además, se encarga de los servicios base del sistema, como la seguridad, la gestión de memoria o la gestión de procesos, entre otros.

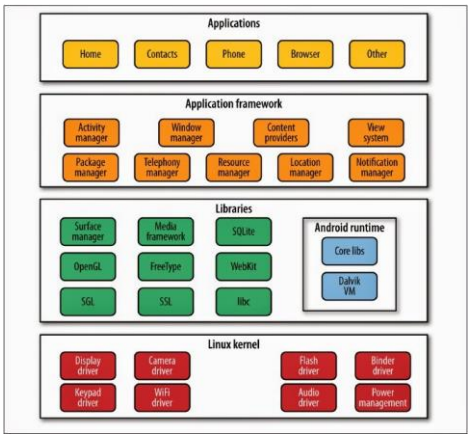


Figura 2.1. Estructura de Android

2.1.4 SQLite

SQLite es una librería software que proporciona un motor de bases de datos muy popular en la actualidad, por tener un tamaño pequeño, no requerir un servidor, precisar poca configuración, ser transaccional y de código libre [5]. Es una herramienta simple pero potente, que puede sustituir a SGBD en algunas tareas.

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, incluyendo una API para llevar a cabo, de forma sencilla, las tareas necesarias. Es por ello que, para almacenar algunos datos, en este proyecto se ha optado por usar SQLite.

2.1.5 MySQL

MySQL es un SGBD relacional, multihilo y multiusuario [6]. Destaca por su gran adaptación a diferentes entornos de desarrollo, permitiendo su interactuación con los lenguajes de programación más usados, como PHP o Java, y su integración en multitud de sistemas operativos. MySQL es, además, open source.

Para este proyecto, como ya se contaba con una base de datos MySQL en el servidor de la empresa, se ha de trabajar con ella. Esta se deberá de ampliar para recoger, entre otra información, los distintos bonos asociados a los países a los que se puede viajar o los clientes que tienen autorización para usar la aplicación.

2.1.6 Servicio de notificaciones Push a través de Google Cloud Messaging

La tecnología *Push* [7] es una forma de comunicación en la que una aplicación servidora envía un mensaje a un cliente-consumidor. En otras palabras, es un mensaje que un servidor envía a una persona alertándolo de que tiene una información nueva. Es el servidor el que inicia esta comunicación, siempre. Para que el cliente reciba la notificación *push*, previamente ha debido de suscribirse a las notificaciones en el servidor.

Para conseguir esto, se usará *Google Cloud Messaging* (GCM), que es un servicio gratuito proporcionado por Google que permite a los desarrolladores enviar mensajes entre los servidores y las aplicaciones cliente. Este se encarga de controlar todo lo relacionado con el almacenamiento en cola de los mensajes y su entrega a las aplicaciones. Las notificaciones llegan incluso cuando la aplicación no está ejecutándose. Sin embargo, GCM sólo pasa la información recibida a la aplicación, no tiene control sobre ella.

Para que funcione, es necesario que intervengan tres componentes: el dispositivo móvil con la aplicación que requiera las notificaciones *Push*, servidores GCM y un servidor de terceros (que será un servidor de la empresa).

Además, son necesarias otras cuatro credenciales:

- *Sender ID*: se usa en el proceso de registro para identificar la aplicación Android.
- *Application ID*: identificador de la aplicación que se está registrando para recibir mensajes. Se corresponde al nombre del paquete de la aplicación indicado en el archivo manifest.
- *Registration ID*: identificador enviado por los servidores GCM a la aplicación como respuesta a una solicitud de registro. Debe ser enviado al servidor de la empresa para identificar el dispositivo que se ha registrado para recibir notificaciones. Está asociado a una aplicación concreta en un dispositivo concreto.
- *Sender Auth Token*: clave que debe almacenar el servidor para poder acceder a los servicios de Google.

El proceso que se sigue [8], el cual se esquematiza en la Figura 2.2, se detalla a continuación:

1. La aplicación se registra en el dispositivo móvil para recibir mensaje, incluyendo el *Sender ID* y el *Application ID*.
2. Si el registro termina correctamente, el servidor GCM devuelve el *Registration ID*.
3. La aplicación envía el *Registration ID* al servidor de terceros para que sea guardado.
4. El servidor de terceros ya puede enviar un mensaje para que sea recibido por el dispositivo. El servidor envía el mensaje a los servidores GCM, donde es puesto en cola.
5. Google envía el mensaje al dispositivo cuando se encuentre accesible.
6. El dispositivo lanza el mensaje a la aplicación especificada y esta procesa el mensaje sin necesidad de que esté ejecutándose previamente.

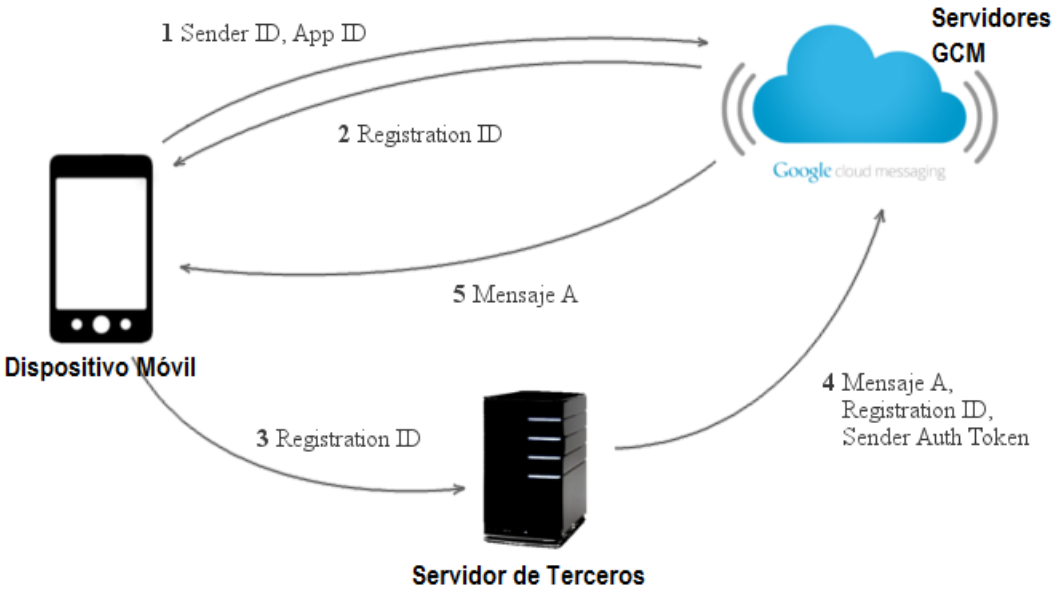


Figura 2.2. Proceso para enviar una notificación Push

En este proyecto, se emplean los mensajes *Push* para que el servidor de la empresa notifique al dispositivo móvil cuándo se ha activado o desactivado el bono, puesto que algunas empresas tardan unas horas o días desde que se realiza la solicitud hasta que se realiza.

2.1.7 BuzzBox

BuzzBox [9] es una librería que permite añadir un planificador de tareas en una aplicación Android. Usando unas pocas líneas de código permite añadir una tarea repetitiva en segundo plano, introduciendo los momentos en los que se tiene que realizar la tarea de forma similar a como se realiza en un sistema operativo Unix con cron. Simplemente se ha de hacer una llamada a BuzzBox, introducir como cadena de caracteres el día-hora en el que se ha de ejecutar la tarea e implementar una clase con el código de la tarea.

Además, dispone de una opción para analizar datos en tiempo real, que permite controlar, por ejemplo, cuándos usuarios están instalando y usando la aplicación.

2.1.8 Android Plot

Se trata de una librería que permite realizar gráficas de diferentes tipos: diagramas de barras, líneas, etc. [10].

En este trabajo, se ha utilizado para mostrar los datos de consumo del dispositivo cuando se encuentra con el roaming fuera del país.

2.1.9 Tank Auth

Tank Auth [11] es una librería de autenticación y manejo de usuarios de Codeigniter (ver sección 2.2.4), que cuenta con funcionalidades que permiten, entre otras cosas, que los usuarios se registren con su correo electrónico o un usuario y contraseña.

Esta librería Tank Auth es la utilizada por la empresa para tareas de autentificación y por ello, ha sido necesario trabajar con ella y sus funcionalidades.

2.2 Herramientas

2.2.1 Eclipse

Eclipse es un IDE de código abierto, desarrollado inicialmente por IBM, aunque ahora pertenece a Eclipse Foundation [12]. Cuenta con módulos o plug-ins [13] para proporcionar toda su funcionalidad, frente a otras plataformas en las que todas sus funcionalidades están ya incluidas, las necesite el usuario o no.

Inicialmente, el proyecto se comenzó a realizar en Eclipse porque se tuvieron algunos problemas con el framework Android Studio, pero posteriormente se exportó el proyecto a este IDE y se continuó trabajando con él.

2.2.2 Android Studio

Android Studio [14] es el IDE oficial, basado en IntelliJ IDEA, para desarrollar aplicaciones Android. Además, permite escribir código de forma eficiente puesto que cuenta con la posibilidad de realizar algunas refactorizaciones. Compila el código escrito y permite generar una APK (Application Package File) a partir de él. Cuenta con la posibilidad de simular la ejecución del código en un dispositivo Android, a partir de emuladores.

Como se ha dicho anteriormente, aunque el proyecto se inicia con Eclipse, después se pasa a Android Studio.

2.2.3 MagicDraw UML

MagicDraw UML [15] es una herramienta CASE compatible con el estándar UML 2.3, desarrollo de código para diversos lenguajes de programación, así como para modelar datos. Es usada con el fin de aumentar la productividad, haciendo más fácil el análisis y el diseño de sistemas, puesto que permite recoger los datos en diagramas con el estándar UML.

2.2.4 Codeigniter

Codeigniter es un framework PHP para la creación rápida de aplicaciones web. Es de código libre y contiene una serie de librerías que sirven para el desarrollo de aplicaciones web además de proponer una manera de desarrollarlas. Es versátil, comparada con otros frameworks PHP, fácil de instalar, flexible y ligera. Además, es mucho más rápido que otros entornos de características similares [16] [17].

En este proyecto ha sido necesario trabajar con Codeigniter porque ya estaba implantado en el sistema de la empresa.

2.2.5 SonarQube

SonarQube [18] es una plataforma de código abierto dedicada al análisis y medida de la calidad de código fuente. Cubre siete ramas de la calidad de código: duplicaciones, test unitarios, complejidad, bugs potenciales, reglas de código, comentarios y arquitectura y diseño. Cuenta con *plug-ins* para ampliar sus funcionalidades y puede trabajar con más de 20 lenguajes de programación distintos.

Uno de los requisitos no funcionales de este proyecto es conseguir que, una vez finalizada la aplicación, esta sea un software de calidad y fácilmente mantenible, por lo que finalizada la implementación se utilizó esta herramienta con el fin de ver los puntos débiles y tratar de mejorarlos, teniendo un seguimiento y control de todo el proceso de mejora.

2.2.6 BitBucket

BitBucket es un Sistema de Control de Versiones, por lo que permite registrar los cambios realizados sobre uno o varios archivos a lo largo del tiempo, permitiendo tener un registro de todas las versiones, por si fuera necesario recuperar alguna en el futuro. BitBucket [19] permite identificarnos con un OpenID, es decir, nuestra cuenta de Facebook, Google, Twitter o GitHub, o simplemente con nuestro correo. Cuenta con la posibilidad de tener varios repositorios privados, crear grupos de trabajo y realizar cambios sobre HTTPS sin tener que configurar ningún puerto.

2.2.7 SourceTree

SourceTree [20] [21] es un cliente de Git y Mercurial para Windows o Mac, que permite sustituir la línea de comandos por una interfaz sencilla e intuitiva que permite gestionar todos los repositorios. Pese a volver más sencillo el manejo de un DVCS (*Distributed Version Control System* o Sistema de Control de Versiones), es una herramienta potente y flexible. Permite crear y clonar repositorios, realizar cambios en los archivos, detectar y resolver conflictos o consultar el historial de cambios de un repositorio, entre otras funcionalidades.

En este proyecto, SourceTree es usado como complemento de BitBucket para tener un control de versiones a la hora de desarrollar el mismo.

2.2.8 Project 2013

Microsoft Project [22] es un software de administración de proyectos desarrollado y comercializado por Microsoft, destinado a administradores de proyectos, con el fin de facilitar la labor en la gestión de los proyectos. Mediante esta herramienta se puede definir tareas, asignar recursos a tareas y dar un seguimiento al progreso, entre otras posibilidades.

En este proyecto se hace uso de la versión 2013 para realizar la planificación y seguimiento del proyecto.

2.2.9 Robotium

Robotium [23] [24] es un framework usado para la automatización de test de caja negra en Android. Está pensado para probar las interfaces de una aplicación y su funcionamiento. En este

proyecto se ha elegido este framework por ser uno de los más usados, ser fácil y no requerir un gran aprendizaje. Será usado para todas las pruebas que se realicen en las interfaces.

2.2.10 Mockito

Mockito [25] es un framework usado a la hora de implementar las pruebas de una aplicación, puesto que permite crear objetos *mock* de los módulos que se desee, entendiendo que un objeto *mock* es un objeto simulado, es decir, un objeto que imita el comportamiento de un módulo, pero de forma controlada.

2.2.11 JUnit

JUnit [26] es un conjunto de bibliotecas usadas para la implementación de pruebas de aplicaciones Java, las cuales permiten controlar y evaluar el funcionamiento de los métodos.

3. Especificación de requisitos

En esta sección se recoge el conjunto de requisitos funcionales y no funcionales de la aplicación a desarrollar.

3.1 Captura de requisitos

La fase de especificación de requisitos de una aplicación empieza por la captura de sus requisitos, para lo cual, en este caso, se ha realizado mediante entrevistas y reuniones con los dos clientes de la aplicación. Asimismo, se llevaron a cabo sesiones de *brainstorming* para obtener más requisitos y refinar los existentes. Además, al seguirse una metodología ágil, ha sido posible que las nuevas ideas que han ido surgiendo se hayan incluido como nuevos requisitos.

La lista definitiva y consensuada de requisitos, funcionales y no funcionales que esta aplicación debe cumplir se detallan a continuación.

3.2 Requisitos funcionales

Los requisitos funcionales establecen el comportamiento del sistema, definiendo una función de este o sus componentes.

En la tabla 3.1, se recogen todos los requisitos funcionales que la aplicación debe cumplir. Estos se identifican con un código único que se acompaña con una breve descripción:

Tabla 3.1. Requisitos funcionales

Identificador	Descripción
RF01	La aplicación deberá poder traducirse a otros idiomas
RF02	La aplicación almacenará datos del consumo de los últimos 12 meses, en cuanto a internet, llamadas, SMS recibidos y enviados.
RF03	La aplicación contará con una gráfica para poder ver, en conjunto (SMS, MMS, Internet, llamadas) o sólo un tipo de ellos, los datos de los últimos 12 meses
RF04	La aplicación permitirá enviar un email o SMS, dependiendo de si el dispositivo cuenta con Wi-Fi o no, a uno de los autorizados de su empresa
RF05	La aplicación permitirá llamar a uno de los autorizados
RF06	La aplicación contará con un listado de los autorizados con los que puede contactar de la empresa
RF07	La aplicación permitirá elegir el país de origen del usuario
RF08	La aplicación debe poder recibir mensajes *Push*
RF09	El usuario podrá modificar su contraseña de acceso desde la aplicación
RF10	El usuario deberá estar autorizado en la BD de la empresa para usar la aplicación

RF11	El usuario autorizado deberá iniciar sesión con su email y contraseña la primera vez para poder usar la aplicación
RF12	La aplicación comprobará 2 veces por minuto la situación geográfica del Smartphone y el estado del *roaming*
RF13	La aplicación comprobará cada 8h si hay que activar o desactivar el *roaming*
RF14	La aplicación mostrará notificaciones en casos de urgencia al detectar que es necesario activar o desactivar el *roaming*
RF15	El usuario podrá programar la activación o desactivación del *roaming* con antelación a través de la aplicación
RF16	El usuario podrá configurar la aplicación para que la activación y desactivación del *roaming*, así como el envío del email, sean automáticos en cuanto detecte el cambio de país
RF17	El usuario podrá configurar la aplicación para que se desactive el *roaming* automáticamente si está en España y son las 23.30h
RF18	La aplicación contará con la opción de enviar un email a la compañía de telefonía (pasando por el servidor) para enviar la fecha de entrada o salida en España y, por tanto, la de activación o desactivación del *roaming*. De esta forma, el responsable de la empresa no tendrá que encargarse de estas tareas
RF19	La aplicación contará con un log
RF20	Además del log propio de la aplicación, esta volcará datos en un log del servidor de la empresa conMymo, que almacenará los datos del log de la aplicación
RF21	La aplicación contará con un menú lateral para acceder a las distintas funcionalidades
RF22	En ciertos casos, la aplicación mostrará una notificación con un botón para llamar a un contacto (Estos casos son, por ejemplo, si ya ha salido al extranjero sin activar el bono, si ha vuelto a su país sin desactivarlo,…)
RF23	La aplicación contará con la opción de activar/desactivar el bono inmediatamente o con unas horas de antelación
RF24	La aplicación permitirá programar la activación y/o desactivación del bono con meses de antelación

3.3 Requisitos no funcionales

Los requisitos no funcionales, por su parte, imponen restricciones en el diseño e implementación de la aplicación. Entre otros, hacen referencia al rendimiento, disponibilidad, operatividad, escalabilidad, interfaz y seguridad.

Los requisitos no funcionales de nuestro proyecto se muestran a continuación, en la tabla 3.2.

Tabla 3.2. Requisitos no funcionales

Identificador	Descripción	Tipo
RNF01	La aplicación debe ser programada para Android nativo	Portabilidad
RFN02	Las interfaces deben poder visualizarse sólo en modo vertical	Usabilidad
RFN03	Las actualizaciones se realizarán mediante Google Play Store	Mantenibilidad
RFN04	La BD de la aplicación residirá en SQLite	Operatividad
RFN05	Las interfaces deben seguir un diseño similar entre ellas	Interfaz
RFN06	La aplicación debe ser fácil e intuitiva	Interfaz
RFN07	La contraseña debe almacenarse de forma segura	Seguridad
RFN08	Se mejorará la calidad y mantenibilidad del producto final	Mantenibilidad

3.4 Casos de uso

Los casos de uso son diagramas que describen la funcionalidad de un sistema a desarrollar, representando la forma en la que el cliente o clientes, denominados actores, operan con el sistema en desarrollo.

A continuación, en la Figura 3.1, se incluye el diagrama de casos de uso principal de la aplicación.

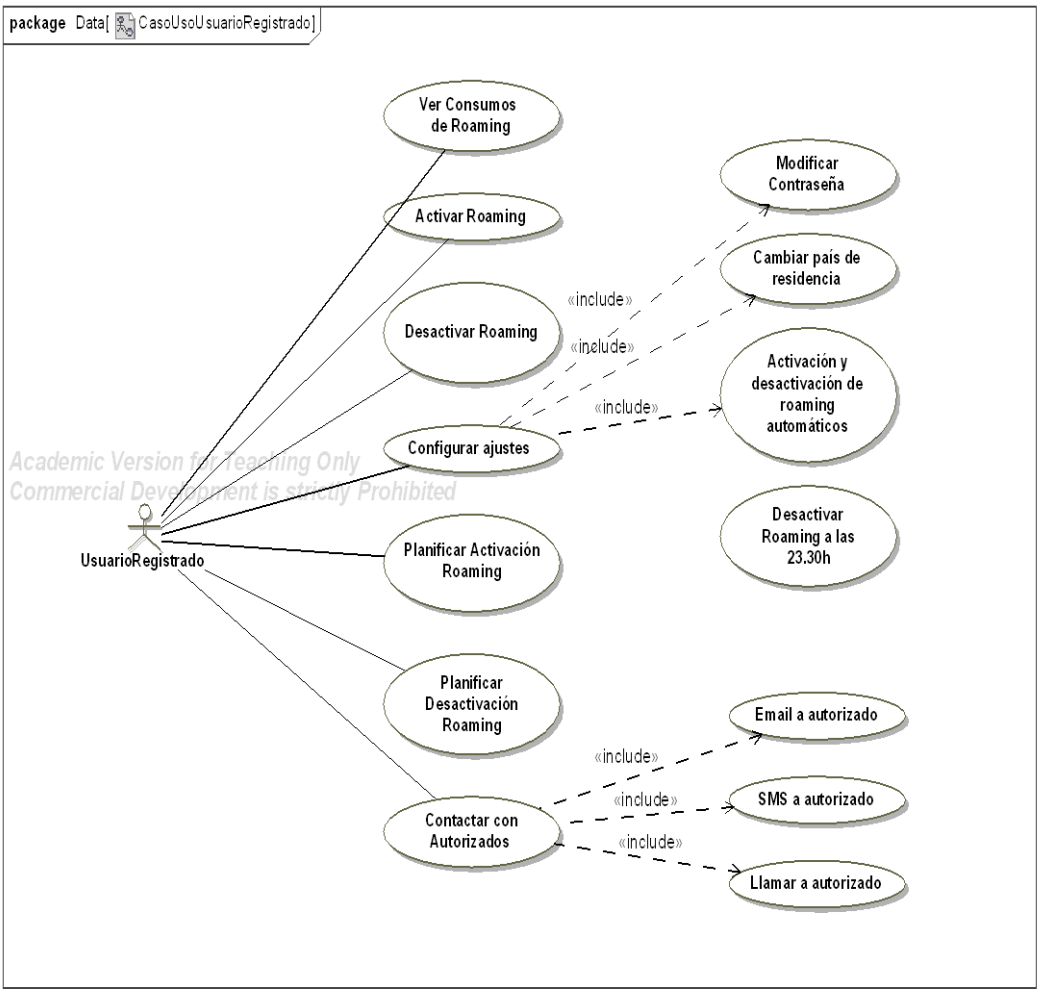


Figura 3.1. Casos de uso principales

4. Diseño de la aplicación

Previamente a la implementación de un sistema, se ha de realizar el diseño que ha de seguir, buscando que se cumplan todos los requisitos con él.

En este apartado se explicará el diseño arquitectónico, el cuál mostrará la arquitectura software que tendrá el sistema, y el sistema detallado, en el cuál se explicarán los componentes de la aplicación.

4.1 Diseño arquitectónico

Diseñar la arquitectura de un sistema software es una tarea cuya finalidad es conseguir una arquitectura que permita cumplir los requisitos establecidos, pero permitiendo cambios posteriores con poco esfuerzo y reutilización a gran escala.

En este proyecto, se ha seguido una arquitectura en tres capas, ver la Figura 4.1, en la que se distinguen las siguientes:

- Presentación: es la que permite la interacción con el usuario, lo que éste ve. Está formada por las interfaces de la aplicación para *Smartphone*.
- Lógica de negocio: realiza las funcionalidades y procesos necesarios, además de la comunicación con los datos. Se encontrará en las clases Java de la aplicación, aunque algunas pequeñas funcionalidades pueden estar contenidas en librerías del servidor de la empresa.
- De persistencia o datos: encargada de almacenar y manipular todos los datos con los que tenga que trabajar la aplicación. Habrá dos bases de datos con las que debe operar la aplicación, la propia donde se recoge su consumo y detalles de personalización y la de la empresa, que almacena la gestión de bonos y usuarios.

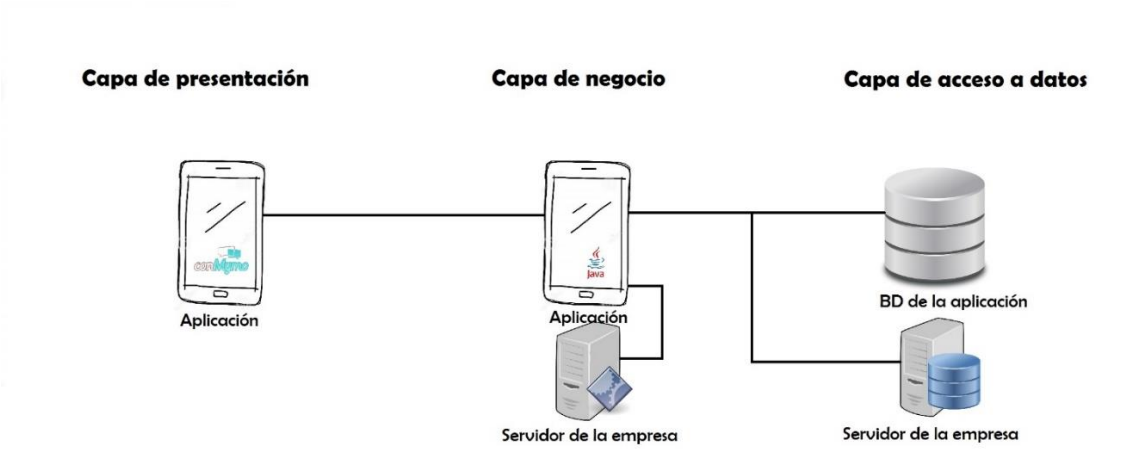


Figura 4.1. Arquitectura del sistema

4.2 Diseño detallado

Una vez se ha elegido la arquitectura, se pasa al diseño detallado, que es la última tarea de descomposición orientada a objetos, y en la cual se llega a las unidades de programación.

A continuación, en la Figura 4.2, vemos el diagrama de despliegue correspondiente a nuestro proyecto.

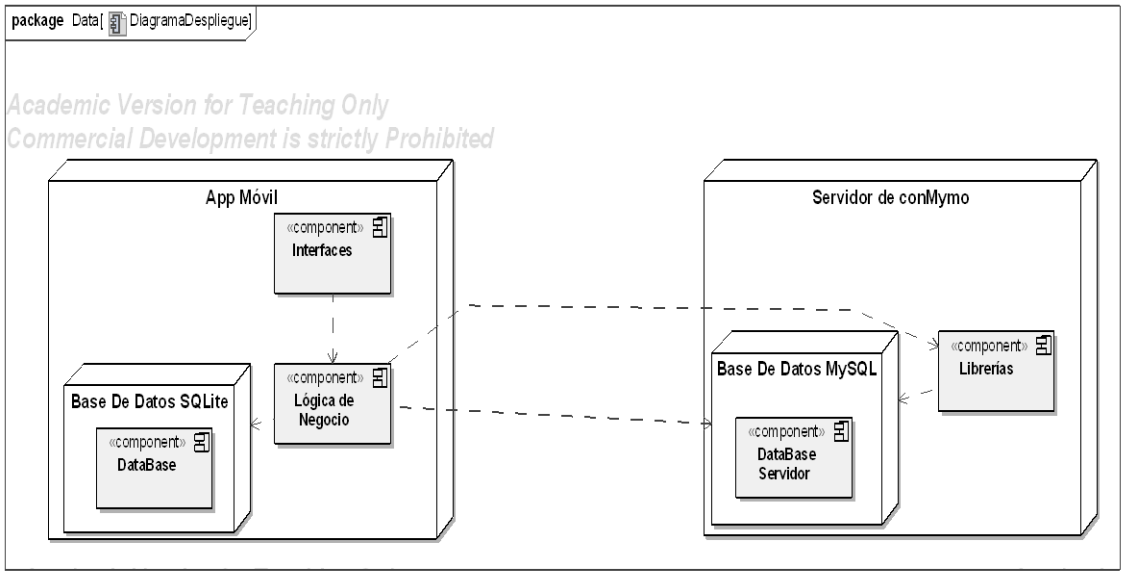


Figura 4.2. Diagrama de despliegue de la aplicación

En el diagrama de despliegue se pueden diferenciar dos módulos importantes. Por un lado, está el servidor de conMymo, la empresa, en el cuál se aloja una base de datos que usa la tecnología MySQL y un conjunto de librerías. Este será único, es decir, sólo habrá un servidor. La base de datos almacena datos necesarios para la aplicación, como puede ser el listado de usuarios que están autorizados para usarla (sólo podrán aquellos a los que sus empresas se lo permitan) o los bonos disponibles en función del país de destino, entre otras cosas. Además de leer datos de la base de datos del servidor, la aplicación se encargará también de almacenar y modificar datos en ella, por ejemplo, indicando en cada momento el estado del *Smartphone*, es decir, si está en el extranjero con el bono activado, si está en el país de origen de vuelta pero aún con el bono activado, etc. Por último, en este módulo, se tienen una serie de librerías ya implementadas, las cuales serán ligeramente modificadas o extendidas para el uso de esta aplicación. Estas librerías son usadas por la aplicación móvil para algunas de sus funcionalidades.

En cuanto al otro módulo, la aplicación móvil, cuenta con una base de datos alojada en el dispositivo, las interfaces que conformarán la capa de presentación y la lógica de negocio, encargada de comunicarse con las librerías y la base de datos del servidor cuando sea necesario. El módulo de la aplicación móvil será desplegado en cada *Smartphone* que use la aplicación.

La base de datos de la aplicación móvil es bastante simple. Su diseño se muestra en la Figura 4.3.

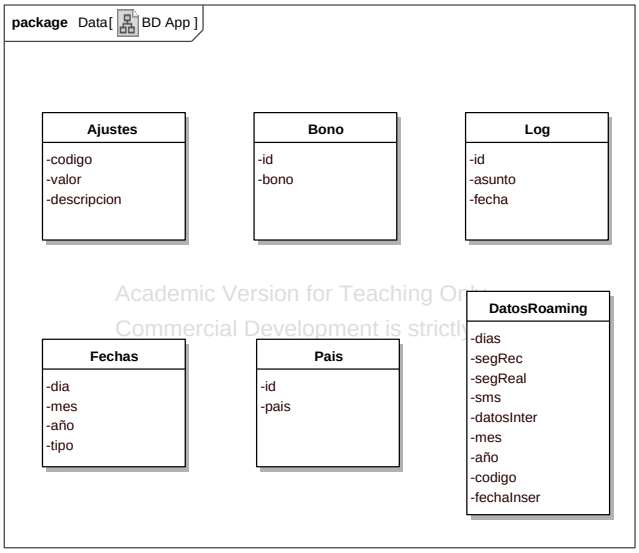


Figura 4.3. Diseño de la base de datos de la aplicación

Como se puede observar, la base de datos se encarga de almacenar la configuración de los ajustes que ha introducido el usuario, el bono activado en ese momento, las fechas de activación y desactivación del bono correspondientes al próximo viaje que va a realizarse al extranjero, el país de origen y los datos de consumo estando en *roaming* de los últimos doce meses. Además, cuenta con un pequeño log que almacena una entrada por cada cambio relevante en la aplicación, como puede ser que se programe un viaje, que se active un bono, se modifique un ajuste entre otros.

En cuanto a la base de datos del servidor, no serán necesarios muchos cambios. Simplemente, almacenar para cada usuario la empresa a la que pertenece y si puede o no usar la aplicación, puesto que este programa será contratado por las empresas y serán ellas quienes elijan cuáles de sus empleados pueden usarlo y cuáles no. También será necesario crear tablas que almacenen todos los países y la relación país-bono, es decir, qué bono hay que contratar en función de a qué país se viaja.

5. Implementación

Una vez se han recogido los requisitos y se ha elegido una arquitectura que los satisfaga, se procede a la implementación de todos los módulos necesarios.

Como ya se ha comentado anteriormente, este proyecto ha sido desarrollado íntegramente por mí por lo que he ido realizando las tareas de programación una a continuación de otra. Entre ellas se encuentran:

- Implementación inicial de las interfaces y del menú lateral.
- Implementación de los tipos de datos usados.
- Creación de la base de datos de la app y modificación de la base de datos del servidor de la empresa.
- Implementación de métodos auxiliares para trabajar con la base de datos.
- Implementación de la capa de negocio.
- Implementación del *Service*, una clase que contiene el código que se ejecuta en *background.*
- Implementación de las notificaciones.
- Creación del log, tanto en la app como en el servidor.
- Mejoras en el diseño de las interfaces (colores, bordes, …).
- Añadir ventanas emergentes cuando hay errores (por ejemplo, si ponemos fechas incorrectas) o para confirmar que los datos introducidos son válidos.
- Implementación de las clases necesarias para que la app pueda recibir y procesar mensajes *Push.*
- Implementación del *login.*
- Añadir una interfaz que permite consultar el listado de los autorizados y llamarlos o enviarlos un SMS o correo electrónico desde la propia app, sin necesidad de salir de ella.
- Añadir la funcionalidad de que se pueda modificar la contraseña de acceso desde la propia aplicación.
- Refactorizaciones y cambios para mejorar la calidad y mantenibilidad del producto final, (ver sección 5.5).

5.1 Capa de persistencia

La aplicación de este proyecto requiere, para su correcto funcionamiento, de dos bases de datos. Por un lado, se tiene la base de datos del servidor de la empresa, implementada en MySQL. Para conectarse a ella sigue un modelo vista controlador. De ella recibe datos, por ejemplo, a la hora de autentificarse, puesto que en el servidor se tienen almacenados los clientes y sus datos, además de si pueden o no usar la aplicación, ya que ha de ser la empresa a la que pertenezcan la que elija esto. También lee datos de esta base de datos a la hora de elegir bonos de *roaming* en función del país al que se viaja. Por último, también cabe destacar que la app también introduce datos en esta base de datos; un ejemplo es cuando cambia de “estado”, entendiendo por estado la combinación de datos que explican tanto si se encuentra en el extranjero o no y si tiene un bono activado o no. De igual forma, almacena en una tabla que hace las funciones de Log los cambios más relevantes (activación/desactivación de un bono, programación de la activación/desactivación de un bono para el futuro o modificaciones en los ajustes de la app), puesto que puede darse el caso de que un usuario haga un mal funcionamiento de la app o de

su Smartphone y luego haya que revisar exactamente qué hizo. Periódicamente esta tabla borra sus registros más antiguos para no llegar a tener un tamaño excesivo. Esto es configurable pero externo a este proyecto, puesto que es la empresa la que se encarga de la base de datos del servidor.

Por otra parte, se tiene la base de datos de la propia app, gestionada por SQLite. Esta ya se explicó y detalló en el apartado 4.2 Diseño Detallado haciendo uso de la Figura 4.3. No almacena muchos datos, puesto que sólo recoge los datos que necesariamente han de permanecer en el dispositivo, aunque éste cierre la aplicación o se apague.

Una vez desarrollada la BD en SQLite y extendida la base de datos del servidor, se crearon en el primer caso, y se reutilizaron en el segundo caso, las clases para la conexión con ellas y la manipulación de sus datos. La empresa contaba ya con unas librerías para conectarse a la base de datos de su servidor y enviarse datos de tipo JSon, las cuales usaban un patrón modelo vista controlador, por lo que fue necesario familiarizarse con dichas librerías, añadirlas nuevas funcionalidades y luego hacer el casting de JSon a los tipos de archivos que deseamos leer de la base de datos del servidor, como pueden ser datos de un usuario o información de los bonos. En cuanto al módulo que permite comunicarse con la base de datos de la propia app, contiene métodos CRUD, con el fin de no repetir código a lo largo de la app

5.2 Capa de negocio

La implementación de la capa de negocio ha sido desarrollada en Android, cuya estructura de paquetes se puede observar en la Figura 5.1 (los paquetes bd, interfaces y menulateral no forman parte de esta capa).

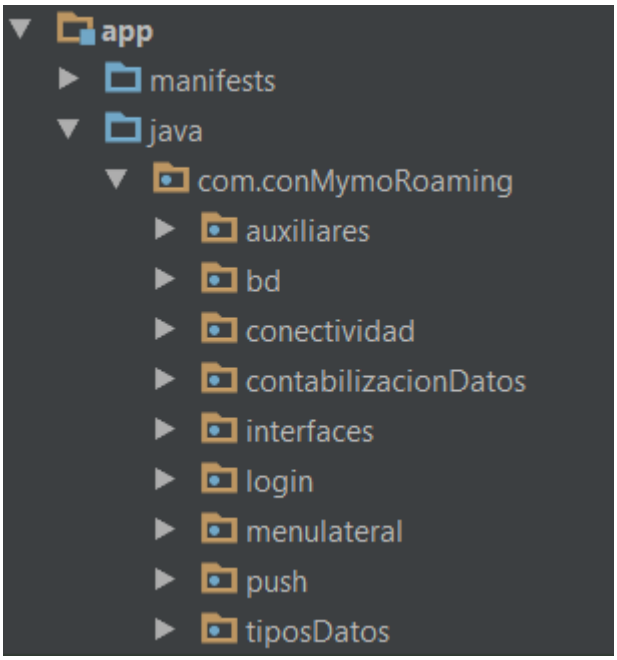


Figura 5.1. Estructura de paquetes

Dentro de la implementación, se pueden distinguir varias partes:

- La implementación de las principales funcionalidades de la app, que son las de activar y desactivar el roaming, enviar información al servidor para que se active o desactive el

bono de roaming, planificar viajes, poder realizar ajustes para personalizar la app al gusto del cliente y ver el consumo de roaming.

- Service, encargado de hacer comprobaciones periódicas en background. Por un lado, comprueba dos veces por minuto si el Smartphone se encuentra o no en su país de origen y, en función de ello, mostrará o no notificaciones en la pantalla o activará/desactivará el roaming y el bono si el usuario ha seleccionado que se haga automáticamente. También es el encargado de activar/desactivar el bono y el roaming cuando se haya planificado un viaje y llegue el momento. Además, por seguridad, cada 8h revisa si ha habido algún cambio en el servidor que no se haya detectado en el Smartphone, debido a algún fallo técnico, como puede ser que ya se tenga el bono activado.
- Ventanas emergentes, creadas con el fin de hacer más segura la app ante fallos humanos. Permiten mostrar avisos al usuario si ha introducido datos incorrectos o una vez ha introducido datos y ha dado a aceptar, como forma de asegurar que el usuario desea introducirlos.
- Añadir entradas al log, tanto en el dispositivo móvil como en el servidor, con el fin de registrar los cambios más relevantes.
- Con el fin de que la aplicación pueda recibir mensajes *Push* del servidor, ha sido necesario implementar unas clases que lo permitan. Por un lado, al inicializar la app por primera vez, esta debe inicializar el servicio GCM o *Google Cloud Messaging*, y por otro, necesitamos otra clase que se encargue de recuperar los datos que se reciben. Sin embargo, el servicio *Push* aún no funciona ni ha sido probado, puesto que es necesario que también se implemente en el servidor la parte que envía el mensaje y de eso se encargará la propia empresa.
- La implementación del login tiene la finalidad de permitir que sólo los usuarios que hayan contratado el servicio que ofrece esta app puedan usarla. Los datos que solicita son el CIF de la empresa del trabajador, su correo y contraseña. Además, cuenta con la opción de que se le envíe un email con la contraseña si la ha olvidado, siempre y cuando introduzca el email del trabajo.
- Para complementar el login y como un Smartphone con SO Android cuenta con la opción de ver las cuentas en ajustes, se ha conseguido que la cuenta de esta app aparezca ahí. Por ahora no hay mucha información en ella que mostrar, pero en un futuro se podría modificar las opciones disponibles a través de este acceso. Esto se puede ver en las Figuras 5.2 y 5.3.

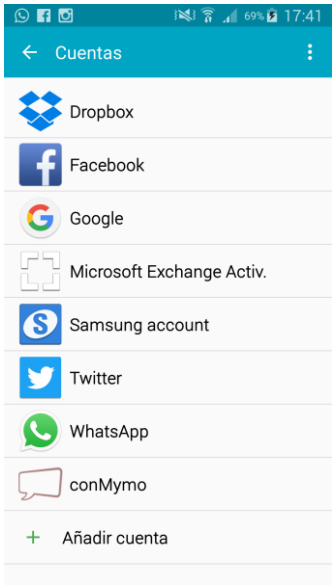


Figura 5.2. Cuentas de un dispositivo Android

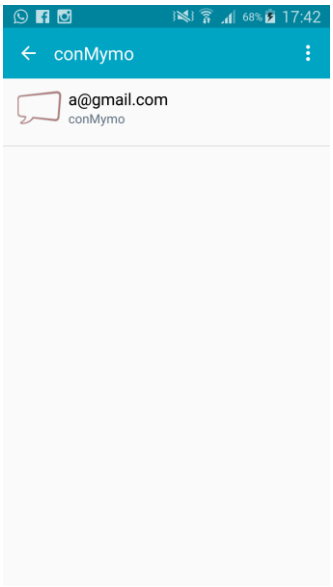


Figura 5.3. Detalle de la cuenta de la app

- La implementación de nuevas funcionalidades se debió a que surgieron nuevos requisitos y el tiempo permitió su implementación. En primer lugar, se consideró útil tener un listado de los autorizados de la empresa para contactar rápidamente con ellos en caso de que el trabajador tuviera algún problema cuando viajara por temas de trabajo. Se elaboró una lista, en la que se puede ver el nombre del autorizado, y si se selecciona en un desplegable en él, se verá su teléfono y email. Además, esta interfaz cuenta con dos botones, uno para llamar al autorizado, y otro para contactar con él de forma escrita. Si seleccionamos la escrita, se mostrará una pantalla en la que el usuario introducirá el mensaje que desea enviarle. En función de si el móvil está con Wi-fi o no, enviará un correo electrónico o SMS, con el fin de minimizar gastos siempre que sea posible.

- También se añadió la opción de modificar la contraseña de acceso desde la propia app por si el usuario lo considera conveniente.
- Con el fin de que la app sea usada desde otros países también, se modificó para que no tenga en cuenta por defecto que el país de origen es España, sino que se pueda configurar. De esta forma, trabajadores de otros países podrán usarla.
- Por último, otra funcionalidad añadida fue, usando la librería Android Plot, la de mostrar el consumo en roaming de los últimos meses en una gráfica, con el fin de ver la evolución que sigue el usuario. Tiene la opción de ver en la gráfica las líneas correspondientes a todos los datos recogidos (megas de internet, mensajes y minutos de llamadas) o visualizar exclusivamente el dato que él desee.

5.3 Capa de presentación

Como se ha dicho anteriormente, no había ningún requisito que especificase la apariencia de las interfaces, por lo que se ha desarrollado libremente. Se ha tratado conseguir unas interfaces limpias, con pocos elementos y colores, con el fin de no saturar visualmente al usuario y obtener una app fácil e intuitiva.

A la hora de desarrollar la capa de presentación, se han de tener en cuenta varios aspectos:

- Los colores y tipos de letra: los colores elegidos han sido similares a los de la página web de la empresa, con el fin de crear la sensación de "marca". El tipo de letra escogido es el que viene por defecto en Android, puesto que es fácil de leer. Los tamaños son similares, aunque los títulos son ligeramente más grandes. Se ha tratado de no usar ni tamaños muy pequeños, que dificulten la lectura, ni muy grandes, puesto que pueden llegar a influir negativamente en la percepción del usuario.
- El menú de opciones: se ha elegido un menú lateral ya que permite un buen aprovechamiento del espacio y es accesible desde cualquier pantalla que se desee.
- Iconos: con el fin de hacer más fácil de usar la app, se han añadido iconos junto a algunas funciones. Además, el logo de la empresa se ha añadido en el menú lateral (de forma decorativa), en el botón de la barra superior para acceder al menú lateral y en el icono de la app para que, junto con los colores elegidos, se dé sensación de marca comercial.
- Tiempo de aprendizaje: como el conjunto de usuarios que usarán la app serán muy variados en cuanto a conocimientos tecnológicos, es importante que el tiempo de aprendizaje sea bajo. Este tiempo se intenta minimizar gracias al uso de iconos y de interfaces intuitivas y fáciles.
- Recuerdo en el tiempo: los usuarios de esta aplicación, dependiendo de la empresa para la que trabajen y el puesto que tengan, necesitarán usar muy a menudo la aplicación o no muy frecuentemente. Esto irá en función de la cantidad de viajes al extranjero que se realicen. Suponiendo que no se usa la aplicación frecuentemente, será vital que los usuarios intermitentes sean capaces de usar el sistema sin tener que aprender cómo funciona partiendo de cero cada vez. Para favorecer esto, se ha tratado de que el menú de opciones no tenga demasiadas opciones y que la aplicación sea fácil e intuitiva.

En cuanto a la implementación de la capa de presentación, se han usado ficheros XML, *Activities* y *Fragments*. En primer lugar, la apariencia de las interfaces y los elementos que aparecen en cada una han sido implementados en ficheros XML. Por otro lado, tenemos las *Activities* y

Fragments, que son componentes en código Java que implementan las interfaces descritas a través de XML. En nuestro caso, se han usado más *Fragments* que *Activities*. Esto es debido a que un *Fragment* requiere de una *Activity* para existir, pero puede haber varios *Fragments* por *Activity*, lo que da la oportunidad de que los elementos de la pantalla vayan apareciendo poco a poco e independientes unos de otros. Esto permite, por ejemplo, una interfaz muy limpia y que se vaya llenando de elementos en función de lo que el usuario desee realizar.

A continuación, se muestran las distintas pantallas que se han programado en la aplicación.

Cuando se accede por primera vez a la aplicación, se muestra un formulario, ver Figura 5.4, en la que habrá que introducir el CIF de la empresa, el correo y la contraseña.

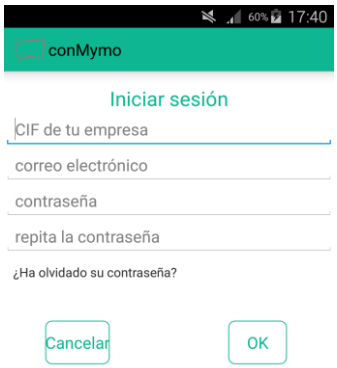


Figura 5.4. Formulario de inicio de sesión

En caso de que no haya conexión a Internet, la aplicación no permitirá el inicio de sesión puesto que no se podrán verificar los datos. Si es así, se mostrará una notificación, ver Figura 5.5. De igual forma, se mostrará también una notificación si faltan datos por rellenar o son incorrectos.

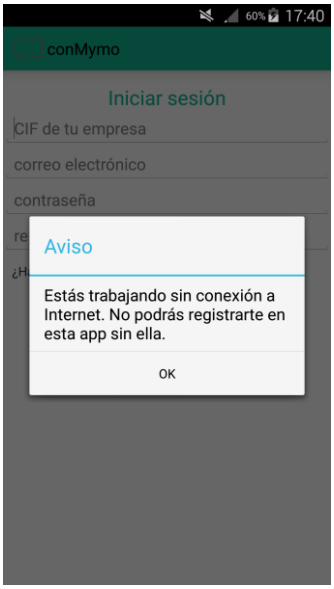


Figura 5.5. Notificación por fallo de conexión

La pantalla de inicio de sesión cuenta también con la opción de acceder a otro formulario en caso de que se haya olvidado la contraseña (ver Figura 5.6). Esta permite recuperar la contraseña a través del email.

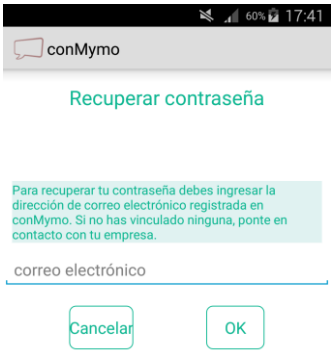


Figura 5.6. Recuperar contraseña

Una vez se han introducido correctamente los datos y se cuenta con conexión a Internet, la aplicación muestra su pantalla inicial, con el resumen de datos de consumo en *roaming* el mes actual y una gráfica para mostrar la evolución de estos consumos en los últimos 12 meses. Como se puede observar en la Figura 5.7, en este caso no hay consumos de *roaming* este mes, pero sí ha habido una evolución en los meses anteriores.

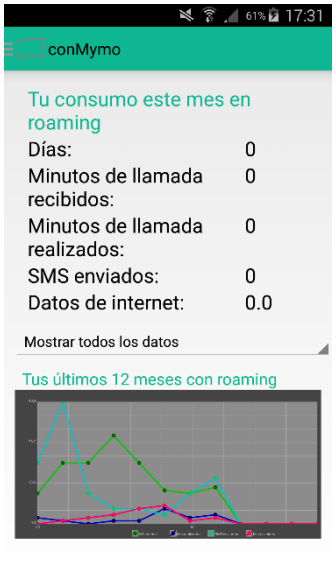


Figura 5.7. Pantalla principal

Desde esta pantalla se puede, a través del desplegable, seleccionar un tipo de datos (minutos de llamadas recibidos o realizados, SMS o datos de internet) para verlo aislado en la gráfica o todos ellos en conjunto. Se muestra esta funcionalidad en las Figuras 5.8 y 5.9.

Figura 5.8. Desplegable de la pantalla principal

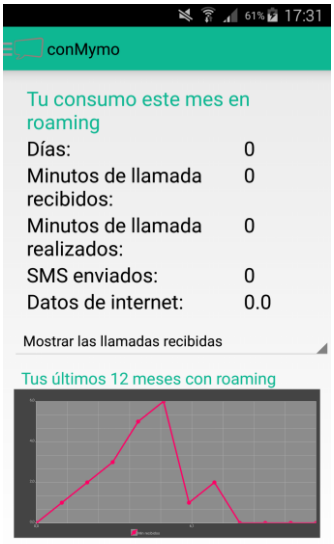


Figura 5.9. Pantalla principal con la gráfica para las llamadas recibidas

Tanto desde la pantalla principal como desde cualquier otra, una vez se ha iniciado una sesión, se puede acceder al menú lateral, el cual permite navegar entre las distintas pantallas. Puede accederse a él o bien deslizando el dedo por la pantalla de izquierda a derecha o mediante el botón superior que aparece en la barra verde. El menú aparece en la Figura 5.10.

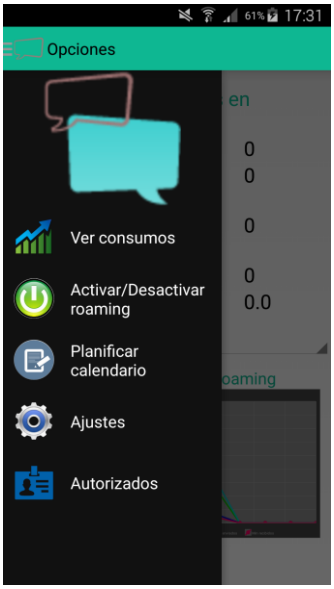


Figura 5.10. Menú lateral

Para acceder a la pantalla principal, se pulsaría la opción Ver consumos. En caso de seleccionar Activar/Desactivar roaming, aparecerá una pantalla con un botón que permite activar o desactivar el roaming. Cuenta con la opción de hacerlo inmediatamente o posponerlo unas horas. Si se trata de activar o desactivar el roaming sin haber introducido el país al que se viaja, aparecerá una notificación avisando de que es necesario introducirlo. De igual modo, para más seguridad, se muestra una notificación de confirmación si el usuario introduce el país y da al botón ok. Las Figuras 5.11, 5.12, 5.13, 5.14, 5.15 y 5.16 muestran gráficamente toda esta funcionalidad.

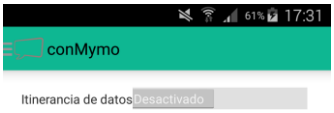


Figura 5.11. Itinerancia apagada

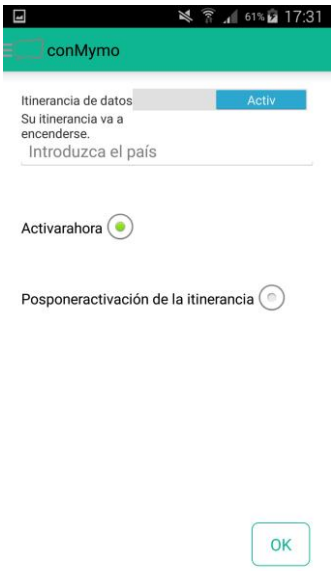


Figura 5.12. Activación de la itinerancia

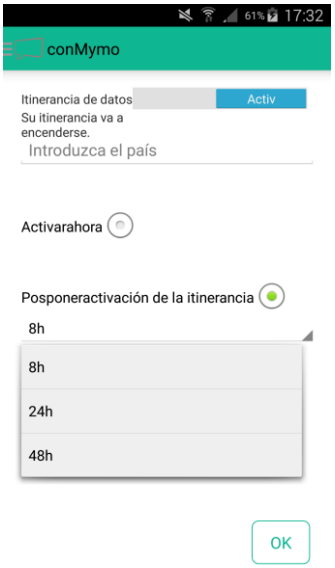


Figura 5.13. Posponer activación de la itinerancia

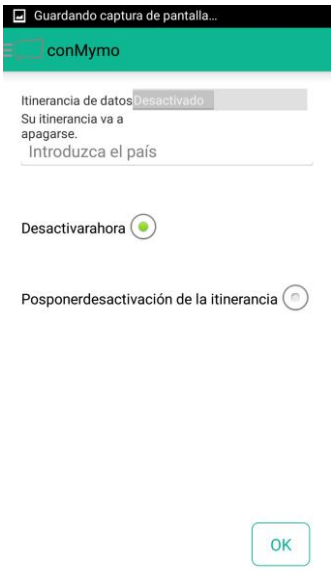


Figura 5.14. Desactivación de la itinerancia

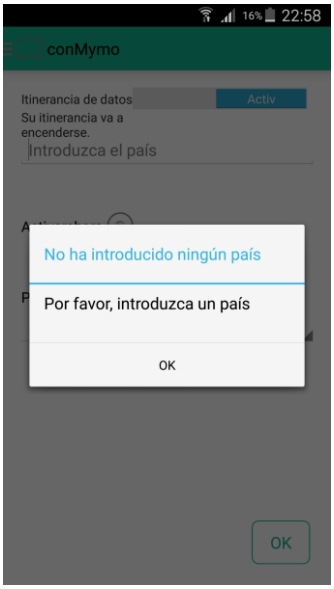


Figura 5.15. Notificación cuando falta el país

Figura 5.16. Confirmación de los valores

Otra de las funcionalidades a las que se puede acceder a través del menú lateral es la de Planificar calendario. En esta pantalla se puede configurar un viaje a un país para que se active o desactive el bono correspondiente. Se puede introducir una fecha de salida, una de entrada o ambas (ver Figura 5.17). Al igual que en la anterior funcionalidad, se muestra una notificación para confirmar los datos al pulsar el botón ok.

Figura 5.17. Planificar calendario

Los ajustes también son accesibles a través del menú lateral (ver Figura 5.18). Desde esta pantalla se puede activar o desactivar algunas opciones de la aplicación, acceder a la pantalla para cambiar la contraseña (ver Figura 5.19) o modificar el país de origen, que por defecto es España (ver Figura 5.20).

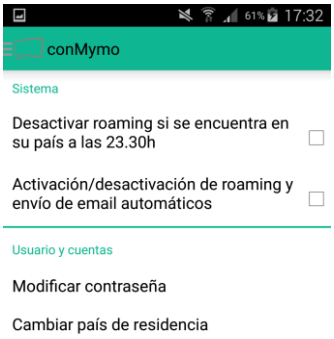


Figura 5.18. Pantalla de ajustes

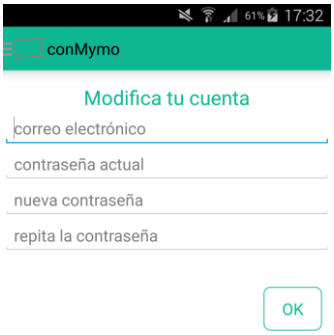


Figura 5.19. Pantalla para modificar la contraseña

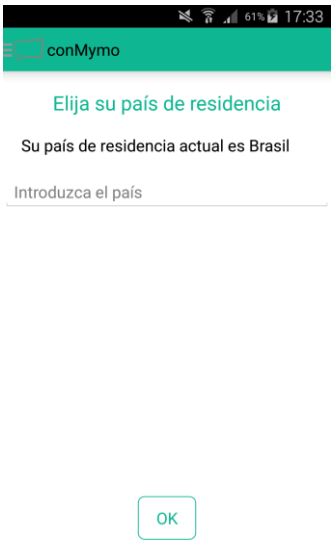


Figura 5.20. Pantalla para modificar el país de origen

La última funcionalidad a la que se puede acceder a través del menú es Autorizados, es decir, la lista de autorizados de la empresa. En caso de que, por algún fallo de conexión o caída del servidor no se pueda recuperar esta información, se mostrará un mensaje indicándolo. En caso de que sí se recuperen datos de los autorizados, se mostrarán sus nombres y, al pinchar sobre ellos, se desplegará su información de contacto. Si se pulsa uno de los botones para contactar antes de haber seleccionado un autorizado, se mostrará un aviso para que seleccionemos el nombre de alguna persona. Una vez seleccionado, se le podrá llamar mediante el botón Llamar o contactar con él por escrito, mediante el botón SMS/Email. Este último botón llevará a una nueva pantalla donde se introducirá el texto que se desee y, en función de si se cuenta con conexión Wi-fi o no, enviará un correo electrónico o un SMS. Esto se debe a que un correo

electrónico con conexión Wi-fi es gratuito, por lo que se tratará de minimizar gastos. Esta funcionalidad se muestra gráficamente en las Figuras 5.21, 5.22, 5.23, 5.24 y 5.25.

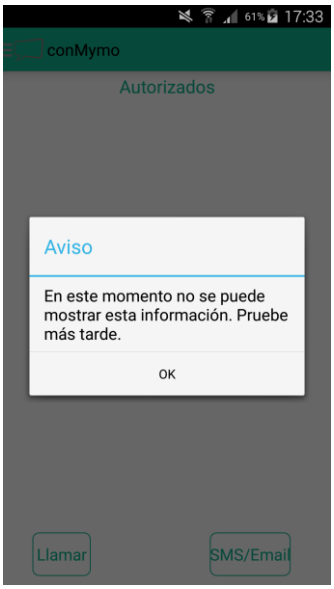


Figura 5.21. Notificación por falta de datos

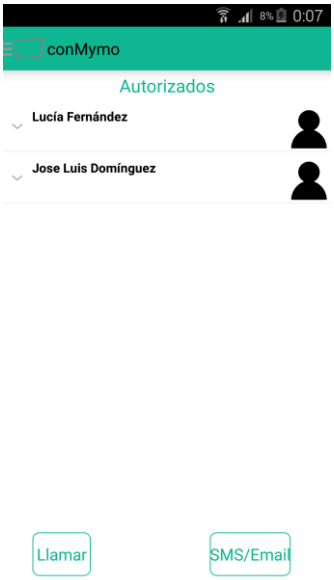


Figura 5.22. Pantalla con los autorizados

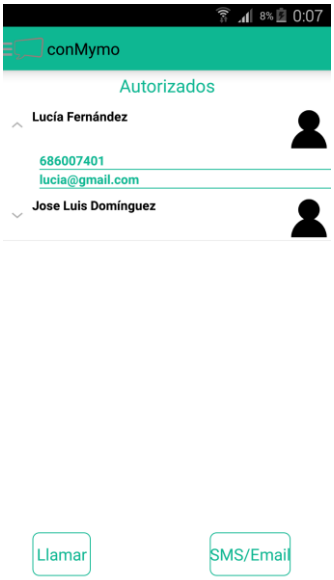


Figura 5.23. Pantalla con autorizado desplegado

Figura 5.24. Pantalla para enviar texto

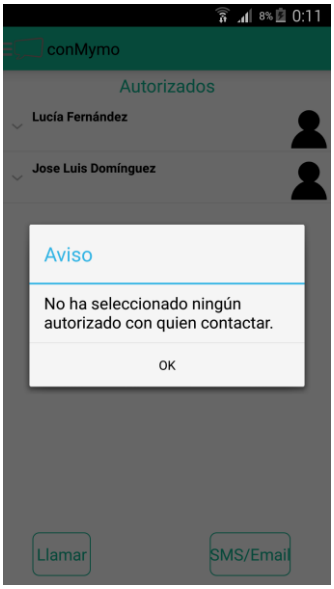


Figura 5.25. Notificación informando que falta seleccionar un autorizado

Por último, la app también muestra notificaciones en caso de que se esté en el extranjero con el bono desactivado y la itinerancia activada o si se ha regresado al país de origen con el bono activado. Estas notificaciones cuentan con un botón que permite llamar a uno de los autorizados inmediatamente. En la Figura 5.26 se muestra un ejemplo de una de estas notificaciones.

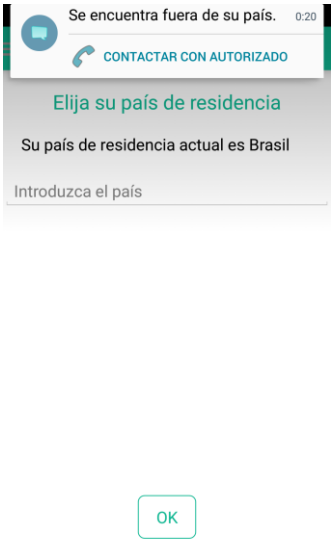


Figura 5.26. Notificación con botón de llamar

5.4 Gestión de la configuración

La gestión de la configuración es un proceso clave en cualquier organización o departamento dedicado a desarrollar software. Posibilita una mejor organización en el desarrollo y mantenimiento, permitiendo controlar los cambios que se van produciendo en el software,

tener todas las versiones anteriores listas para volver a ser usadas si es necesario revirtiendo los cambios y facilitando que varios programadores trabajen simultáneamente en la aplicación.

Con el fin de automatizar en lo posible esta labor y facilitarla, existen en el mercado varias herramientas de control de versiones. En nuestro proyecto, inicialmente se usó Apache Subversion, abreviado SVN, puesto que ya se había usado en proyectos anteriores de la empresa y estaba ya configurado. Como inicialmente la aplicación se desarrolló en Eclipse, se usó el complemento de SVN que existe para esta plataforma. Cuando el proyecto se importó a Android Studio, se decidió usar BitBucket como repositorio, y la herramienta SourceTree para trabajar con el repositorio. La razón principal del cambio fue que el desarrollo se seguiría haciendo sin estar en la empresa por lo que se requería un repositorio privado.

5.5 Medición de la calidad

Con el fin de mejorar la calidad y mantenibilidad del software, una vez se finalizada la fase de implementación, se procedió a realizar un Plan de Mejora de la Calidad.

Principalmente, se centró en disminuir la duplicación de código y las *issues* o evidencias que se han encontrado, gracias al uso de la herramienta SonarQube. Con esto se consiguió disminuir la deuda técnica y mejorar la SQALE Rating, que es una clasificación de aplicaciones en función de su calidad, siendo el mejor valor la A.

Muchas de las evidencias encontradas aparecieron en librerías externas usadas por la aplicación, por lo que tras revisar la primera versión o versión 1.0, se decidió hacer un análisis con SonarQube exclusivamente del código implementado, obteniendo una versión 2.0. Algunas de las issues se marcaron como resueltas en SonarQube, aunque no lo estuviesen, puesto que podían ignorarse. Este tipo de issues fue, por ejemplo, aquellas que marcaban unas pautas para nombrar las variables y constantes de las clases o las que obligaban a marcar un atributo como final, aunque no debía serlo (a veces, SonarQube muestra evidencias, aunque en realidad no lo son).

En cuanto a las otras que sí se han ido solucionando, han sido de diversos tipos. Por ejemplo, algunos métodos tenían parámetros o variables que no se usaban, bloques de código antiguo comentado, paréntesis que no eran necesarios, se podían crear constantes en muchas ocasiones para no repetir cadenas de texto o valores múltiples veces, entre otros.

La Figura 5.27 muestra la evolución del proyecto, recordando que la versión 1.0 era la aplicación completa, la 2.0 el código de la aplicación implementado en este proyecto evitando las librerías y la 3.0 el código implementado tras las refactorizaciones y mejoras.

	APP ITINERANCIA **1.0** 22/ABR/2016	APP ITINERANCIA **2.0** 15:21	APP ITINERANCIA **3.0** 16:51
Lines of code	7,260	5,255	3,324
Duplicated lines (%)	28.6%	3.0%	0.3%
Issues	2,672	670	0
SQALE Rating	B	A	A
Technical Debt	27d	12d	0
Technical Debt Ratio	6.0%	3.9%	0.0%
Complexity	1,036	1,036	581
Complexity /file	20.7	23.0	12.4
Complexity /function	3.7	3.7	3.1
Comments (%)	40.9%	29.0%	24.5%
Comment lines	5,031	2,150	1,080

Figura 5.27. Evolución de la calidad

Como se puede apreciar, el número de líneas de código del proyecto ha disminuido considerablemente en la última versión. Fijándose en el porcentaje de líneas duplicadas y teniendo en cuenta que muchas *issues* eran debidos a variables que no se usaban o código comentado que podían eliminarse, es entendible que esto haya ocurrido.

Gracias a las refactorizaciones también se ha conseguido disminuir, considerablemente, la complejidad del proyecto.

Por último, se debe señalar que el proyecto final ha obtenido una valoración A en la SQALE Rating, siendo esta la mejor calificación posible, como ya se había dicho anteriormente.

6. Pruebas

Es sabido que cualquier aplicación software es propensa a fallos por lo que, con el fin de minimizar el número de ellos existentes en un software, se verifican sus comportamientos y funcionamiento mediante pruebas de diferentes tipos.

Hay diferentes tipos de pruebas y diferentes metodologías a la hora de diseñarlas; en este proyecto, se diseñarán y codificarán posteriores al desarrollo del código que han de probar. Además, se llevarán a cabo pruebas de cuatro tipos, unitarias, de integración, de sistema y de aceptación, las cuales se explicarán más detalladamente a continuación.

6.1 Pruebas unitarias

Las pruebas unitarias permiten verificar el correcto funcionamiento de un módulo aislado de una aplicación, encontrando fallos potenciales.

En este proyecto se han probado las clases y métodos no triviales, usando el framework JUnit para desarrollar los test unitarios de cada módulo y Mockito en el caso de que fuese necesario emular el comportamiento de otro módulo. Cabe destacar que las interfaces y el menú lateral han sido probados en las pruebas de integración y que las clases de los paquetes *conectividad* y *contabilizacionDatos* ya habían sido probadas antes de empezar el proyecto, puesto que habían sido desarrolladas en la empresa antes de la realización de este proyecto.

6.2 Pruebas de integración

Una vez se desarrollaron las pruebas unitarias y solucionado los fallos encontrados gracias a ellas, se llevaron a cabo las de integración, las cuales prueban un conjunto de módulos combinados entre sí.

En esta aplicación las pruebas de integración se ejecutaron sobre las interfaces, que interaccionan entre ellas y con las otras capas de la aplicación, comprobando que realizaban las funciones correspondientes según el flujo de eventos diseñado, por lo que se siguió una estrategia guiada por funcionalidad. Para implementar los test se usaron los framework JUnit y Robotium.

6.3 Pruebas de sistema

Las pruebas de sistema son las que verifican el comportamiento del sistema completo, comprobando propiedades emergentes, generalmente, correspondientes a requisitos no funcionales.

Dentro de estas pruebas, se llevarán a cabo: a) pruebas de seguridad, comprobando que cualquier usuario no puede usar la app, sino sólo los que estén registrados y con permisos de uso; b) pruebas de usabilidad, mediante un test de usabilidad, que aparece en el Anexo I, llevado a cabo por 6 usuarios con el fin de que haya variedad de opiniones y, c) pruebas de instalación

para validar el funcionamiento del sistema en su entorno final, utilizando para ello varios dispositivos Android con distinta versión de SO.

Estas tres fases de pruebas fueron, en su mayoría, codificadas en el mismo proyecto. En la Figura 6.1 podemos ver la estructura de los test. A partir de ahora, las pruebas que se detallarán ya no serán automatizadas.

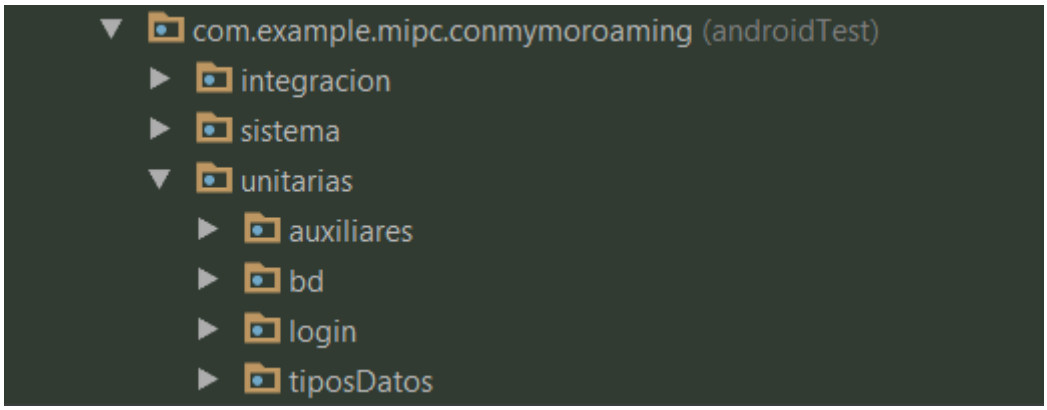


Figura 6.1. Esquema de los test automatizados

6.4 Pruebas de aceptación

Las pruebas de aceptación son las que se llevan a cabo frente al cliente, el cual verifica y comprueba si el producto cumple los requisitos especificados.

En las primeas fases de este proyecto, estas pruebas se realizaron periódicamente, puesto que se siguió una metodología ágil como ya se ha expuesto anteriormente. Las pruebas consistían en enseñar al cliente un entregable con alguna funcionalidad operando correctamente, con el fin de recibir retroalimentación por su parte. También se les mostraron entregables con las interfaces diseñadas, aunque sin la funcionalidad, para que diesen su aceptación al diseño de la aplicación.

Por último, una vez finalizado el proyecto, se programó realizar una última prueba de aceptación, con toda la aplicación completada. Para la realización de esta prueba, se concertó una cita con los responsables de la empresa, a quienes previamente se les había entregado el producto y una APK para su instalación. Esta se programó para la primera semana de junio. Tras esta reunión, los clientes dieron su aprobación a la aplicación completada (ver Anexo II).

7. Conclusiones y trabajos futuros

7.1 Conclusiones

Este proyecto tiene por objeto la creación de una aplicación móvil que ayude a las empresas a gestionar el uso de la itinerancia de datos de los Smartphones de sus trabajadores. El fin último es evitar facturas elevadas de telefonía como consecuencia de no haber activado un bono de itinerancia adecuado al país de destino.

Se debe decir que el proyecto ha cumplido con todas las especificaciones definidas e incluso, se han añadido otras durante el desarrollo del proyecto como la de mostrar notificaciones en algunos casos con un botón para contactar con el autorizado o tener la opción de ver un listado de todos los autorizados y comunicarse con ellos a través de la aplicación. Además, se ha realizado todas las fases del ciclo de vida de una aplicación software, siendo metódica y organizada, lo cual ha redundado en una aplicación funcional y fácilmente mantenible como los responsables de la empresa ConMymo indican en su carta de aceptación del producto.

Una de las mayores complicaciones para abordar este proyecto fue la de tener que trabajar con librerías ya creadas en diferentes lenguajes y adaptarse a ellas, aunque se ha logrado completar exitosamente.

En cuanto a la visión personal, este proyecto me ha servido para introducirme en el mundo de las aplicaciones Android, el cual desconocía. Al ser una aplicación compleja por usar muchas tecnologías desconocidas hasta el momento, me ha permitido tocar muchos puntos interesantes del desarrollo de aplicaciones: *Fragments* y *Activities* (componentes a partir de los cuales se crean las interfaces que luego vemos en una app), menús laterales, servicios, mostrar notificaciones… Gracias a los conocimientos de programación e Ingeniería del Software adquiridos en la Universidad, el proceso de aprendizaje no ha sido costoso y se ha producido a un ritmo adecuado. Además, debido a que no había un diseño o especificaciones de cómo debían de ser las interfaces, simplemente que se siguiese un diseño a lo largo de la aplicación, he tenido que ser yo quien crease también la apariencia de la aplicación, teniendo muy en cuenta los conocimientos adquiridos en asignaturas como Interacción Persona-Computador, con el fin de conseguir unas interfaces amigables y sencillas de utilizar.

Gracias a este proyecto he podido comprobar cómo van surgiendo nuevas necesidades en las empresas, puesto que los temas de *roaming* no tenían ninguna relevancia hace unos años, y cómo el software ayuda a solucionarlos. Personalmente, me ha demostrado que, aunque tenga que enfrentarme a la resolución de un problema en el que todo es nuevo, desde el lenguaje hasta las librerías en las que se apoya o el campo, tengo capacidad para ir aprendiendo y amoldándome a ello.

7.2 Trabajos futuros

El desarrollo de este proyecto ha conseguido cumplir sobradamente los objetivos propuestos en un inicio, además de haber incorporado mejoras que surgieron durante su desarrollo.

Sin embargo, en un futuro, se podrían incluir nuevas funcionalidades como son:

- Evitar la descarga de contenido multimedia cuando se está con *roaming* en el extranjero, a no ser que esté con Wi-Fi, para evitar gastos.
- Añadir la opción de evitar la descarga de contenido multimedia a los ajustes de la aplicación.
- Añadir una alarma que avise al usuario cuando ha excedido un límite en el consumo, estando en *roaming*.
- Añadir la opción de activar o desactivar la alarma en la sección de ajustes y permitir introducir el valor que se desee.
- Diseñar e implementar un *widget*. En este caso no está aún diseñado o planificado, pero podría incluir información sobre si tienes o no el bono de *roaming* activado, cuántos días te quedan para tu próximo viaje o cuántos te faltan para volver a casa, etc.
- Añadir en ajustes la opción de evitar las actualizaciones automáticas, para ahorrar en consumo.
- Añadir la opción en ajustes de evitar la carga de contenido multimedia al navegar por internet, para disminuir el consumo.
- Crear una interfaz web para que puedan usar los autorizados, pudiendo ver en todo momento el estado de los dispositivos de sus trabajadores.
- Mostrar la leyenda de la gráfica de forma más visible
- Diseñar un icono para la aplicación que sea más visible cuando se quiere acceder a ella desde el menú de Android.

Anexo I. Pruebas de usabilidad

AI.1 Introducción

Este anexo tiene por objeto definir una encuesta para evaluar la usabilidad de la aplicación construida y analizar las respuestas dadas.

AI.2 Encuesta

En este apartado se exponen las cuestiones que incluye la encuesta, con el fin de realizar la prueba de usabilidad a la aplicación. A continuación, se muestran las preguntas:

1. ¿Es la aplicación autoexplicativa e intuitiva?
2. ¿Se usa un lenguaje adecuado?
3. ¿Son legibles los mensajes de usuario y proporcionan la información adecuada?
4. ¿Existe consistencia en el estilo, formato, convenciones, terminología, etc. de todas las páginas de la aplicación?
5. Cuando la seguridad es importante, ¿hay suficiente redundancia? (Por ejemplo, mostrando mensajes cuando se van a realizar cambios importantes para confirmarlos).
6. ¿Tiene el sistema un número excesivo de opciones?
7. ¿El sistema retorna algún tipo de información de recepción de entrada de datos? (Por ejemplo, mostrar ventanas emergentes cuando se han introducido o enviado datos confirmando que se ha realizado la operación satisfactoriamente).
8. ¿Es fácil navegar e interactuar con la interfaz? (Opciones de retorno, necesidad de usar minúsculas o mayúsculas, etc.)
9. En una segunda interacción con la interfaz, ¿el usuario recuerda fácilmente las distintas opciones?
10. Sugerencias

AI.3 Resultados obtenidos

La encuesta se realizó a 6 personas de diferente perfil: usuario doméstico, persona que dispone de móvil, pero carece de conocimientos de informática, usuario técnico, con conocimientos de informática a nivel de usuario, y experto, ingeniero informático. Los resultados se recogen en la Tabla 1.

Tabla AI.1. Respuestas de las personas encuestadas

Preguntas /Individuo	1. Usuario doméstico	2. Usuario experto	3. Usuario técnico	4. Usuario experto	5. Usuario doméstico	6. Usuario técnico
1	Sí	Sí	Sí	Sí, la interfaz es muy intuitiva y clara.	Sí	Sí
2	Sí	Sí	Sí	Sí	Sí	Casi siempre
3	Sí, aunque trataría de resaltarlos más	Sí	Sí	No todos los mensajes en la interfaz se ven. La información es correcta	Sí	Sí
4	Sí, es muy sencillo y fácil de usar.	Sí	Sí	Sí, todas guardan un formato similar	Sí	Sí
5	Sí	Sí	Sí	Sí, hay sistemas de confirmación	Sí	Sí
6	No	No	No	No, la aplicación cuenta con pocas opciones, pero son las necesarias	No	No
7	Sí	Sí	Sí	Los mensajes de retorno son correctos y claros	Sí	Sí
8	Sí, es fácil hasta para los más torpes con la tecnología	Sí	Sí	Para mi gusto, faltaría poner algún botón de retorno, pero siempre se puede volver a la pantalla anterior con la tecla del móvil asociada	Sí	Sí
9	Sí	Sí	Sí	Sí, recuerdas que tienes que pulsar para la información que quieres visualizar	Sí	Sí

Preguntas /Individuo	1. Usuario doméstico	2. Usuario experto	3. Usuario técnico	4. Usuario experto	5. Usuario doméstico	6. Usuario técnico
10	El icono de la aplicación, cuando se muestra en el menú del Smartphone, tiene parte transparente. Una sugerencia es modificar el color de dicho icono para que sea más fácilmente visible entre todas las apps.	Cambie el color de fondo del icono, al ser negro no se aprecia	La gráfica cubre parte de los datos.	a. Cambio en la pantalla de inicio: Que la gráfica no se muestre directamente si no que comience sin nada, y al seleccionar una opción aparezca. b. Poner una pantalla de bienvenida c. Alguna forma de cerrar la aplicación (por ejemplo, dar dos veces seguidas al botón retroceso, o incluso añadir una opción de Cerrar) d. El icono del menú es demasiado grande y se ve pixelado. Redimensio-naría la imagen para que no sea tan visible la distorsión y que no ocupe tanto en el propio menú e. Escalar más la gráfica. Aunque es perfectamen-te legible, no se interpreta correctamen-te las gráficas. Faltaría una leyenda de las ordenadas y abscisas. Y reescalar para que salgan más números.		Poner las letras y números de la gráfica más grande

AI.4 Conclusiones

Tras analizar los resultados obtenidos, se puede concluir que la aplicación ha conseguido cumplir las expectativas de ser fácil e intuitiva, no necesitar un periodo de aprendizaje largo. La mayoría de las respuestas coinciden en que la aplicación es autoexplicativa y con una apariencia consistente, lo que facilita su uso.

Las sugerencias aportadas por los usuarios que han realizado la encuesta son muy oportunas, por lo que serán tomadas en cuenta como mejoras futuras (Ver apartado 7.2 Trabajos futuros de este documento).

Anexo II. Aceptación del cliente

En este anexo se recoge la carta de aceptación, firmada por los dos clientes del proyecto, tras llevar a cabo las pruebas de aceptación de la aplicación, mostrando así que la aplicación cumple las expectativas de ambos. Dicha carta puede verse en la Figura AII.1

conMymo OFT, S.L.
B39765136
Isabel Torres 11, Edificio 3000, PCTCAN
39011 Santander
Tel. / Fax: 942266428
www.conmymo.com

Dº Jaime Gómez-Acebo con DNI 72052874S y Dº Jose Salmón García con DNI 20200992T en calidad de directores de proyectos de la empresa Conmymo OFT S.L.,

HACEN CONSTAR:

Que Dª Rebeca Bárcena Orero, con DNI 72081914Y, alumna del Grado en Ing. Informática de la Facultad de Ciencias de la Universidad de Cantabria, nos hizo entrega de su software en fecha 31 de mayo de 2016 y, tras unas pruebas funcionales, certificamos que cumple todos los requisitos especificados, aceptando, por tanto, el producto desarrollado.

El sencillo diseño de la interfaz favorece una rápida curva de aprendizaje de sus usuarios y reduce las reticencias de éstos a utilizar la aplicación.

Se debe remarcar la calidad del código desarrollado así como su documentación lo que facilitará su mantenimiento.

Y, para que conste a los efectos oportunos, expido el presente certificado, a petición de la persona interesada, en Santander, a 10 de junio de 2016.

Fdo.: Jaime Gómez-Acebo Ara

Fdo.: Jose Salmón García

Figura AII.3. Carta de aceptación

Bibliografía

[1] *España, primer país europeo en el uso de 'smartphones' en 2014, con un 81%.* (21/01/2015). Recuperado el 16 de marzo de 2016, de

http://www.rtve.es/noticias/20150121/espana-primer-pais-europeo-uso-smartphones-2014-81/1084704.shtml

[2] *Conoce (bien) los principales sistemas operativos móviles*. (13/9/2014). Recuperado el 16 de marzo de 2016, de

http://blogthinkbig.com/sistemas-operativos-moviles/

[3] *Explicación del roaming móvil.* (s.f.). Recuperado el 20 de marzo de 2016, de

http://www.gsma.com/publicpolicy/wp-content/uploads/2012/08/GSMA-Mobile-roaming-web-Spanish.pdf

[4] *Android* (s.f.). Recuperado el 3 de agosto de 2015, de

https://es.wikipedia.org/wiki/Android

[5] *Bases de Datos en Android (I): Primeros pasos.* (31/01/2011). Recuperado el 7 de agosto de 2015, de

http://www.sgoliver.net/blog/bases-de-datos-en-android-i-primeros-pasos/

[6] *MySQL*. (s.f). Recuperado el 20 de agosto de 2015, de

https://es.wikipedia.org/wiki/MySQL

[7] *Google Cloud Messaging: Overview.* (s.f.). Recuperado el 25 de agosto de 2015, de

https://developers.google.com/cloud-messaging/gcm#arch

[8] *Google Cloud Messaging. Parte I: Introducción*. (18/1/2013). Recuperado el 25 de agosto de 2015, de http://belencruz.com/2013/01/google-cloud-messaging-parte-i-introduccion/

[9] *Android Notification Library – Introduction.* (s.f.). Recuperado el 14 de agosto de 2015, de

http://apps.buzzrank.com/android-sdk/

[10] *Android Plot*. (s.f.). Recuperado el 20 de marzo de 2016, de

http://androidplot.com/

[11] *Instalar Tank Auth en CodeIgniter.* (12/5/2012). Recuperado el 30 de marzo de 2016, de

http://www.elreplicante.com.ar/2012/05/12/instalar-tank-auth-en-codeigniter/

[12] *Eclipse*. (s.f.). Recuperado el 30 de marzo de 2016, de

https://es.wikipedia.org/wiki/Eclipse_(software)

[13] *Eclipse IDE*. (s.f.). Recuperado el 30 de marzo de 2016, de

http://www.genbetadev.com/herramientas/eclipse-ide

[14] *Meet Android Studio*. (s.f.). Recuperado el 30 de marzo de 2016, de

http://developer.android.com/intl/es/tools/studio/index.html

[15] *MagicDraw UML*. (s.f.). Recuperado el 30 de marzo de 2016, de

https://es.wikipedia.org/wiki/MagicDraw_UML

[16] *CakePHP – Codeigniter Benchmark*. (s.f.). Recuperado el 28 de marzo de 2016, de

http://web.archive.org/web/20100924005145/http://www.sellersrank.com/php/cakephp-codeigniter-benchmark/

[17] *Web frameworks benchmark*. (s.f.). Recuperado el 28 de marzo de 2016, de

http://www.sellersrank.com/web-frameworks-benchmarking-results/

[18] *SonarQube*. (s.f.). Recuperado el 30 de marzo de 2016, de

http://www.sonarqube.org/

[19] *BitBucket: Mi mejor alternativa frente a GitHub*. (s.f.). Recuperado el 20 de marzo de 2016, de

http://blog.desdelinux.net/bitbucket-mi-mejor-alternativa-frente-a-github/

[20] *A free Git & Mercurial client for Windows or Mac.* (s.f.). Recuperado el 30 de marzo de 2016, dehttps://www.sourcetreeapp.com/

[21] *SourceTree, cliente GUI para manejar repositorios Git o Mercurial, llegará en breve a Windows*. (s.f.). Recuperado el 30 de marzo de 2016, de

http://www.genbetadev.com/sistemas-de-control-de-versiones/sourcetree-cliente-gui-para-manejar-repositorios-git-o-mercurial-llegara-en-breve-a-windows

[22] *Microsoft Project* (s.f.). Recuperado el 12 de abril de 2016, de

https://es.wikipedia.org/wiki/Microsoft_Project

[23] *Robotium* (s.f.). Recuperado el 15 de abril de 2016, de

https://github.com/RobotiumTech/robotium

[24] *Robotium – About us* (s.f.). Recuperado el 15 de abril de 2016, de

http://robotium.com/pages/about-us

[25] *Tasty mocking framework for unit tests in Java*. (s.f.). Recuperado el 10 de abril de 2016, de

http://mockito.org/

[26] *Junit*. (s.f.). Recuperado el 11 de abril de 2016, de

https://es.wikipedia.org/wiki/JUnit