Connect2, Launcher Android configurable desde una interfaz web.

Grado de ingeniería informática
Proyecto fin de grado
Junio de 2015, Las Palmas de Gran Canaria

Autor Juan David Vega Rodríguez
Tutores José Juan Hernández Cabrera y José Évora Gómez
Dpto. Informática y Sistemas

Agradecimientos

Lo que me motiva a escribir estas líneas es que soy consciente de que no estaría escribiendo este documento hoy si no hubiera sido por todas esas personas que me he encontrado en el camino, familia, amigos, mentores y profesionales de una calidad impecable. Es cierto que hubo momentos malos, pero seguramente si tuviera que elegir algo de toda esta etapa, me quedaría con las personas, personas que han hecho que yo hoy tenga claro a lo que me quiero dedicar el resto de mi vida. Y a mi corta edad eso vale más que cualquier otra recompensa o premio.

Primero tengo que agradecer de mi familia, de mi padre y mi madre que durante todos mis años educativos jamás me preguntaron una nota, o en que puesto había quedado, siempre me enseñaron que superarse a uno mismo y ser feliz era lo importante. Además tengo la suerte de ser el más pequeño de cinco hermanos, así que desde pequeño he crecido sobre cuatro pilares que me han mantenido firme. Tengo claro que cualquiera de ellos esta tan contento de lo que he conseguido como yo.

Es imposible no pensar en Yaneli, después de todo hemos llegado juntos aquí y durante este tiempo he vivido con la tranquilidad y felicidad de aquel que se sabe querido, apoyado y escuchado porque aunque le guste negarlo, la carrera la hice yo, pero parte de la culpa de que llegará hasta aquí es suya. En momentos en los que ya había tirado la toalla fuiste el apoyo que me hizo continuar.

Fuera de mi familia, la primera persona de la que tengo que acordarme es de Dani, pues en realidad la mitad de mi título le pertenece a él, era muy poco probable que dos personas tan distintas acabaran trabajando juntos durante tres años, día a día y asignatura a asignatura, y es que podría contar las ocasiones en las que hicimos un trabajo, una práctica o incluso que estudiamos por separado.

En el apartado más profesional en los últimos dos años es imposible no pensar en Johan, José, Cristopher y José Juan, gracias por crear Gran Canaria Ágil, gracias por enseñarme lo bonito de mi profesión, y darle sentido a la frase *''Elige un trabajo que te guste y no tendrás que trabajar un día en tu vida''*, enseñarme un mundo en el que la programación

es un arte, una tarea divertida y gratificante. Dentro de la comunidad no puedo evitar nombrar a Ronny ‘‘50 cent’’, gracias por dejarme ser tu mentor, por confiar en mí y luego convertirte en un amigo y un compañero de aprendizaje.

Y por ultimo gracias a todos los que me han apoyado de alguna manera, porque sin ellos estos años no tendrían ningún sentido.

Gracias

Índice general

Capítulo 1

Problema y Solución

El aspecto más importante en el desarrollo de productos y servicios software es la utilidad que ellos suponen a la sociedad, contribuyendo en su beneficio. En el sector de las aplicaciones móviles se tiende a la especialización: las compañías desarrollan aplicaciones que resuelven una necesidad concreta y los usuario están habituados al uso de una aplicación para un único propósito. Es por eso que tenemos una aplicación para el correo, una para el calendario, otra para las tareas de la vida persona, un lector de pdf, una que sincroniza la nube, otra para reproducir música, etc.

En las últimas actualizaciones del sistema operativo Android se ha tratado de mejorar la experiencia de usuario y simplificar su uso. Además, se ha dotado de mayor libertad, lo cual es una ventaja para usuarios avanzados, pero que se convierte en un inconveniente para usuarios novatos, quienes pueden tener dificultades con la introducción de estas libertades al haberse acostumbrado previamente a un modo concreto de hacer las cosas.

Después de la experiencia como desarrollador y usuario de Android, se ha identificado la necesidad de una aplicación, que permita configurar la pantalla principal de un dispositivo Android de manera remota. De este modo, se pretende que la pantalla principal vaya modificándose remotamente de acuerdo con las necesidades que pueda tener el usuario del dispositivo, sin que haya ningún tipo de extra que no aportan valor al usuario. En la siguiente sección se detalla cual es el público objetivo al que este producto está destinado. Además, se ha tenido en cuenta el reducir la interfaz de la pantalla principal del dispositivo móvil a elementos simples y fáciles de usar.

En lo profesional, como desarrollador que se encuentra en fase de entrada al mercado laboral y con una ideología alineada al Software Craftsmanship [9], para mi era muy importante aplicar metodologías de desarrollo ágil. Es decir, la definición de requisitos mediante iteraciones, entregas frecuentes de prototipos funcionales, ver de primera mano

como afecta a la estabilidad y calidad del código el cambio de requisitos continuo y la adaptación de la solución mediante fases exploratorias. Y todo esto en el desarrollo de un proyecto multiplataforma y políglota.

A continuación se mencionan contextos en los que esta necesidad identificada puede ser útil. Después, se presentará brevemente la solución a la que se ha llegado.

1.1. En el ámbito de la empresa

Gran número de empresas dan a sus empleados terminales para ayudarles en alguna faceta concreta del trabajo. En este aspecto existen algunos elementos a tener en cuenta por parte de la empresa:

1. Por lo general cuando una empresa presta un terminal a un empleado para que realice algún tipo de tarea, éste suele mantener el teléfono consigo durante toda la jornada laboral o incluso se lo lleva a su casa. Uno de los problemas a los que se puede enfrentar la empresa es que los empleados den un uso personal al terminal, sin permiso de la empresa.

2. Son muchas las empresas ya poseen su propia aplicación para realizar todas las tareas necesarias. Sin embargo, esta aplicación se ejecuta sobre un dispositivo que proporciona muchas más funciones. Eso conlleva costes en el mantenimiento del dispositivo ya que pueden ocasionarse fallos en el funcionamiento del mismo por aplicaciones ajenas a la empresa.

3. No todos los empleados tienen la misma capacidad para adquirir las habilidades necesarias en el uso de este tipo de terminales. Por ello, es importante facilitar en la medida de lo posible la interacción del mismo con el dispositivo móvil, de tal modo que el trabajador sea más productivo y no pierda tiempo por culpa de una barrera tecnológica.

4. El hecho de que estos dispositivos deban ser actualizados periódicamente (ya sea por una actualización de la aplicación o del propio sistema operativo) obliga a que la empresa tenga que tener un departamento especializado y un stock de dispositivos superior al número de trabajadores. Esto es así puesto que muchas de estas actualizaciones podrían requerir que el dispositivo se llevará a dicho departamento al no ser posible actualizar remotamente la aplicación.

1.2. En el ámbito social

Para una persona mayor que ya posee un terminal Android, se suele hacer muy complejo su uso por la gran cantidad de menús que posee, lo fácil que es realizar una acción sin querer, los múltiples escritorios que posee la pantalla principal, etc. Es muy común que este tipo de usuario se apoye en un familiar para que le ayude a configurar el teléfono a sus necesidades.

Durante los últimos años, han aparecido varias alternativas buenas a la pantalla principal por defecto. Dentro del colectivo de personas mayores, existen soluciones que reducen la dificultad de uso que ofrece el sistema. Para poder definir realmente cual era la situación a la que nos exponíamos en el nicho de mercado objetivo, realicé una investigación de las alternativas más famosas parar mejorar la experiencia de las personas mayores en Android. Se podría decir que las técnicas para la mejora de la accesibilidad que esas soluciones tienen pueden categorizarse en dos ramas:

1. Aumentar el tamaño de los elementos de la interfaz de usuario, botones e iconos más grandes, y como efecto lo que notamos es una menor cantidad de iconos a simple vista y en muchos casos se cambia los iconos por otros que son más intuitivos.
2. Filtro de aplicaciones que permite definir que aplicaciones puede ver el usuario y en que orden. Como complemento algunas alternativas ofrecen también la posibilidad de añadir contactos de ‘‘tocar y llamar’’.

Algunas de las soluciones abordaban las dos ramas, unas se centraban solo en una, pero todas tenían la misma filosofía, eliminar elementos de la interfaz y añadir menús de configuración.

1.3. Objetivos iniciales

Después de evaluar las soluciones que se ofrecen actualmente mi objetivo inicial es: Desarrollar una aplicación Android que permita la configuración remota de la pantalla principal del dispositivo y que además ofrezca una buena experiencia de usuario y sea capaz de limitar las funciones del dispositivo para adaptarse a las necesidades de los usuario. Como objetivos específicos destacaría:

- Durante el tiempo que dure el desarrollo del trabajo, se pretende explorar la mejor forma de facilitar el uso del sistema operativo Android a los usuarios que presenten dificultades contando con el apoyo de un familiar y/o amigo.

- el desarrollo del proyecto será Open Source aunque se decida explotar comercialmente
- Usar un proceso iterativo e incrementar en el que semanalmente se entrega una pieza de valor que supone un incremento funcional.

Para cumplir los objetivos arriba descritos y hacerlo de la mejor manera posible, era necesario un conocimiento amplio sobre como se estructura el sistema operativo Android, entender en que apartado encaja el Launcher y las herramientas que ofrece para realizar el proyecto que se plantea. A continuación explicaré de manera breve que es un Launcher y como encaja en la estructura del sistema operativo.

1.4. ¿Qué es un Launcher?

En el contexto del sistema operativo Android, un Launcher es una aplicación que, entre otras cosas, contiene la pantalla inical. El primer paso para comprender que es un Launcher, desde el punto de vista técnico, pasa por entender qué es y cómo esta estructurado el sistema operativo Android.

Android es un sistema operativo con núcleo unix, diseñado en un origen para Smartphones y que hoy en día tiene una fuerte presencia en la industria estando disponible en Smartphones, Tablets, Relojes, Televisiones y coches. El sistema operativo posee una versión Open Source y la empresa Google se encarga de la linea principal de desarrollo del núcleo y servicios del sistema. Sin embargo, su aspecto Open Source ha hecho que muchos terminales con buenas especificaciones sean asequibles. Esto se traduce en unas cuotas de mercado que han llegado al noventa por ciento [2], por tanto, el desarrollo de productos para Android cuenta con un amplio mercado.

A nivel técnico el sistema operativo esta estructurado en una serie de capas, en la figura 1.1 podemos ver que en el nivel más bajo encontramos el kernel de Linux en el que están disponibles todas las funciones de bajo nivel que tienen acceso directo al hardware, todo tipo de drivers, gestión de la energía y el ciclo de vida del teléfono (por ejemplo autoencendido).

En la segunda capa desde abajo, encontramos la capa de abstracción de Hardware (HAL), una capa en la que se encapsulan todas las peticiones que el resto de capas realiza para interactuar con los driver que proporciona el núcleo.

Justo por encima encontramos la primera capa al que el desarrollador tiene acceso directo, Las librerías de C/C++ o Librerías Nativas, las cuales tienen su propio Kit de

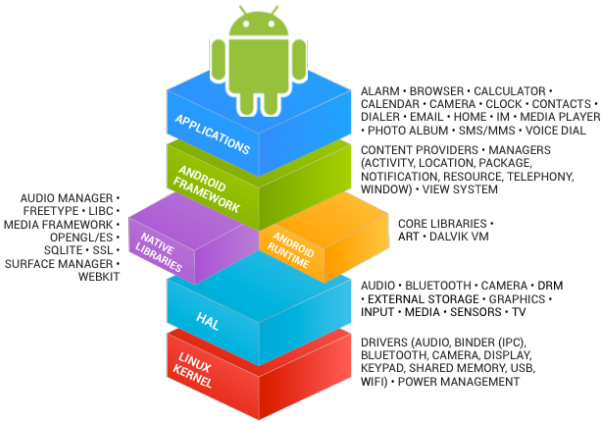

Figura 1.1: Estructura del Sistema Operativo

desarrollo (NDK) y se usan sobre todo en el desarrollo de procesamiento de imagen y/o sonido.

La capa al mismo nivel en la jerarquía que la anterior es el Android Runtime (ART) que es la adaptación de la JVM para Android, y sobre la que se ejecutan los ficheros DEX que simulan los fichero CLASS que se incluyen en los comprimidos APK que se asemejan a los JAR de una JVM tradicional.

La ultima capa a la que no tiene acceso al usuario es el Framework de aplicaciones, sobre el que se diseñan y estructuran las aplicaciones y el cual proporciona las funciones que los desarrolladores usamos para crear nuestras aplicaciones.

Al más alto nivel en una capa que es accesible durante el uso cotidiano del móvil lo que tenemos sería el Contenedor de Aplicaciones, el cual, incluye todas las aplicaciones que pueden ser ejecutada por el usuario.

Es importante comprender que cualquier aplicación que se encuentre en el Contenedor de Aplicaciones ha sido desarrollada usando algunas de las librerías y/o framework que se encuentran en las capas inferiores. El contenedor en si mismo no puede listar o abrir aplicaciones. Tenemos que un tipo de aplicación que usando las herramientas del framework nos da la capacidad de listar, abrir, añadir y eliminar aplicaciones (Figura: 1.2), es decir, un ‘‘Lanzador’’ (Launcher) es una aplicación desarrollada exactamente de la misma manera que el resto de las aplicaciones del contenedor pero con dos características muy especiales:

- Responde al evento ‘‘Home’’, todos los terminales Android poseen tres botones (Figura: 1.3), el del centro es conocido como ‘‘Home’’ y se usa para minimizar todas las aplicaciones en primer plano, una vez que eso ocurre el sistema operativo genera un evento, que es capturado por el lanzador permitiéndole ejecutarse (o

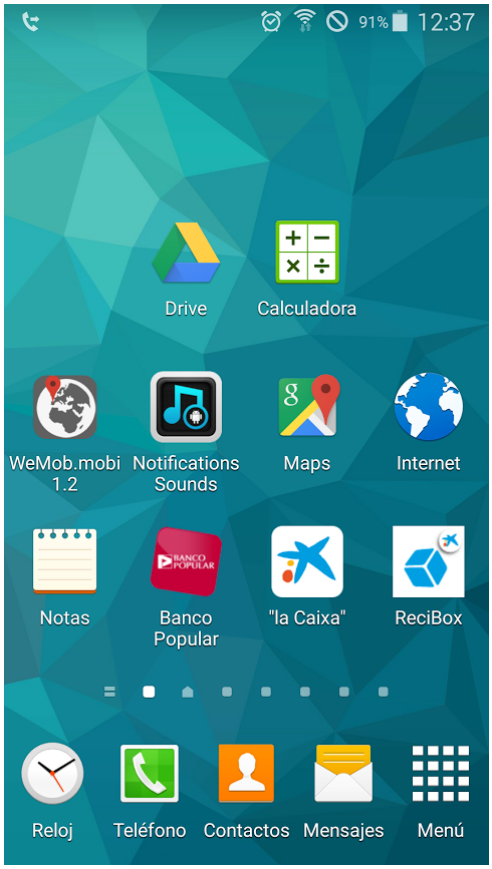

Figura 1.2: Pantalla de inicio en TouchWiz Launcher

Figura 1.3: Botones clásicos botones de un terminal Android

pasar a primer plano), haciendo que funcione como el ‘‘Escritorio’’ que estamos acostumbrados a tener en nuestro ordenadores.

- Solo existe una instancia de la aplicación que no puede morir. Una vez se lanza una instancia de la aplicación, cuando el usuario lanza otra aplicación nuestro Launcher pasa a segundo plano. También es importante entender que todos los servicios, asociados a la aplicación continúan en ejecución. Lo que nos permite tener un control de todo lo que sucede en el teléfono, y una interfaz en la cual reflejarlo para darle información al usuario.

Para tener el control suficiente que requiere el proyecto el desarrollo de la parte que se ejecuta en el terminal es un Launcher, que en lugar de listar aplicaciones u ofrecer las características típicas de configuración, posee un modulo que es capaz de escuchar las peticiones del servidor para notificar que la configuración debe cambiar.

1.5. Solución

Este proyecto se ha orientado a crear un Launcher que permite la configuración remota a través de una web, se puede cambiar en tiempo real la pantalla de inicio del teléfono, puesto que desde la web se puede: enviar mensajes, añadir aplicaciones y contactos. La pantalla de inicio ha sido simplificada, Además la aplicación elimina de la vista del usuario los elementos que no son importantes. En resumen se puede adaptar la pantalla de inicio de manera fácil a cada usuario.

Teniendo en cuenta los ámbitos arriba descritos hemos detectado dos nichos de negocio para el proyecto:

Figura 1.4: Estructura del proyecto

- La empresa (sección: 1.1), suele tener móviles que puede dejar a sus empleados para facilitar el desempeño de su labor durante la jornada laboral. Con nuestra solución tenemos un dispositivo que tiene solo la información y funcionalidad necesarias, no se puede hacer un uso indebido del terminal, no es necesario que el empleado este acostumbrado al uso de ese sistema, se mantiene la potencia del sistema operativo Android y su gran robustez pero adaptada a la necesidad de la relación empresa-empleado.

- Supongamos un adulto joven que tiene algún mayor a su cargo, por ejemplo, su abuela, (sección: 1.2) es común que el rechazo de la persona mayor hacia la tecnología se deba a la frustración que produce al usarla cuando no se conoce bien , para que la persona mayor tenga un uso continuado del móvil el adulto joven, su nieto, tendría que estar físicamente configurando el dispositivo a mano, añadiendo las aplicaciones, alarmas, ect. Nosotros no solo proponemos añadir las funciones que las alternativas ya poseen (sección: 1.2), nuestro valor se encuentra en poder realizar todas las tareas desde tu casa, de manera simple y cómoda en una web sin tener que estar físicamente con el terminal.

El proyecto ha sido dividido en dos grandes bloques (Figura 1.4), la aplicación del dispositivo y la web, para la interacción de estos bloque se usan librerías de terceros que serán explicadas más adelante en este documento (sección: 3.2, la totalidad del proyecto esta desarrollada en Java, ayudándose de HTML, CSS, JavaScript, Hazelcast y el Framework de Android.

El objetivo primordial es conseguir un producto mínimo viable que represente las soluciones que proponemos en este apartado y que pueda ser probado para refutar nuestra hipótesis de que realmente existe esa necesidad.

Para conseguir que realmente se aplique el producto a los ámbitos descrito es necesario trabajar en que no existan impedimento a la hora de comenzar a usarlo o durante su uso regular, es por esto que nuestro flujo objetivo es que el usuario solo tenga que:

1. Crear una cuenta en la web.
2. Descargar el Launcher en el dispositivo, ejecutarlo y acceder con una cuenta de usuario.
3. Ir a la web y seleccionar el dispositivo para comenzar a trabajar con él.

Tanto en los sucesivos prototipos como el producto final se tiene muy en cuenta la mejora continua de la usabilidad, es una de nuestras bases y sobre la que se pivota para aportar al proyecto un gran valor de cara al uso diario de los usuarios.

Capítulo 2

Resultados

Cuando se comienza el desarrollo de un proyecto y se realiza un análisis de los nichos de mercado y los posibles problemas que prendemos cubrir es común que caer en la trampa de creer saber lo que el cliente quiere sin validarlo.

Influenciado por el método Running Lean [10] , nuestro enfoque fue usar el tiempo disponible en el desarrollo del trabajo de fin de título para desarrollar un Producto Mínimo Viable (MVP) cuyo objetivo no es salir a la venta pública y competir en el mercado sino validar, de manera cualitativa, si realmente nuestro producto tiene aplicación en los nichos de mercado identificados y si de verdad el cliente final percibe como problemas los aspectos que se pretenden solucionar.

En las próximas secciones voy a explicar parte a parte la arquitectura del sistema (sección: 2.1), la interfaz de usuario (sección: 2.2), el protocolo de comunicación (sección: 2.3), la arquitectura interna de la aplicación (sección: 2.4) incluyendo persistencia y por ultimo la arquitectura interna del servidor (sección: 2.5), también con su persistencia.

2.1. Arquitectura del sistema

El sistema esta constituido por tres grandes bloques, la aplicación móvil, la web y el servidor. A continuación voy a resumir la configuración del software de terceros y hardware para cada uno de los bloques que se detallan en la arquitectura del sistema (Figura: 2.1).

La aplicación web se ejecuta en un navegador, actualmente la web ha sido probada en Firefox 35+ y Chrome 40+ sobre el sistema operativo Windows 8.1, usando pantallas de 1366x768 y 1920x1080 píxeles, sin embargo, el uso del framework Materialize (sección:

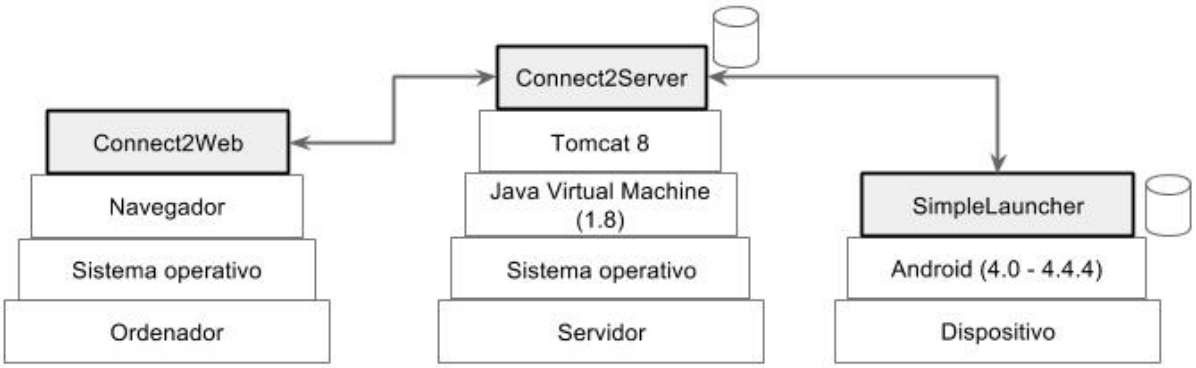

Figura 2.1: Arquitectura del sistema

3.2.1) nos asegura compatibilidad con: Chrome 35+, Firefox 31+, Safari 7+, IE 10+ en todos los sistemas operativos y pantallas, gracias a esto podemos decir que el navegador, el sistema operativo y la pantalla del usuario no afectan a la ejecución de nuestra web.

El servidor se esta ejecutando sobre el contenedor de aplicaciones Tomcat 8 [12], dado que la aplicación esta escrita en Java es necesario que en la maquina principal este instalada la Java Virtual Machine (JVM) 8 y esto hace que realmente no importe cual sea el sistema operativo siempre y cuando se tenga una versión de JVM 8 para ese sistema operativo, actualmente el servidor de despliegue es un Ubuntu 14.04, tiene 512 MB de RAM y un procesador mono núcleo a 2.4GHz. También se han hecho pruebas con Tomcat 8 y la JVM 8 en su versión para Windows 8.1 en un ordenador con 8 GB de RAM y un procesador de cuatro núcleos a 2.5GHz.

El Launcher Android ha sido desarrollado usando la API 15 lo que significa que podría ejecutarse en cualquier sistema operativo Android 4.0+. No obstante solo se ha probado en Android 4.0, 4.1 y 4.4, las versiones conocidas comercialmente como Ice Cream Sandwich, Jelly Bean y KitKat, lo que supone un 81 % (Figura: 2.2) de los dispositivos actuales según datos de la web oficial de Android [6]. Los dispositivos concretos en los que hasta ahora he ejecutado la aplicación son: Samsung Galaxy S3(4.0), Samsung Galaxy S4(4.1) y Nexus 5 (4.4).

En el capitulo anterior (capítulo: 1) comenté que nuestro flujo objetivo sería: el usuario instala la aplicación y hace login, luego va a la web y ya tiene el dispositivo disponible para seleccionarlo y comenzar a configurar. Esta es una de la razones por las que ha sido necesario construir el sistema dividido en 3 bloques, la web, el dispositivo y el servidor que funciona como intermediario. A nivel de técnico, el flujo que buscábamos implica los siguientes pasos:

1. Cuando el usuario hace login en la aplicación se registra el dispositivo en el Google Cloud Messaging(GCM) (sección: 3.2.5), una vez el dispositivo recibe su token único

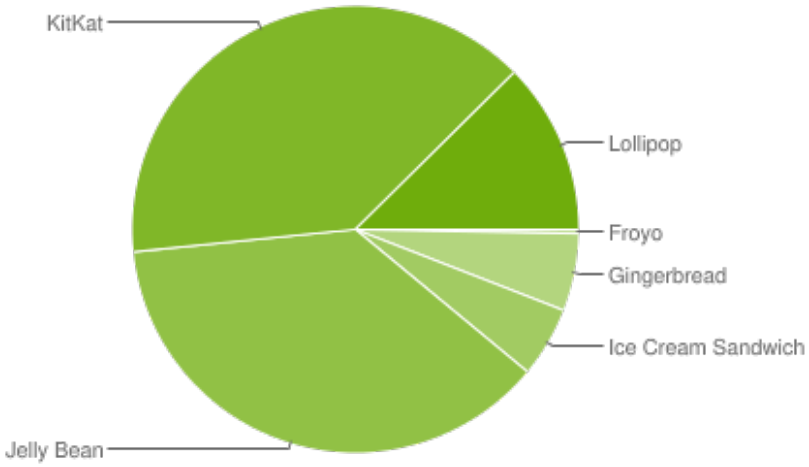

Figura 2.2: Porcentaje de versiones instaladas Android en el mundo

del GCM. Hace una petición a mi servidor con la siguiente información: que número identifica ese dispositivo, e-mail, contraseña y el token del GCM.

2. Luego El dispositivo se registra y asocia a un usuario. Tras la respuesta positiva, el dispositivo le pide la configuración y el servidor se la envía.

3. Ahora el usuario selecciona el dispositivo en la web, lo que genera una llamada al servidor que le responde de manera parcial informándole que la petición fue recibida, y mantiene viva la conexión HTTP almacenándola en memoria.

4. El servidor usa el GCM y con el token único, envía al dispositivo el comando ‘‘ReadApps’’, que cuando es recibido por el dispositivo, éste envía al servidor una lista de las aplicaciones instaladas.

5. Una vez recibe la respuesta de las aplicaciones instaladas, usa la petición HTTP que dejo abierta en el paso 3 para notificar al navegador del usuario cuales son los datos que ha recibido del móvil.

6. Por ultimo el usuario puede ver una vista previa de las aplicaciones ya activas en el móvil y tiene la posibilidad de añadir una de las ya instaladas que acaban de ser leídas del móvil.

En la descripción anterior vemos un ejemplo que implica todas las tecnologías usadas en el proyecto, el como se han relacionado estas tecnologías con el software desarrollado por mi hace que el proyecto sea adaptable de manera sencilla tanto en el ámbito social como en el ámbito de empresa. En apoyo a la capacidad de adaptación que el proyecto tiene, posee una serie de test que aseguran que la funciones comunes que están presentes

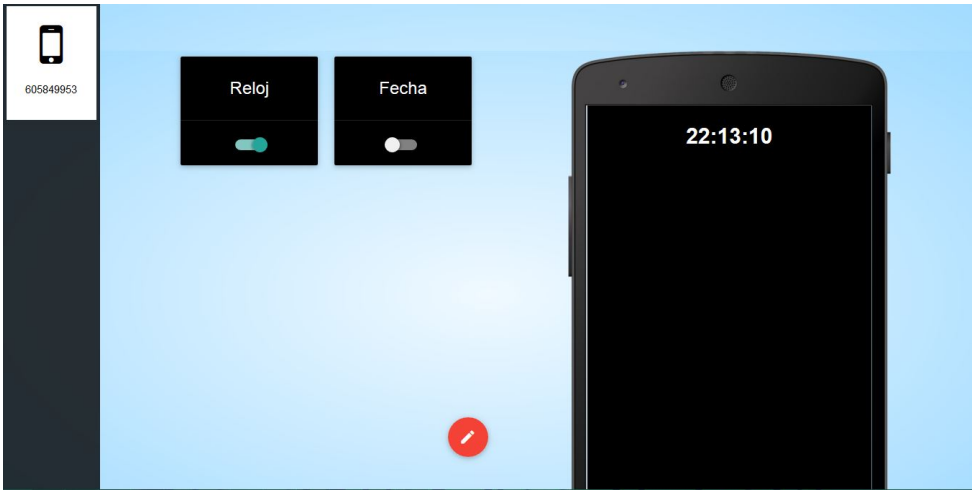

Figura 2.3: Pantalla principal de usuario

en las diferentes líneas de desarrollo se mantienen intactas y funcionan igual cualquiera que sea el entorno que las rodee.

Otro punto a tener en cuenta en la escalabilidad y evolución (capítulo: 4), ha sido la importancia de desarrollar la base de datos como un modulo aparte que se exporta como una interfaz. Actualmente el proyecto esta usando una implementación de esa interfaz que tiene como motor de persistencia Hazelcast(sección: 3.2.6), aunque cualquier otra clase que implemente la interfaz podría ser sustituida por la actual y el resto del sistema no notaría la diferencia.

Durante todo el desarrollo del proyecto se han tenido en cuenta los principios SOLID [1] y especialmente la inyección de la dependencia.

Las funciones de la web han sido creadas de manera que sea muy simple aprender a usarla y además todo se encuentra expuesto (Figura: 2.3) de manera que no es necesario navegar por menús para encontrar las opciones y en ningún caso es necesario recargar la pagina, por tanto, se obtiene una muy buena experiencia de usuario donde funciones se descubren y usan de manera natural. Además la vista previa a la derecha de la pantalla (Figura: 2.3), refleja lo que el operario ve en la pantalla(Figura: 2.4), por tanto, la configuración remota no deja dudas sobre lo que esta viendo el usuario final.

Todas las opciones proporcionadas por el proyecto son exportadas para terceros mediante una API, que permite realizar las mismas funciones que la web hace sobre el móvil, esto aporta gran valor, dado que la integración en un entorno profesional en el que otras herramientas de gestión juegan un papel muy importante sería sencilla, lo que no requeriría de que el usuario conozca como funciona nuestra web, se le puede otorgar un entorno en el que este más cómodo.

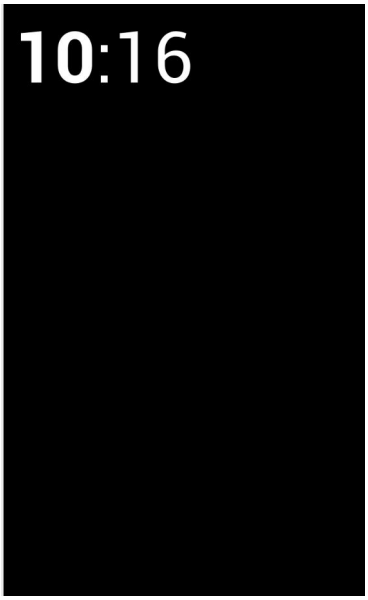

Figura 2.4: Esto sería lo que se ve en el dispositivo según la configuración en la figura anterior

2.2. Interfaz de usuario

La interfaz de usuario no es un bloque representado en el diagrama de estructura del proyecto, sin embargo, es un apartado que se encuentra tanto en la aplicación como en la web y que dado el objetivo de este producto, es uno de los aspectos que más se debe cuidar.

Como resumen general de la interfaz gráfica destacaría que para dar uniformidad al proyecto, dado que se trata de un proyecto Android, la experiencia de usuario de la web, se ha a usado ‘‘Material Design’’ [5], es la línea de diseño que usa Google desde que salio a la luz Android Lollipop y a la que se han adaptado todos los servicios y aplicaciones que se ofrecen para Android, la incorporación de ‘‘Material Design’’, ha sido gracias al Framework Materialize (sección: 3.2.1).

El reloj y la fecha, son elementos que hemos considerado básicos en cualquiera de los ámbitos en los que apliquemos nuestro producto, bien en el ámbito de empresas o en el ámbito social, puesto que la hora y la fecha a simple vista son elementos que ayudan tanto a la labor de un trabajador como al día a día de cualquier otro usuario. Para incluir estos elementos se tuvo en cuenta que tuvieran un alto contraste, y fueran fáciles de leer en cualquier condición.

En la interfaz del dispositivo, la pantalla de inicio esta simplificada haciendo que el usuario se sienta cómodo y además al ser un fondo negro el gasto de batería es menor [8].

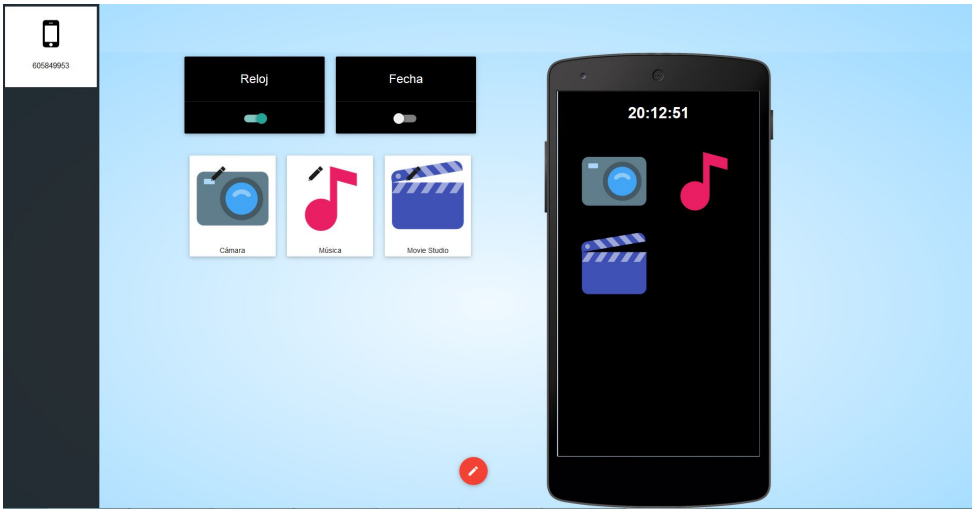

Figura 2.5: La web, después de seleccionar un dispositivo

La disposición de las aplicaciones en un cuadro ancho y con un margen entre ellas fue dispuesto de esa manera para reducir la posibilidad de hacer click en un elemento por error, pues dado el perfil del usuario al que va dirigido reducir la dificultad en el uso de la aplicación era un de nuestro objetivos iniciales.

Del otro lado tendríamos la interfaz de la web, simple y limpia. La información surge al ritmo que el usuario va realizando las acciones, de izquierda a derecha tenemos (Figura: 2.5): el listado de dispositivos disponible, el listado de aplicaciones activas en el dispositivo, el botón de editar y la vista previa del dispositivo.

Los elementos han sido diseñados usando Materialize (sección: 3.2.1), tienen una estética similar a la de Android. la disposición de los elementos se ha hecho en base a los pasos que debe seguir el usuario, es decir, lo elementos están colocados de manera que el usuario tiene la sensación de esta siguiendo un proceso, paso a paso, por ejemplo:

1. Paso 1, seleccionar un dispositivo (el menú más a la izquierda)
2. Paso 2, añadir una aplicación (centro de la pantalla)
3. Paso 3, ver resultado (elementos más a la derecha de la pantalla)

Esto hace que el uso de la web se vuelva intuitivo, la vista, el ratón y el flujo de información se mueven en un solo sentido, lo que permite al usuario ser capaz de seguir el proceso de manera intuitiva, es por esto que mencioné que la información surge conforme se realizan acciones, cuando el usuario selecciona un dispositivo, aparece la información relevante a ese dispositivo, cuando decide añadir una aplicación aparece el formulario necesario, cunado la añade, se actualiza la vista previa y se genera una notificación justo en la parte superior de la mitad derecha.

2.3. Comunicación

En el protocolo de comunicación intervienen: mi servidor propio, el usuario desde la web, el dispositivo y el Google Cloud Messaging (sección: 3.2.5). Dentro del protocolo de comunicación vamos a distinguir los siguientes procesos: Alta de dispositivo con un usuario, leer las aplicaciones instaladas en un dispositivo y enviar la configuración a un dispositivo.

Vamos a empezar por el proceso de alta de usuario que es lo primero que se realiza y donde se almacena toda la información para que el resto de procesos puedan suceder. Éste proceso posee los siguientes pasos:

1. El dispositivo se da de alta en el GCM y éste le proporciona un token único que usamos como dirección.
2. Con la dirección y la credenciales de usuario el dispositivo solicita al servidor darse de alta
3. El servidor crea un nuevo dispositivo asociado al usuario y le asigna como dirección el token del GCM.
4. El servidor informa al dispositivo de que el registro ha sido realizado de manera satisfactoria.
5. Y el dispositivo solicita al servidor que le mande la configuración.

Es importante entender que es el dispositivo el que primero se da de alta en el GCM y le proporciona la información necesaria para que este pueda generar una dirección que luego el servidor almacena para el resto de proceso.

El siguiente proceso después del alta de un dispositivo es leer las aplicaciones instalada en ese dispositivo puesto que a la hora de añadir una aplicación desde la web solo se permite añadir aplicaciones que ya estén instaladas. Para poder leer las aplicaciones ya instalas son necesarios mantener abiertos dos canales de comunicación: el del servidor con el navegador del usuario que solicita la información y que tiene al usuario esperando y el del servidor con el dispositivo que si no recibe un comando no sabe que tiene que enviar las aplicaciones instaladas, para que este proceso tenga éxito son necesarios los siguientes pasos:

1. El usuario en la web selecciona un dispositivo

2. Usando JavaScript desde el navegador solicitamos una conexión con el servidor y le mandamos el número del dispositivo.

3. El servidor almacena en memoria la conexión actual con el navegador y envía una petición al GCM pidiendo que envié un mensaje al dispositivo que dice ‘‘ReadApps’’

4. Cuando el modulo de recepción lee el mensaje ‘‘ReadApps’’ genera un objeto con toda la información, lo serializa y lo envía al servidor mediante HTTP POST en forma de JSON.

5. Al recibir la configuración del dispositivo que contiene información de quien genero el mensaje, el servidor busca la conexión que había dejado abierta y envía esa información al JavaScript en el navegador.

6. El JavaScript genera los elementos visuales para esa información y los muestra en la ventana para que los vea el usuario

Durante este proceso se mantiene viva la conexión con el navegador que es asíncrona y para ser capaz de enviar la información una vez el dispositivo conteste, el servidor mantiene la conexión almacenada en memoria con un identificado que el envía al dispositivo y este le devuelve, esta es la manera en la que el servidor gestiona las conexiones para que de la sensación sea que los datos fluyen desde el móvil a la web casi como si los leyera del servidor.

El último proceso a destacar dentro del protocolo de comunicación es el envío de la configuración, lo primero es entender que este proceso no implica una acción explicita del usuario, es un proceso que se ejecuta siempre que hay un cambio en la configuración del dispositivo y en realidad es importante por que se encarga de enviar toda la información en un JSON y gracias a la potencia del GCM (sección: 3.2.5) la labor es muy simple y el servicio hace que los mensajes no tengan latencia, puesto que esta pensando para mensajes de chat en vivo. La forma en la que se gestiona el envió de esa información es la siguiente:

1. El navegador informa al servidor que el usuario a realizado un cambio en un dispositivo y le proporciona el número.

2. El servidor carga desde la base de datos la configuración actual del dispositivo y la serializa en JSON

3. Genera un objeto mensaje que contiene la configuración y la dirección y pide al GCM que lo envíe al dispositivo

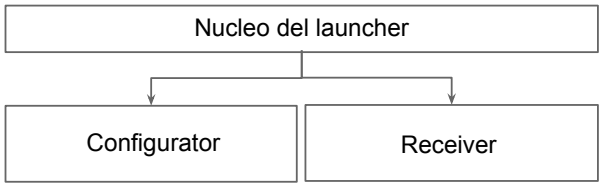

Figura 2.6: Estructura interna de la aplicación

4. Cuando el dispositivo recibe la información la deserializa y la hace llegar hasta la interfaz.

Otro elemento destacable es que el servidor no pide confirmación de la recepción por dos motivos muy simples, primero el GCM generaría una excepción en caso de que el dispositivo no recibiera ese mensaje y segundo antes de generar la excepción y en intervalos de tiempo regulares el GCM envía repetidamente el mensaje al dispositivo.

2.4. Dispositivo: SimpleLauncher

A la parte del producto que se ejecuta en el dispositivo le hemos llamado SimpleLauncher (Figura: 2.7), Esta aplicación es uno de los núcleos del proyecto, conseguimos que se comporte como un Launcher común que tiene un interfaz muy simple, pero en realidad en segundo plano se están ejecutando usa serie de servicios (Figura: 2.6) que se encargan de la gestión de la configuración y el envió, recepción y tratamiento de datos que provienen del servidor.

Los principales elementos que hacen robusta la aplicación son:

1. El ahorro de batería en todas las transacciones entrantes o salientes.
2. Tiempo de respuesta rápido ante una nueva configuración.
3. Abstracción entre el Configurador y la implementación concreta de Android gracias al patrón Holder
4. Cache para imágenes y reutilización de celdas.

La mayoría de los elementos están relacionados entre si, y es que, el uso del patrón Holder implica dos elementos cruciales en el uso responsable de los recursos, el primero es que las búsquedas de los componentes de la interfaz por Id solo se hacen la primera

Figura 2.7: Captura del Launcher

vez que se crea una celda, el segundo elemento de importancia es que nos permite tener almacenados objetos java con información del modelo que se esta mostrando en la interfaz, con lo cual no es necesario que se vuelva a realizar una asociación entre la vista y el modelo.

Otro elemento que ayuda al ahorro de batería y al cacheo de imágenes es el uso de la librería Universal Image Loader (sección: 3.2.7), una librería de gran potencia que se encarga de forma eficiente de gestionar las transacciones y el almacenamiento de imágenes.

El ahorro de batería en las transacciones es debido al uso de dos tecnologías distintas, una es Volley (sección: 3.2.8) que es una librería de cliente para generar peticiones HTTP y que se encarga de gestionar conexiones y colas para evitar el sobre uso de recursos y todo esto lo hace en un hilo a secundario, por tanto no afecta al rendimiento de la aplicación y el otro aspecto es el uso del GCM (sección: 3.2.5) que posee una optimización para conexiones entrantes mediante tecnología Push (sección: 3.2.9) especialmente diseñada para ejecutarse en dispositivos móviles con Android.

A continuación voy a describir como se relacionan y las funciones de cada uno de los módulos que encontramos dentro de la aplicación.

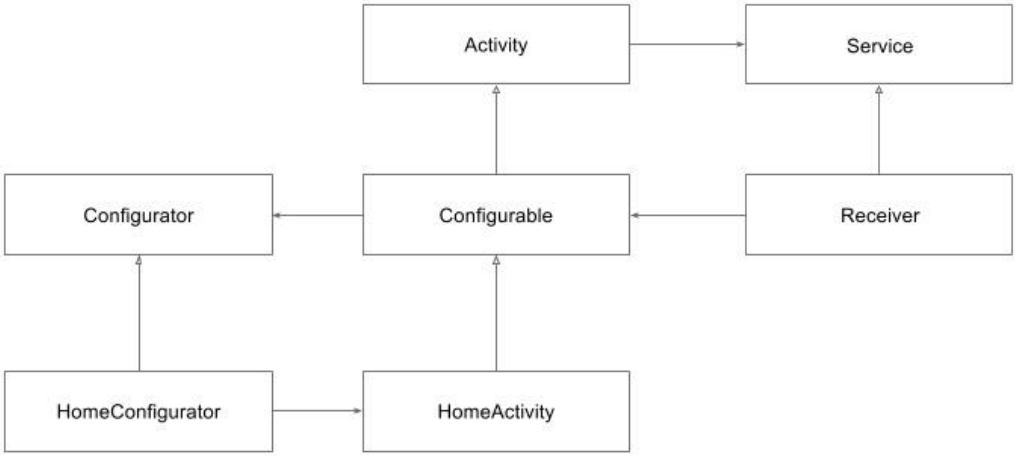

Figura 2.8: Estructura del Núcleo del Launcher

El Núcleo del Launcher, El HomeActivity es la instancia principal de la aplicación, nunca se finaliza, es el encargado de lanzar las aplicaciones y cuando se lanza alguna aplicación se queda en segundo plano. Además conoce a los otros dos módulos y funciona de enlace para hacer fluir los datos desde la información que se recibe del servidor hasta la interfaz que ve el usuario (Figura 2.8).

Para que el flujo de información sea posible, el HomeActivity es un Configurable lo que significa que es una actividad Android que esta obligada a tener dos métodos, el hasBeenConfigure y el update, en ambos métodos se relaciona a esta clase con un Configurator, en el inicio de la aplicación cuando se va a cagar la vista por primera vez o cuando se realiza una actualización el HomeActivity notifica a su configurator.

El HomeActivity como Configurable posee el método update, que es llamado por el modulo Receiver cuando hay una configuración disponible. Para que esto sea posible al inicio de la aplicación se debe relacionar al HomeActivity con el ReceiverService, esto se hace usando el mecanismo de ‘‘binding’’ que Android nos proporciona y que nos permite establecer una conexión entre la instancia de un Activity y un Service, una vez que se realiza la conexión, en tiempo de ejecución el Service tiene acceso a la instancia actual que se esta ejecutando del HomeActivity y cuando un mensaje del servidor con una configuración le despierta avisa al Home de que existen nuevos datos de configuración, de esta manera el Home puede notificarle al Configurator que existen nuevos datos.

El Configurator, es capaz de dada una configuración establecer como se deben ver y relacionar los elementos de la interfaz, se encarga de convertir en elementos visuales y accesibles por el usuario los datos que contiene la configuración que ha sido recibida desde el servidor(Figura 2.9).

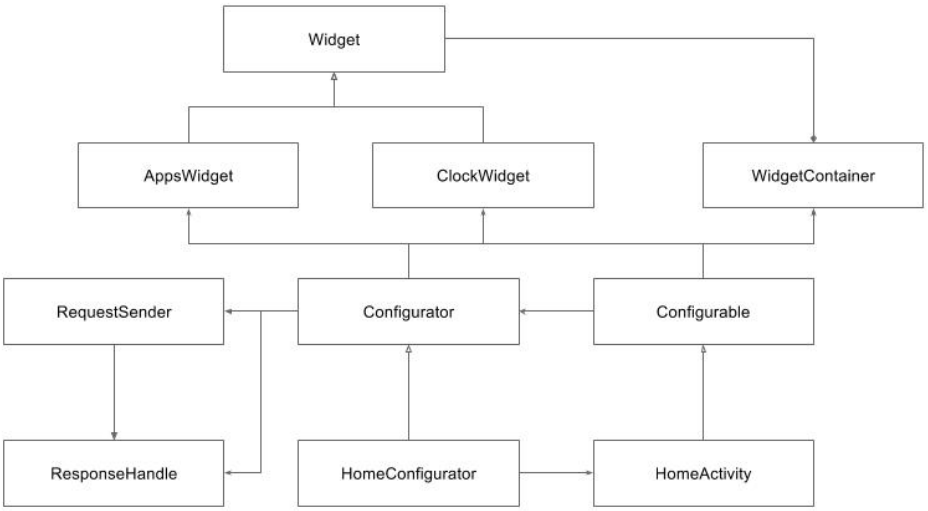

Figura 2.9: Estructura del Configurator

El RequestSender es clase encargada de realizar las conexiones salientes, el Configurator, usa esa clase para solicitarle la configuración actual es el único punto de salida al exterior y lo hace de forma asíncrona, por lo que es necesario que el Configurator se registre como ResponseHandler dentro de la instancia del RequestSender, una vez que finalice la solicitud que el Configurator lanza. el RequestSender le notifica a través del método onResponse del ResponseHandler que el Configurator registró.

Lo que hemos conseguido es asignar una única responsabilidad a las Clases o módulos, el Configurator se encargar de dada una configuración reflejar eso en Widgets que luego van a la interfaz, mientras que el RequestSender tiene la responsabilidad de enviar una petición concreta a una URL concreta y cuando reciba respuesta, avisar al que originalmente la envió.

Receiver Es un servicio que se encuentra dormido y que es despertado por el sistema operativo basándose en unos parámetros del GCM cuando se recibe información desde el servidor, es encargado de interpretar los diferentes comandos que se pueden recibir desde el servidor y realizar las operaciones para que esa información llegue al Núcleo (Figura 2.10).

Este es quizás uno de los elementos más complejos de la aplicación, pues supone un profundo conocimiento del funcionamiento del framework de Android y puede hacer que la aplicación entera se cierre si la conexión entre Activity y Service se realiza mal o no se tiene en cuenta que hay que reiniciarla cuando se genere una nueva instacia del Activity o la actual pase a segundo plano.

Durante el desarrollo me ha surgido una duda, ¿Mantener almacenada la configuración en local o hacer la petición cuando sea necesario?, finalmente desarrolle un sistema para

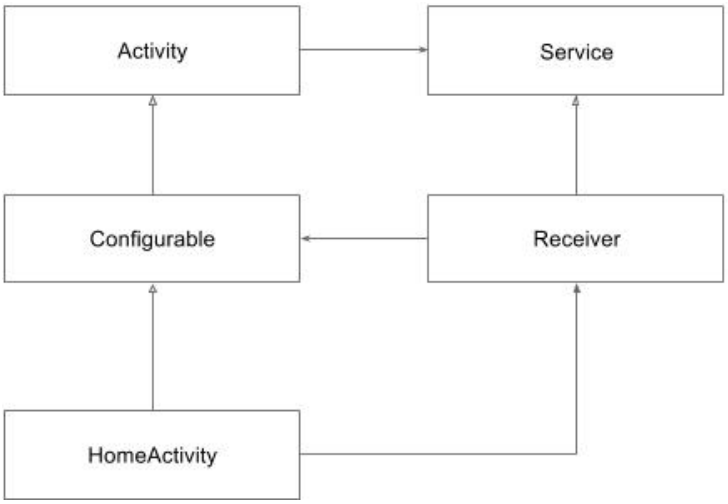

Figura 2.10: Estructura del Receiver

mantener la configuración almacenada en local usando la herramienta que proporciona Android, sin embargo, durante el proceso de desarrollo uso otra forma, siempre que se necesita la configuración se pide a servidor, esto lo hice así porque era para mi una manera simple de trabajar con el producto y estar seguro que siempre era la ultima configuración. Es este momento puedo decir que si la conexión a internet es buena y teniendo en cuenta que los datos pesan menos de 4KB[1], la diferencia entre almacenarlos o descárgalos es inapreciable.

2.5. Servidor

El lado del servidor ha sido desarrollado como un conjunto (Figura: 2.11) en el cual los elementos de su interior funcionan como elementos independientes conectados por las acciones que realiza el usuario, se han creado cuatro módulos. A continuación explicaré cada uno de ellos y como se relacionan con el resto.

El módulo con el que interactúa el usuario le hemos llamado **DeviceConfiguration** (Figura: 2.12), y es el encargado de gestionar las acciones directas que puede realizar el usuarios, es decir, registro de usuarios y navegación web. Para simplificar el uso de este módulo desde los otros puntos todas las funciones que realizan pueden ser accedida desde de la clase DeviceConfigurationService, de esta manera los otro módulos no necesitan conocer la estructura interna del módulo.

Dentro del módulo encontramos dos partes: La navegación web que usa el WebPage-Service, un servicio que es capaz de dado un objeto WebPage, por ejemplo, Index que esta

[1]Ver la sección 3.2.5 para entender el dato

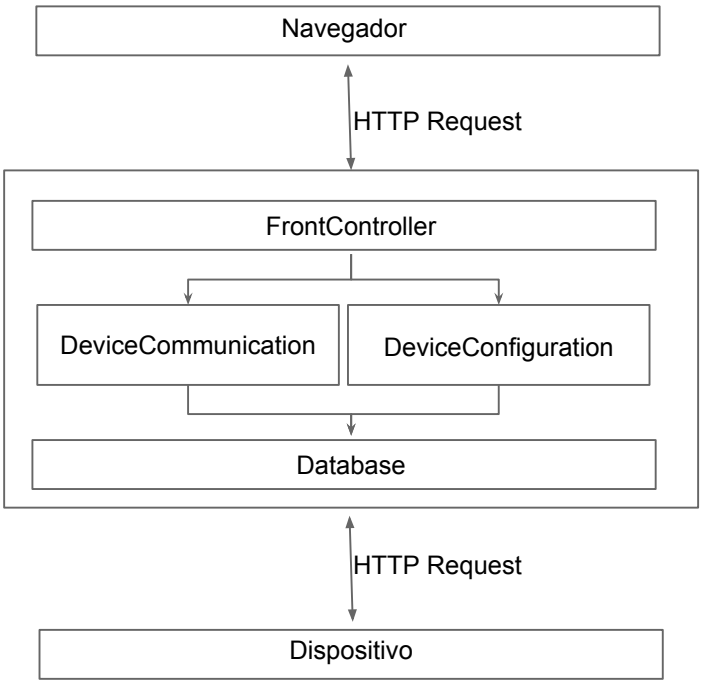

Figura 2.11: Estructura de la aplicación que se ejecuta en el servidor

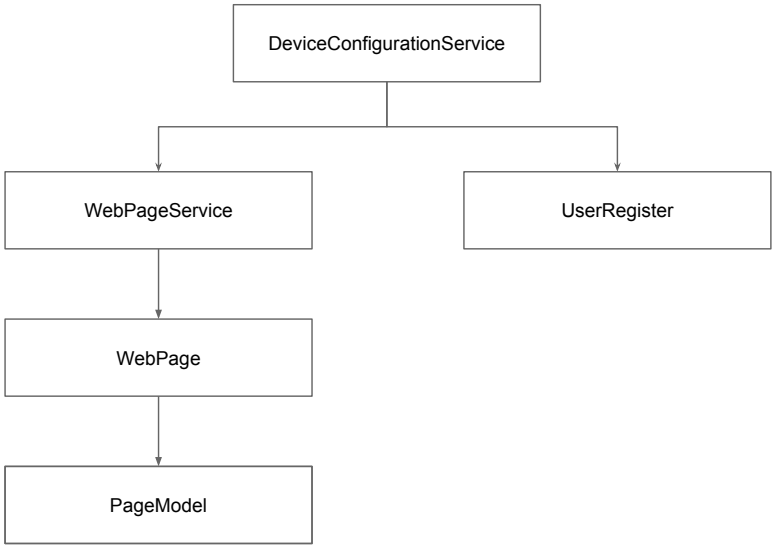

Figura 2.12: Diagrama de clases del servidor de configuración

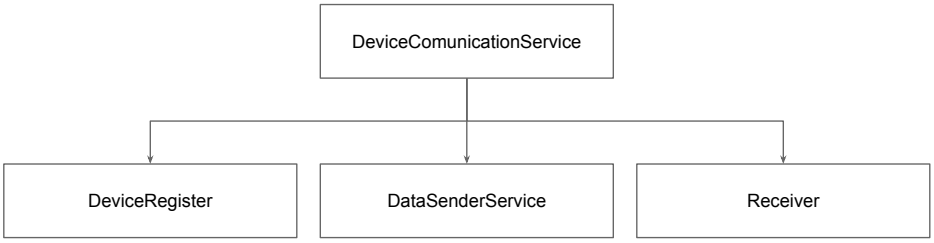

Figura 2.13: Diagrama de clases del servidor de dispositivos

asociado a una plantilla física (fichero) y a un PageModel, usar esos elementos para crear un ModelView que se devuelve al FrontController para que genere la respuesta HTML usando el Engine de renderizado correspondiente. La otra parte es el registro de usuarios, el UserRegister comprueba que los datos de usuario son validos para construir un objeto User y luego hacer uso de la interfaz del repositorio para almacenarlo.

Por otra parte, tenemos el módulo encargado de la comunicación con el dispositivo **DeviceCommunication**. El servidor de comunicación desempeña las funciones de enviar y recibir la información de los dispositivos además de su registro y actualización (Figura: 2.13).

Como en el módulo anterior existe una clase que engloba todas las funciones que este módulo exporta al exterior, DeviceCommunicationService es la clase que instancia el FrontController de la aplicación web y simplifica el uso de las siguientes funciones:

Registro de dispositivos, el DeviceRegister es un servicio que se encarga de chequear la validez de los datos antes de crear un objeto Device, asociarlo a un usuario y luego notificar al repositorio que se deben actualizar los datos de ese usuario.

Dado que en muchos caso el servidor necesita notificar alguna información al dispositivo, este modulo cuenta con el servicio DataSender encargado de generar los mensajes a partir de los datos que se quieren enviar y enviarlos al dispositivo usando la librería que proporciona el GSM, es importante entender que este servicio es solo de envío y totalmente asíncrono, cuando se requiere una respuesta del móvil lo que se hace es registrar un objeto Receiver en el DeviceCommunication, ese Receiver es en si mismo un pequeño servicio dedicado a gestionar la respuesta de una petición concreta que se genero en algún momento y que el DataSender a delegado en el DeviceCommunication

Como tercer módulo esta el encargado de gestionar la persistencia, que se exporta para ser usado por los dos anteriores mediante una clase Repository, en este módulo se incluye la implementación concreta del sistema de persistencia, incluye todas clases y librerías necesarias para gestionar la comunicación con la base de datos. Para almacenar

los datos se ha hecho uso de Hazelcast (sección: 3.2.6) que utiliza una base de datos MySQL, aunque durante el proceso de desarrollo se usa una base de datos basada en un fichero que almacena los datos en JSON puesto que nos permite ahorrar varios segundos en cada despliegue que hacemos en tiempo de desarrollo tiempo vital para que el avance del proyecto sea continuo y productivo.

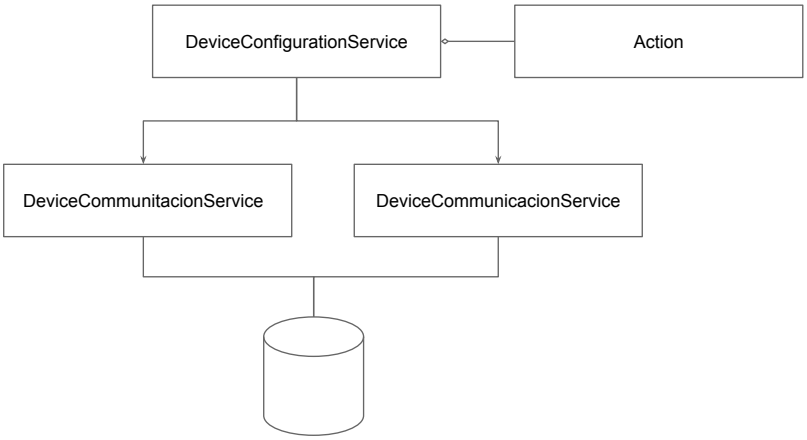

Figura 2.14: Diagrama de clases del servidor

Como ultimo módulo y en una de las facetas más importantes de una aplicación web tendríamos una clase que funciona como un Front Controller, se encarga de gestionar todas las URL y conoce las Action que debe ser lanzada cuando el usuario realiza la petición. (Figura: 2.14). Conoce la instancia en ejecución de los otros módulo y hace fluir los datos entre ellos, y el usuario final, es el encargado de lanzar los diferentes Engine que usamos para generar JSON o HTML según sea el tipo de respuesta de la acción, por tanto no es solo un punto de entrada si no también el encargado de recoger la información final que ha devuelto la acción para servirla al usuario.

Capítulo 3

El Proceso de desarrollo

3.1. Agile y Software Craftsmanship

> How it is done is as important as getting it done
>
> Sandro Mancuso

Desde hace algunos años es muy común oír hablar de metodologías ágiles, Agilismo, Scrum, y todo lo relacionado con el mundo Agile[1] en general. Yo desde hace algunos años conozco e intento practicas el Agilismo y desde hace menos tiempo me he fijado en el Software Craftsmanship [9] como ideología a seguir, los Craftsmen creemos en los principio que divulga el Agilismo pero aportamos una fuerte preocupación por como se hacen las cosas, lo importante no es solo hacerlo, si no hacerlo de la manera correcta. Entendemos que en el proceso de un proyecto software surgen imprevistos y el software siempre será propenso a fallos, por eso, abrazamos el cambios, nos acostumbramos y nos adaptamos a él.

Se trata de poner profesionalidad, pragmatismo y orgullo en lo que hacemos, no siento el desarrollo solo como un trabajo, así que, para mi era importante que eso se viera reflejado en el desarrollo de este trabajo. Dada la incapacidad de usar un equipo, y a pesar de contar con apoyo de los tutores el desarrollo ha sido hecho solo por mi, por lo tanto, no tenía sentido aplicar ninguna metodología tipo Scrum o eXtreme Programming(XP), sin embargo, además de estar altamente influenciado por la filosofía Lean [10] y la planificación iterativa de XP, además de sus valores y principios.

[1] http://agilemanifesto.org/iso/es/

3.2. Tecnologías usadas

En esta sección voy a introducir de manera breve las librerías de terceros, herramientas y framework que he utilizado en el desarrollo del proyecto.

3.2.1. Materialize

Framework para el desarrollo de Front-End basado en la linea de diseño ‘‘Material Design’’ [5] de Google. Su objetivo es acelerar el desarrollo, mejorar la experiencia de usuario y poder hacerlo en con un entorno fácil y cómodo. El framework aporte tres grandes bloques:

1. CSS, trae definido una rejilla (Grid) que permite maquetar de manera muy cómoda, además todo lo maquetado tiene un estilo por defecto.
2. Componentes, trae predefinidos unos componentes que son muy fáciles de añadir, por ejemplo, el ‘‘Preloader’’ permite de manera muy sencilla añadir una barra de carga o un spinner de carga. Además encontramos: colecciones, tarjetas, botones, formularios, iconos, etc.
3. JavaScript, Proporciona una librería JavaScript con gran número de funciones que permiten de manera sencilla añadir comportamiento a elementos del DOM, por ejemplo, los ‘‘Dialogs’’ permiten una manera muy sencilla y amigable de mostrar diálogos al usuario desde JavaScript.

3.2.2. Git: Gitflow

Git, es un sistema de control de versiones, se usa para guardar un historial de cambios a lo largo del tiempo, lo que permite tener un historial por el que se puede navegar y que contiene todas las versiones que tu hayas querido crear de tus ficheros, la explicación de esta tecnología se sale de los limites de este documento. Para el uso de este sistema existen varios workflows, yo he usado Gitflow [3], porque durante el desarrollo de mi vida profesional me he dado cuenta que es el sistema de trabajo que más cómodo me resulta y además te ayuda a llevar una disciplina. Para el uso de Git con la metodología de trabajo GitFlow he usado SourceTree.

3.2.3. Spark

Spark, es un Framework que esta inspirado en Sinatra[2], que usando las nuevas funciones de Java 8, nos proporciona una interfaz muy simple y potente para construir nuestra Rest API o web. Realmente genera una arquitectura similar al FrontController puesto que toda la gestión de las URL y el despacho de las funciones se realiza en un único punto de entrada.

3.2.4. Freemarker

Freemarker es un motor de plantillas para Java, y es muy cómodo de usar porque viene en el núcleo de Spark, de los motores de plantillas disponibles para Java integrados con Spark me he decidido por este por que era muy simple de aprender, además la forma en la que le inyecta los datos del modelo se adaptaba perfectamente a la parte del proyecto ya estaba desarrollada cuando se introdujo.

3.2.5. Google Clound Messaging

Es un servicio de google que funciona como un DNS (Domain Name Server) para dispositivos Android, permite enviar los datos del servidor al dispositivo con Android. Esto podría ser una notificación de inserción ligera diciendo la aplicación que hay nuevos datos que deben recoger del servidor (por ejemplo, el comando para leer apps), o podría ser un mensaje que contiene hasta 4 KB de datos de carga útil (la configuración del dispositivo).

3.2.6. Hazelcast

Trasladando la información publicada en el articulo ¿Qué es Hazelcast? [7]: Hazelcast es un Data Grid en Java, o dicho de otra forma una plataforma escalable para la distribución de datos.

Entre sus características más interesantes:

1. Implementaciones distribuidas de Set,List,Map,Lock,MultiMap
2. Mensajería distribuida P/S

[2] `http://www.sinatrarb.com/`

3. Soporte transaccional e integración JEE vía JCA
4. Soporte encriptación a nivel de sockets
5. Persistencia síncrona o asíncrona
6. Clusterizado Sesión HTTP
7. Discovery dinámico
8. Monitorización JMX
9. Escalado dinámico
10. Particionado dinámico
11. Fail-over dinámico
12. Modelo open-source con 2 versiones

El uso de Hazlecast implica almacenamiento en memoria RAM y la persistencia real es independiente de la librería en si, se puede usar una base de datos relacional, una no relacional o un repositorio de ficheros personalizado según las necesidades, si es que las hubiera de persistir los datos que maneja Hazelcast.

3.2.7. Universal Image Loader

Esta librería proporciona una forma potente, flexible y altamente personalízale para la carga, la muestra y el cacheo de imágenes en el sistema operativo Android. Es actualmente la librería más famosa de Android en GitHub y posee una interfaz de uso muy sencilla para una de las labores más complejas a las que se enfrenta un desarrollo en Android, la carga de imágenes.

3.2.8. Volley

Volley es una librería HTTP que hace de manera sencilla y lo que más importante rápida el uso de transacciones de red en Android, es un proyecto que se encuentra disponible en el repositorio del Android Open Source Project.

Volley ofrece los siguientes beneficios:

1. Gestión de la cola de peticiones de red
2. Conexiones HTTP múltiples y concurrentes de forma trasparente
3. Cacheo de respuestas HTTP
4. Soporte para la priorización de peticiones.
5. Alto nivel de personalización en las peticiones y las respuestas.

3.2.9. Tecnología Push

Se refiere a todas aquellas transacciones en las que la petición y el envió de información se genera en el servidor y fluye hasta el cliente, por ejemplo, cuando se lee de la base de datos la configuración de dispositivo se crea un mensaje y se le envía un dispositivo concreto, el origen de la comunicación se encuentra en el servidor.

3.3. Iteraciones

Durante todo el proyecto la metodología de trabajo ha sido simple, habiendo fijado desde un inicio cual era nuestro objetivo, luego, semana a semana hemos ido trazando un plan para conseguir el producto que buscábamos, en un principio tuvimos unas iteraciones más exploratorias, en las que buscábamos como conseguiríamos nuestro objetivos, tecnología adecuada, interacción, registro de usuarios, etc. Después aunque aún estábamos abiertos a cambiamos fijamos algunas historias que ya eran seguras y el flujo que consideramos adecuado y trabajamos semana a semana refinando los objetivos para conseguir un MVP.

3.3.1. Primera iteración

Durante este periodo la mayor parte del trabajo fue de investigación, había tres elementos importantes que debía saber antes de comenzar el desarrollo del proyecto, el primero era conocer la configuración que se debía establecer para que el sistema operativo supiera que la aplicación que yo estaba desarrollando era un Launcher, luego era importante conocer las herramientas que ofrece el sistema operativo para conocer las aplicaciones instaladas y la potencia de las mismas, y por ultimo explorar como podría mantener un servicio escuchando y ejecutándose en segundo planos sin que fuera perjudicial en el consumo de batería.

Dando respuesta a mis preguntas obtuve:

1. Establecer el Manifest para ser considerado un Launcher y no una aplicación corriente:

Listing 3.1: Configuración de un Launcher

```
<!-- AndroidManifest -->

        ...
    <activity

        android:name=".HomeActivity"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Wallpaper.NoTitleBar"
        android:launchMode="singleTask"
        android:stateNotNeeded="true"
        >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.HOME" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
    ...
```

2. El ‘‘PackageManager’’ es parte de la respuesta, descubrí que se distiguian dos tipos de aplicaciones, las extenar instaladas por el usuario a las que se hacia referencia por el package de Java y las Acciones que era aplicaciones que venían instaladas pero que no se ejecutaban usando un paquete.

Listing 3.2: Lanzar una aplicación o acción

```
        ...
        //Lanzar una app isntalada por el usuario
        Intent intent = activity.getPackageManager()
                                        .getLaunchIntentForPackage(appIdentifier);
        activity.startActivity(intent);
        //Lanzar una app que son referidas como Acciones
        activity.startActivity(new Intent(AcctionIdentifier));
        ...
```

3. Mantener un servicio escuchando y ejecutándose en segundo planos sin que fuera perjudicial en el consumo de batería, al final, fue un tema sencillo dado que Android proporciona unos eventos conocidos como ‘‘emisiones’’ a lo que se puede suscribir cualquier clase que cumpla la interfaz de ‘‘WakefulBroadcastReceiver’’

Después de descubrir los elementos descritos arriba, desarrolle un prototipo que venia a ser el proyecto mínimo para poder considerar que tenía un Launcher. Implementaba lo necesario para listar y ejecutar aplicaciones. Trabaje en elementos que sabía que iban a ser cruciales en el avance del proyecto, por ejemplo, que debíamos hacer cuando pasábamos a segundo plano y reutilizar celdas en el listado, que es importante en listado grandes y/o cambiantes.

3.3.2. Segunda iteración

Aún en una fase exploratoria del proyecto, modifique el prototipo anterior para que la interfaz se ajustará más a lo que buscaba e intentar concreta cual sería la aplicación objetivo y como iba a comportarse además su apariencia gráfica, en este apartado nacieron elementos como el reloj y la fecha en la pantalla principal.

El apartado a destacar de esta iteración es que aquí definimos la arquitectura actual de la aplicación (Figura: 3.1). Aunque realmente aún no existía la mayor parte de la funcionalidad, al crear el reloj y la fecha los identifique junto a la lista de aplicaciones como ‘‘Widgets’’ y entendí en ese momento que la configuración de los widgets no era responsabilidad del HomeActivity, así que identifiqué la necesidad de un Configurator, y en un primer acercamiento a la configuración remota, prepare un mock que simulaba leer la configuración remota.

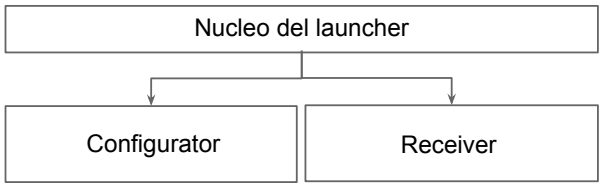

Figura 3.1: Estructura del Launcher

El resultado más importante de esta iteración a nivel de aplicación fue la estructura ya comentada en el capítulo de resultados (2):

- **Núcleo del Launcher:** El HomeActiviry es la instancia principal de la aplicación, nunca se finaliza, es le encargado de lanzar las aplicaciones y cuando se lanza alguna aplicación se queda en segundo plano. Además conoce a los otros dos módulos y funciona de enlace para hacer fluir los datos desde la información que se recibe del servidor hasta la interfaz que ve el usuario.
- **Configurator:** Dada una configuración es capaz de establecer como se deben ver y relacionar los elementos de la interfaz, se encarga de convertir en elementos visuales y accesibles por el usuario los datos que contiene la configuración que ha sido recibida desde el servidor.
- **Receiver:** Es un servicio que se encuentra dormido y que es despertado cuando se recibe información desde el servidor, es encargado de interpretar los diferentes comandos que se pueden recibir desde el servidor y realizar las operaciones para que esa información llegue al Núcleo.

Dado que todo el proyecto esta escrito en Java, durante esta iteración también definí los modelos (Figura: 3.2). Que son comunes a ambas partes del proyecto.

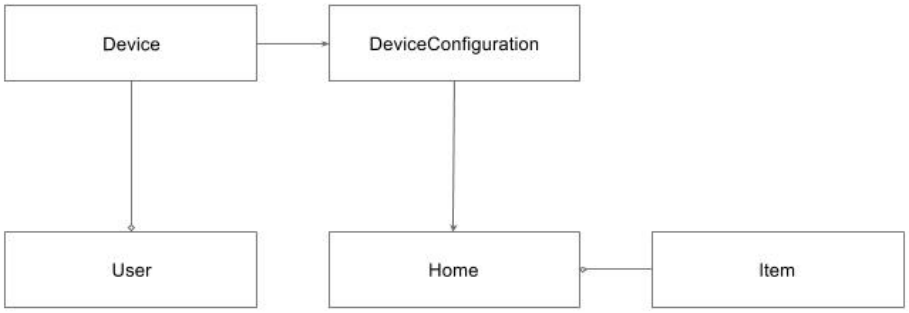

Figura 3.2: Diagrama de clases del modelo

Hemos modelado la aplicación como que un usuario que puede tener varios dispositivos, que a su vez poseen una configuración que en este momento solo incluye un Home, si se añaden pantallas en la aplicación habría que añadir pantallas en la configuración, el Home que representa la configuración de una pantalla principal lo podemos ver como un conjunto de items que alguna clase sabrá como deben ser dispuestos en la pantalla.

3.3.3. Tercera iteración

La primera parte de la iteración fue una tarea de indagación sobre los framework que había para desarrollar una web y una API REST en Java, en un principio comenzamos a usar Google Web Toolkit (GWT)[3], pero tenía demasiados componentes y una curva de aprendizaje compleja, no se ajustaba a nuestra necesidad. Después comenzamos un prototipo usando Restlet[4], con este framework tuvimos dos problemas principales, tiene un incontable número de dependencias, y como consecuencia pesa al rededor de 70 MB. Finalmente, llegamos a Spark (sección : 3.2.3), ligero, simple de usar, sin dependencias[5], tras realizar un pequeño ejemplo de uso, tomamos la decisión de desarrollar el proyecto usando este framework.

Desarrollando el primer prototipo de la web, me di cuenta que había invertido demasiado tiempo en un desarrollo que solo representaba un prototipo a nivel visual, así que sin llegar

[3] `http://www.gwtproject.org/`

[4] `http://restlet.com/`

[5] No requiere de ninguna librería, incluso es ‘‘autodesplegable’’ gracias a que incluye un servidor Jetty embebido

Figura 3.3: Mock-up de la pantalla que vería el usuario durante la configuración

a finalizar el prototipo, cree un prototipo pero usando el software de mock-up, Balsamiq, creé un mock-up de lo que sería la pantalla de configuración que tendría el usuario. Sobre prototipo (Figura: 3.3), más adelante veremos como al implementar llevarlo a practica surgieron cambios, algunos elementos fueron mantenidos, otros modificados y algunos eliminados.

En el proyecto, el dispositivo esta asociado a un usuario, la forma de hacer esa asociación es que el usuario acceda con su cuenta desde el dispositivo, así que durante esta iteración se creo y se introdujo la pantalla de Login, aún no sabía si las notificaciones Push iban a ir mediante Google Cloud Messaging, así que en esta primera iteración la lógica de esta pantalla se reducía a almacenar en local las credenciales, más adelante añadí la interacción con GCM y con mi servidor.

En esta iteración comenzó la investigación del uso de Google Cloud Messaging, primero dedique algunas horas a investigar las alternativas, y dado que es gratis y tiene librería para desarrollar en Java, me valía para el cliente y el servidor, así que escogí esta, para poder probar su funcionamiento desarrolle un proyecto aparte, este proyecto era una web muy pequeña que tenia un campo de texto y al escribir un mensaje, lo enviaba mediante push (sección: 3.2.9) a la aplicación del teléfono que mostraba una notificación cuyo texto era el mensaje enviado desde la web.

Al final de esta iteración tenía: la decisión de usar Spark, mock-up de la web, pantalla de login (app) y prototipo de uso del Google Cloud Messaging. Así que, aunque el incremento funcional propio del proyecto no era significativo con respecto a los anteriores, esta es la iteración que más ha influido en el flujo de uso de del producto y en el protocolo de comunicación entre aplicación y servidor.

3.3.4. Cuarta iteración

Esta es la primera iteración de la web, pero realmente solo se desarrollo a nivel visual integrandose con Spark y el motor de plantillas cual sería la pantalla de inicio. En este momento se tomo la decisión de usar Freemarker frente a otras opciones. Después usar It-rules, que aún se encontraba en fase de desarrollo, terminé una primera versión visual de la pantalla de inicio y cuando me decidí por comenzar a dividir en diferentes ficheros las plantillas que había obtenido, descubrí que aún no se encontraba implementada esa funcionalidad, sin embargo, había detectado que una parte muy buena de este motor de plantillas es que las plantillas eran claras y me había sido fácil empezar a usarlo. Pero no era posible, así que, una vez detectado que It-Rules no me ofrecía las funciones que yo había establecido como necesarias comencé una búsqueda de los motores de plantillas que se podían integrar de manera sencilla en Spark y encontré tres, Velocity, Mustache y Freemarker (actualmente han añadido unos cuantos más). Para tomar mi decisión me base en tres criterios: Curva de aprendizaje muy simple, Documentación amplia, y si en un futuro tenía que cambiar de motor de platillas no fuera necesario tocar la implementación de WebPageService ni de mi clase principal del servidor.

Los tres motores candidatos cumplían el ultimo criterio, sin embargo, tras realizar tres plantillas iguales una con cada uno de ellos, desde mi punto de vista me fue mucho más simple comenzar a usar Freemarker. Por ultimo el apartado de la documentación que era muy importante, dado que no tenia siempre disponible a alguien que me pudiera orientar si me quedaba atascado, por lo que necesitaba que la comunidad y la documentación me ofrecieran buenas soluciones, es este apartado considero que Velocity es más conocido y tiene mejor documentación y comunidad pero Freemarker tiene una única web con toda la documentación bien catalogada, y puesto que me había sido más simple crear el ejemplo usando Freemarker, opte por usar Freemarker.

En esta iteración también se dejó preparados los elementos para lo que luego se transformaría en el ‘‘DeviceConfiguration’’. Desde el lado del servidor, se podrían identificar dos fases en el desarrollo de esta iteración:

La segunda parte de la iteración consistió en trasladar el ejemplo que había desarrollado en la iteración anterior sobre GCM e incluirlo en mi proyecto, así que empecé a desarrollar

ese ejemplo en el que la web manda un mensaje al launcher y este lo muestra como una notificación, no fue complicado adaptar el ejemplo a el proyecto que ya tenía.

En la aplicación además de realizar el ejemplo ya mencionado, desarrolle dos apartados muy importantes, dentro del módulo del Configurator, se desarrollo el apartado que actualizaba la configuración cuando se le inyectaba una nueva configuración en tiempo de ejecución. Y como consecuencia del desarrollo del ejemplo con algunas modificaciones se creo lo que luego paso a ser la clase encargada de recibir que ha habido un mensaje desde el servidor y lanzar un hilo que se encargue de tratarlo.

De nuevo, dado que el trabajo de investigación se fue desarrollando a la par que se programan las diferentes funcionalidades del software, en algunas iteraciones notamos que el incremente funcional no es significativo, pero lo importante es la cantidad de información que somos capaces de obtener para luego tomar una decisión que seguramente tendrá más robustez y va a proporcionar mucho más valor al proyecto después, ya que, teníamos el conocimiento necesario para tomarla de forma correcta.

3.3.5. Quinta iteración

Esta iteración se podría considerar la mitad del proceso de desarrollo y un hito importante fue unir por primera vez a través del servidor la aplicación móvil con la web, desde el lado del servidor la iteración comenzó añadiendo Hazelcast (sección: 3.2.6) para poder tener algún sistema de persistencia aunque los datos se perdieran al reiniciar el servidor. Pensando en la modularidad de la aplicación para añadir el sistema de persistencia creé un módulo aparte dentro de mi proyecto IntelliJ y desarrolle en primer lugar una interfaz ‘‘Repository’’ que representaba las consultas que iba a necesitar desde los módulos de comunicación y configuración. Seguidamente cree una implementación de dicho repositorio usando Hazelcast y cree el DeviceRegister y el UserRegister dentro de los módulos DeviceCommunication y DeviceConfiguration respectivamente, prepare ambas clases para que se les inyectará la interfaz del repositorio desde el controlador principal que gestiona el arranque de la aplicación.

Una vez desarrollada la persistencia pase a completar el DeviceConfiguration, dentro de este añadí el UserRegister y el DeviceConfigurationService para poder interactuar desde las acciones, tras escribir una serie de test para comprobar que la colaboración entre el UserRegister y la persistencia sucedían según lo esperado, di por terminado el desarrollo de Backend y pase a desarrolla el formulario de registro y Login e incluirlos en la versión ya terminada del index desarrollada en Freemarker, hasta este momento y con solo una página que no variaba en función del usuario no veía necesario crear el WebPageService,

así que el FrontController era el encarga de realizar las tareas que luego se trasladaron a otro servicio.

Cuando completé lo descrito en el párrafo anterior comencé a desarrollar la lógica detrás de la aplicación y puesto que en la iteración anterior añadí al proyecto la interacción con GSM lo primero que hice fue dotar de lógica completa a la pantalla de Login, para que primero se registrará en el GSM luego enviará el token único a mi servidor junto al resto de datos (sección: 2.3), tras implementar esto hice la primera prueba de integración que me permitía, registrarme en la web y luego hacer login desde la web o la aplicación.

Llegados a este punto aparece el concepto de dispositivo en el servidor, aunque ya esta modelado, hasta ahora no se había usado para nada, así que lo primero que hago es crear el servicio DeviceRegister que pertenece al módulo de comunicación y se encarga en un inicio de recoger los datos que el controlador principal le envía después de recibir un login desde un dispositivo nuevo. El DeviceRegister se implemento y se probo que la interacción con el repositorio era la esperada.

Como ya se había incluido el ejemplo del GSM en el controlador principal, lo que se hizo fue crear una clase DataSender que se incluía dentro del módulo DeviceCommunication que es la que es capaz de enviar mensajes a los dispositivos.

El ultimo paso de esta iteración fue que cuando el usuario accedía en la web podía ver una lista de los dispositivos con los que había hecho login y al pulsar se enviaba un mensaje al dispositivo que aunque recibía gracias a la clase que se incluyó y que provenía del ejemplo del GSM no hacia nada con los datos recibido.

Esta vez creo que se puede destacar el alto incremento funcional que sufre la aplicación en esta iteración, al final de esta iteración tenemos: Registro desde la web, login desde el dispositivo y la web, comunicación HTTP y GSM entre el servidor y el dispositivo. Este repentino incremento se debe sobretodo a que en fases previas se trabajo muy bien la base de lo que son cada una de las partes y con un conocimiento robusto e ideas muy claras el avance funcional del proyecto a partir de esta iteración es siempre muy significativo.

3.3.6. Sexta iteración

Esta iteración fue dedicada por completo al servidor, se termino la maquetación de inicio de la página lo que sería el index usando las imágenes y textos que se han mantenido hasta la actualidad.

Tras terminar el index, el siguiente paso era crear el home, que es la página que ve el usuario una vez ha accedido, lo primero que hice fue utilizar el mock-up que ya

había creado en la iteración cuatro para hacerme una idea de como iban a ir ubicados los elementos. A partir de ahí usando el que hasta entonces era mi framework para frontend, Twitter Bootstrap [11], desarrolle toda la maquetación del layout con HTML y Freemarker. Una vez había desarrollado la parte visual, para hacer funcionar la web necesitaba crear un modelo de datos que luego era inyectado en la plantilla para generar un ModelView que es el objeto que el motor de platillas puede renderizar. En este momento aunque ya me percaté de que el FrontController estaba realizando funciones que no le correspondían, puesto que generar un modelo de datos que representará la información que contenía la web no era una tarea que le perteneciera a él. Me limite a hacer que la creación de la página con diferentes datos según el usuario que había hecho Login funcionará.

Una vez que lo anterior funciono pase a solventar el problema de responsabilidades que había detectado, el primer paso fue modelar una página usando java, para mi, una página es: Una plantilla que la reconozco por el nombre de fichero y un PageModel, el PageModel es un objeto que representa la información de manera que desde Freemarker sea muy sencilla trabajar con ella, para mi el objeto página al que llame Page pertenece a un enumerado en el que se encuentran listadas las páginas que tiene la web, de esta manera tenemos una instancia de Page que se llama Index y que sabe que su plantilla es el ‘‘index.flt’’ y que su modelo de datos es el IndexPageModel.

Usando lo anterior conseguí crear una página que según el usuario que hacía Login mostraba unos dispositivos u otros, hasta entonces simple mostraba los mismo dispositivos indistintamente del usuario. Una vez tenía los dispositivos listados en la web el siguiente paso era dar la posibilidad al usuario de comenzar a interactuar con ellos, así que, use el sistema de login de la iteración anterior para comenzar a añadir de manera dinámica dispositivos al usuario y permitirle interactuar con el aunque la información que se enviaba al dispositivo aún no se usaba para nada.

Lo primero fue implementar del lado del servidor una URL que dado un número de dispositivo, si el usuario posee un dispositivo con ese número devolvía su configuración, en ese momento la configuración era falsa, generada a mano. Luego del lado del cliente había un código JavaScript que cuando recibe una configuración es capaz de generar las aplicaciones que tiene el dispositivo activas y una vista previa para que el usuario empiece a gestionar las aplicaciones y contactos. También del lado del cliente en esta iteración se desarrollo el código necesario para que el cliente pudiera eliminar aplicaciones y contactos pero solo de una manera falsa dado que estos datos no se persistían en el servidor, otro elemento que se añadió en esta iteración fue el botón que más tarde será eliminado ‘‘Enviar Configuración’’ y que en ese momento su labor era enviar una petición al servidor diciendo que enviará un mensaje al dispositivo.

Durante esta iteración se trato de trabajar mucho en el diseño de iteración y de interfaz y así fue que durante el desarrollo de la pantalla de home de la web, detectamos que el uso no era intuitivo puesto que el usuario necesita:

1. Seleccionar el dispositivo a la izquierda del todo
2. Modificar la configuración en el extremo derecho
3. Por ultimo los resultados se mostraban en la vista previa que estaba aproximadamente en la mitad de la pantalla.

Para mejorar este flujo alternamos la posición de la vista previa con la zona de configuración, más adelante veremos que luego se opto por usar otro framework de frontend y se entrará mas en detalle.

3.3.7. Séptima iteración

Como continuación de el código de cliente desarrollado en la iteración anterior que permitía eliminar contactos y aplicaciones, desarrollamos una acción en el servidor que se encarga de eliminar items de un dispositivo y por supuesto desarrollamos una que permitía añadir aplicaciones y otra que permitía añadir contactos, mientras que para borrar vale la misma petición para crear por simplicidad se han creado dos acciones, así que se paso a conectar el código de cliente ya desarrollado con el nuevo código de servidor y añadir el código nuevo en el cliente.

En la retrospectiva de la iteración anterior mencioné que habíamos notado que la experiencia y el diseño de usuario no se estaban aproximando a lo que buscábamos, empezamos por cambiar por completo el diseño, cambiamos de framework de frontend y comencé a usar Materialize (sección: 3.2.1), lo bueno es que los colores, las formas y la fuentes y los componentes de este framework están basados en Android por tanto obtuvimos una mejora incalculable en la aproximación al diseño de interfaz objetivo. Realmente notamos que uno de los puntos flojos es que no era intuitivo, la información no fluía de manera natural, por ejemplo, era necesario enviar la configuración para que llegará al dispositivo, cuando ya estaba almacenada en el servidor. Eso creaba momentos de inconsistencia, es decir, lo que veía el usuario de la web y lo que veía el del dispositivo era distinto hasta que se e daba al botón enviar en la web, se elimino ese obstáculo y se paso a realizar el envió automático de la configuración siempre que se hiciera un cambio.

Para terminar de dar un aspecto fluido a la web se alternaron las posiciones de la vista previa y y la lista de aplicaciones activas, por tanto, y como dije en el capítulo de

resultados 2.2 con dos cambios muy sencillos se paso de una interfaz que no trasmitía a una interfaz en la que los elementos surgen al ritmo de trabajo del usuario.

Desde lado del terminal ya se habían desarrollado elementos importantes, se recibían mensajes del GCM, se cambiaba en tiempo de ejecución la configuración y se había establecido comunicación entre la instancia principal y los servicios en segundo plano. Era el momento de utilizar todas esas piezas para construir nuestro puzzle, así que, usando las peticiones que se han descrito en los párrafos anteriores de esta iteración hicimos que el dispositivo comenzará a hacer uso de los datos que recibía.

El primer paso para comenzar a hacer uso de esos datos fue hacer que la clase que era la que se despertaba cuando se recibía un mensaje de GCM capturará el mensaje y despertará a mi servicio inyectándole los datos provenientes del GCM. Una vez que mi servicio estaba en ejecución realizaba dos pasos sencillos, primero deserializa los datos transformándolos en un objeto SeralizeDeviceConfiguration, y segundo usa la conexión que existe entre el módulo Receiver y el Núcleo para enviar al método update del Configurable los nuevos datos. Cundo el Configurable detecta que se le ha realizado un update, notifica a su Configurator de que los datos han cambiado. Recordemos que el sistema para actualizar la configuración en tiempo de ejecución ya estaba desarrollado así que hemos conseguido hacer fluir datos desde el el usuario de la web a la interfaz del dispositivo.

En el incremento funcional hasta ahora tenemos: Registro, Login (Aplicación y Web), añadir aplicaciones y contactos, enviar configuración y aplicar configuración en tiempo de ejecución.

3.3.8. Octava iteración

Para que el producto mínimo viable objetivo represente de verdad lo que sería un producto final. Era indispensable que el servidor fuera capaz de leer las aplicaciones instaladas en el dispositivo para que el usuario de la web cuando va a añadir una aplicación añada una que este instalada. Por otra parte era importante que al añadir un contacto o aplicación fuera posible personalizar la foto que aparece en el dispositivo.

Primero desarrollé el sistema para leer las aplicaciones, esto se hace bajo demanda, es decir, cuando un usuario selecciona un dispositivo desde el navegador realizo una petición al servidor, donde genero un contexto asíncrono asociado a la petición del usuario y creo un cliente de mensaje Push al que le proporciono ese contexto. A partir de ese momento realizo dos acciones en paralelo por un lado envió una respuesta parcial a la web usando el cliente Push para informar que se ha recibido la solicitud de leer las aplicaciones y por

otra parte se envía usando GCM un mensaje al dispositivo que dice ‘‘ReadApps’’ en ese mensaje no se incluye ninguna otra información, sin embargo, como sistema de verificación el dispositivo en el mensaje de respuesta incluye: el número que lo identifica y a que usuario pertenece y por supuesto una lista de las aplicaciones instaladas, para recuperar el cliente Push que aún esta almacenado en memoria uso el número del dispositivo y el correo el que dispositivo envía y si existe un cliente Push, es que, el cliente web aún esta pendiente de una respuesta con la información del dispositivo, así que, uso el cliente Push para enviar los datos recibidos desde el dispositivo al navegador web y elimino el cliente Push.

Como ultimo elemento funcional que se añade al proyecto tenemos la subida de imágenes, que implica dos pasos, primero se sube una imagen desde el cliente al servidor y el servidor la almacena y devuelve la URL temporal, luego en el cliente se pide al usuario que la recorte, una vez el usuario que la recorta y verifica que esa es la imagen final en el cliente, el servidor la almacena usando un identificador único e informa al cliente de cual es el identificador, cuando el usuario pulsa por ejemplo crear contacto el navegador genera una petición en la que envía nombre del contacto, número e id único de la imagen a la que esta asociado, esta misma información se envía al dispositivo que conoce que todas las imágenes se encuentran disponibles en ‘‘/img/uniqid.format’’ una acción del servidor que perite leer las imágenes.

Llegados a este punto, como retrospectiva general de todo el proceso de desarrollo destacaría el continuo avanza paralelo de ambas partes y la toma de decisiones siempre basadas en experiencias e investigaciones o sobre un punto de pivote que estaba verificado, por ejemplo, es un hecho que la mejora de la usabilidad siempre es un añadido a cualquier desarrollo de software.

Capítulo 4

Aportaciones, conclusiones y trabajo futuro

En este proyecto se ha abordado una necesidad que ya había identificado previamente. Esta necesidad es la de poder configurar remotamente un dispositivo móvil, pudiendo incluso modificar lo que aparece en la pantalla principal del mismo. Esta necesidad atiende a problemas a los que, por ejemplo, las empresas se enfrentan al proporcionar dispositivos móviles a sus trabajadores para que éstos puedan desempeñar su trabajo. Esto puede implicar, primero, una dificultad añadida para los trabajadores a la hora de desempeñar su labor (curva de aprendizaje compleja) y, segundo, que se haga un uso responsable del dispositivo.

Por otra parte, también he identificado que otro de los sectores más castigados a la hora aprovechar el potencial de las nuevas tecnologías es el colectivo de personas mayores. Pese a posean interés por el uso de estas tecnologías, la dificultad a la hora de usar estos dispositivos hace que en muchos caso conlleve a la frustración o el rechazo.

Atendiendo a los nichos de mercado arriba descritos, se ha creado una aplicación que permite que la pantalla principal del teléfono se configure de manera remota a través de una web. Esta aplicación se nutre de la libertad que proporciona Android para permitir que la pantalla de principal se pueda configurar remotamente, sin necesidad de estar junto al terminal. Además, se ha tratado de diseñar una pantalla principal muy sencilla con el objetivo de eliminar aquello que no es útil según el criterio que el encargado de configurarla considere.

Para asegurar la continuidad del proyecto, actualmente se encuentra publicado en BitBucket[1], bajo la licencia GPL [4], para que pueda ser reutilizado como base para

[1] `https://bitbucket.org/`

otros proyectos o descargado y desplegado como un servicio. Después de dos años como miembro activo de una comunidad de desarrolladores, he llegado a la conclusión de que compartiendo conocimiento es como se genera conocimiento, es por eso por lo que he tomado la decisión de publicar mi proyecto y así aportar un grano de arena a la montaña que hoy en día supone el Software Libre.

De los dos nichos de mercado planteados inicialmente, el orientado a empresas, es quizás el que primero ha mostrado su validez, puesto que durante el desarrollo del proyecto, una empresa, mostró interés en la tecnología que el proyecto contiene.

Uno de los puntos fuertes de nuestro proyecto es la configuración remota. Además, es importante señalar lo sencillo que es comenzar a usar el producto: desde que se conoce el proyecto hasta que está en funcionamiento, sólo sería necesario seguir los siguientes pasos:

1. Crear una cuenta en la web.
2. Descargar el Launcher en el dispositivo, ejecutarlo y acceder con una cuenta de usuario.
3. Ir a la web y seleccionar el dispositivo para comenzar a trabajar con él.

Se ha trabajado bastante en la mejora de la usabilidad. Un ejemplo de ello, es que no es necesario enviar la configuración mediante un botón, pues cada vez que se detecta un cambio, el sistema envía la configuración al terminal y este se actualiza. Este cambio se introdujo como resultado de pivotar sobre con la premisa en mente de encontrar un producto intuitivo.

La mejor manera de entender el valor de lo obtenido es representar un escenario en el que hasta ahora no se usaba y luego introducir nuestro producto y ver como ayuda.

Desde el punto de vista profesional el aporte es claro, no solo tenemos un proyecto de software, sino que actualmente hay una empresa continuando este proyecto. Por tanto, en ese aspecto, se ha creado un proyecto de software libre que, además de contribuir al crecimiento de la comunidad y a la difusión del conocimiento, ha servido para el desarrollo profesional de una empresa. Lo que significa que es útil y que se ajusta a las necesidad que surgen en un sector determinado.

Para mi como profesional, este proyecto ha supuesto el resultado de unos meses de continuo aprendizaje y esfuerzo que se han traducido en un resultado que no es medible. No obstante, la realización de un proyecto de fin de título, que se evalúa de forma individual, fomenta un trabajo en solitario y hace prácticamente imposible el desarrollo de un software

en equipo. Lo ideal es que un software se nutra de las aportaciones de varias formas de pensar, para añadir la máxima calidad y valor al producto. En este proyecto, a diferencia de mi experiencia profesional, he tenido que realizar el trabajo en solitario, y a pesar de la buena predisposición de los tutores, considero que la calidad final habría sido mucho mayor si se hubiera desarrollado en equipo con otros proyectantes.

Aún así me gustaría recalcar que durante el desarrollo aprendí de manera práctica una cantidad de tecnologías y métodos, que aunque algunos los conocía de manera teórica, aplicarlos en un proyecto implica una madurez profesional que solo se adquiere trabajando. Si tuviera que destacar un elemento concreto del gran aporte que he recibido hablaría de la gestión del tiempo y como conforme avanzaba el proyecto era capaz de medir mejor lo que me iba a llevar una tarea y tenía mejor capacidad para dividir las tareas en un esfuerzo asumible.

Tras el desarrollo de este proyecto he afianzado aún más mi creencia en que el desarrollo de software debe guiarse de un modo iterativo que iteración a iteración se muestre al cliente, el estado del proyecto, recibir un feedback, adaptar el proyecto y sus características para al final obtener un producto que hace justo lo necesario y que lo hace de la manera que el cliente cree necesaria.

Aunque el proyecto desarrollado supone un producto completo y funcional, no es un producto para ser explotado en el mercado, es un Producto Mínimo Viable que siguiendo la metodología Lean usaremos para validar la capacidad de cubrir las necesidades de los nichos que hemos detectado. Es por esto que si el producto llegará a demostrar su valía tenemos gran cantidad de funcionalidades y aspecto que supondrían un incremento muy bueno, con la liberación del proyecto esperamos que el crecimiento sea continuo y el producto actual y la idea en la que se basa no se queden solo en el marco de este trabajo, dentro de las funcionalidades que hemos planteado como mejora para futuro destacaría:

- Operaciones en masa, sería positivo sobre todo para las empresas tener la posibilidad de realizar una configuración que fuera aplicable de manera automática a todos los dispositivos de la empresa.
- Agrupar, ordenar, etiquetar, actualmente se identifica los dispositivos mediante su teléfono o con un nombre, propongo extender esta funcionalidad y tener la posibilidad de añadir un sistema para agrupar dispositivos, por etiquetas o algún otro sistema y poder ordenar como se muestran en el menú de la izquierda.
- Obtener los logs (registros) del sistema, supongamos que estamos ejecutando una aplicación concreta que puede ser propiedad de la empresa o quizás es una aplicación de seguimiento por GPS, sería interesante tener la posibilidad de poder visualizar

o descargar esos logs, podríamos medir también elementos estadísticos o cualquier otro elemento que se considere importante.

- Control de Volumen, Brillo, seguimiento de la batería, finalmente este apartado no llego a entrar en el producto actual y es un apartado importante sobre todo la batería que nos puede dar información que nos permitiría por ejemplo avisar al usuario de que cargue la batería o reducir algunas funciones hasta que la batería este por encima de un umbral.
- Para mejorar la potencia de la interfaz sería un gran valor dar al usuario de manipular la imagen que acaba de subir para que quede como el espera.
- Tener la capacidad de instalar aplicaciones de manera remota usando algún gestor de paquetes o mandado un fichero de ejecución comprimido.
- También existen funciones que se pensaron y se introdujeron el producto actual como el envío de mensajes pero que luego se retiraron y es importante volverlas a añadir.
-

Yo personalmente y en un futuro cercano pretendo extraer del proyecto actual una librería que ofrezca las funciones que un Launcher necesita, como por ejemplo, la función para filtrar las aplicaciones, la capacidad de lanzar cualquier aplicación, etc. Son funciones que he identificado durante el desarrollo del proyecto y que considero que pueden ser de utilidad para el desarrollo de otros proyectos.

Bibliografía

[1] Blé, C. *Diseño Ágil con TDD.* www.carlosble.com, 2010.

[2] Cid, M. Android vuelve a superar el 90 % de cuota de ventas en españa mientras blackberry toca suelo. *http://www.xatakamovil.com/* (2014).

[3] Driessen, V. A successful git branching model. *http://nvie.com* (2010).

[4] Foundation, F. S. Licencia pública general de gnu (gpl). *http://www.gnu.org* (2015).

[5] Google. Material design. *http://www.google.com/design/spec/material-design/introduction.html* (2014).

[6] Google. Dashboards. *https://developer.android.com/* (2015).

[7] Gracia, L. M. ¿qué es hazelcast? *http://unpocodejava.wordpress.com* (2013).

[8] Guillermo. Fondo negro en el smartphone: ¿sirve para ahorrar batería? *http://tecnovortex.com/* (2014).

[9] Mancuso, S. *Software Craftsmanship.* Prentice Hall, 2015.

[10] Maurya, A. *Running Lean: Cómo iterar de un plan A a un plan que funciona.* UNIR Emprende, 2014.

[11] Sanchez, A. F. ¿qué es bootstrap? *http://openwebcms.es* (2013).

[12] The Apache Software Foundation . Apache tomcat. *http://tomcat.apache.org/* (1999-2015).