Grado en Ingeniería Informática

Trabajo Final de Grado

CoolAntivirus Anti-Theft, aplicación Android destinada a la protección del smartphone en caso de robo o extravío

Autor:
Víctor Felip Andreu

Supervisor:
Gerardo Prieto
Tutor académico:
Vicente Cholvi Juan

Fecha de lectura: 29 de Octubre de 2015
Curso académico 2014/2015

Agradecimientos

Me gustaría expresar agradecimiento a todas las personas que fueron de gran ayuda para el desarrollo de este proyecto de fin de carrera.

Comenzando por mi supervisor en las prácticas, Gerardo Prieto, quien me proporcionó los materiales necesarios para comenzar el desarrollo y me ayudó en la especificación de los objetivos. También he de agradecer al resto de compañeros de *Electronic Commerce Factory SL* por haberme prestado materiales y consejos durante el desarrollo, en especial a Álvaro y a Sam, quienes me facilitaron el acceso a todo lo que necesitaba y compartieron sus experiencias conmigo para ayudarme en varias partes del proyecto, gracias.

A mis padres y mi hermana por su apoyo incondicional y su confianza.

Finalmente, a mi tutor Vicente Cholvi Juan, por hacer siempre un hueco para atenderme e indicarme como proceder.

Resumen

Este proyecto surge del interés de la compañía startup *Electronic Commerce Factory SL* de aumentar la funcionalidad de *CoolAntivirus*, su antivirus para móviles Android. Esta aplicación consiste en un conjunto de módulos específicos e independientes que tienen funcionalidades relacionadas con el mantenimiento y protección del dispositivo.

El modulo creado en este proyecto, *Anti-Theft*, implementa diversas opciones para permitir al usuario determinar o realizar diferentes acciones sobre el smartphone en caso de robo o extravío. Algunas de estas funcionalidades se gestionan vía Internet mediante una aplicación web que permite al usuario conectar con el dispositivo.

Como resultado de este proyecto se ha obtenido la aplicación independiente correspondiente al modulo *Anti-Theft* con todas las funcionalidades previstas y la aplicación web que gestiona algunas de sus funciones. Sin embargo, no se llegó a la fase de la integración con *CoolAntivirus* en el plazo de la estancia en la compañía.

Palabras clave

Aplicación, smartphone, Android, anti-robo, protección.

Keywords

Application, smartphone, Android, anti-theft, protection.

Índice general

Capítulo 1

Introducción

1.1. Motivación del proyecto

La motivación del proyecto consiste en ampliar la funcionalidad de una aplicación para teléfonos móviles desarrollada por *Electronic Commerce Factory SL* (www.e-commfactory.com) dedicada a la protección y mantenimiento de los datos del teléfono. La funcionalidad que se desea añadir permitirá proteger los datos de los usuarios ante posibles pérdidas o robos del dispositivo.

Se desea que el usuario pueda realizar acciones que impidan el robo de datos tanto antes como después de la pérdida del teléfono. De esta forma, aunque haya perdido su teléfono, podrá realizar varias acciones al respecto.

El proyecto desarrollado tratado en esta memoria se divide en dos partes, una aplicación para móviles *Android* (Ver Apéndice A.1) y una aplicación web. Ambas están interconectadas de tal forma que desde la aplicación web se deben poder realizar ciertas configuraciones en la aplicación.

1.2. Objetivos del proyecto

1.2.1. Objetivos generales

Como se ha mencionado anteriormente, el principal objetivo es el diseño de una aplicación para móviles Android que:

- Permita al usuario configurar opciones para proteger su teléfono.
- Notifique al usuario en caso de robo o extravío.
- Pueda recibir notificaciones para poder realizar acciones sobre el teléfono de forma remota.

1.2.2. Objetivos específicos

- Las tareas de protección del teléfono se ejecutarán en segundo plano.
- La aplicación se mantendrá comprobando el estado del teléfono continuamente.
- Las notificaciones se enviarán y recibirán por wifi.
- La localización del teléfono se realizará exclusivamente por GPS.
- El usuario establecerá una contraseña con la que protegerá el dispositivo en caso de robo o extravío.
- El usuario se registrará en la aplicación en el primer uso para tener acceso a la página web de administración.

1.3. Tecnologías empleadas

Al inicio del desarrollo del proyecto ya se sabe que se iba a programar la aplicación para móviles Android, por lo que se decide utilizar *Java*. Éste es el lenguaje de programación nativo de *Android*, que tiene mayor soporte por parte de *Google*.

Utilizar *Java* interesa a la hora de desarrollar la aplicación porque se van a utilizar una serie de servicios de la API de *Google Play Services* para las tareas de localización y notificaciones que se tratarán en el apartado 4.

Las interfaces gráficas de la aplicación móvil y de la aplicación web han sido proporcionadas por la empresa y están escritas en lenguaje *HTML* y *CSS*. Para enlazar esas interfaces con la aplicación que se desarrolla en este proyecto se utiliza *JavaScript* y *Jquery* de la forma que se explicará en el apartado 4.1.1.

Capítulo 2

Planificación del proyecto

En éste capitulo se tratará la preparación del proyecto, comenzando por la estimación de recursos necesarios para llevarlo a cabo. Después se estudiará la propuesta técnica que se hizo del proyecto junto con la planificación temporal del mismo, observando los cambios entre la planificación inicial respecto a la final.

2.1. Estimación de recursos del proyecto

Una vez realizado el análisis de objetivos se procede a la estimación de recursos. El desarrollo de la aplicación necesita de un ordenador con acceso a Internet para desarrollar las aplicaciones. Para realizar las pruebas se deben utilizar diferentes modelos de smartphones. También se necesita llevar un control de versiones y copias de seguridad del proyecto, por ello se decide de antemano utilizar un repositorio en la nube gratuito.

En cuanto al software para el desarrollo de la aplicación móvil, se decide utilizar el IDE (Integrated Development Environment) que proporciona *Google* para el desarrollo de aplicaciones en *Android*, el *Android Studio* (Ver Apéndice B.1).

Lo recursos necesarios par el proyecto se enumeran a continuación:

- Ordenador para desarrollar la aplicación.
- Smartphones para realizar las pruebas.
- Software gratuito *Android Studio*.
- Tiempo del autor del proyecto para formarse en el desarrollo de aplicaciones para Android y sobre la utilización del Android Studio.
- Repositorio alojado en *Bitbucket* (bitbucket.org), host de repositorios gratuito.
- Tiempo del autor del proyecto para el desarrollo del proyecto.

Una vez conocidos los recursos necesarios, se estudian los costes monetarios de los mismos. La empresa proporciona un ordenador desde su inventario para el desarrollo de la aplicación al autor del proyecto, por lo que no requiere ningún coste de compra. Lo mismo ocurre con los smartphones que se utilizarían en las pruebas, que fueron proporcionados por compañeros de la empresa y por el propio realizador del proyecto. Finalmente, todo el software utilizado para el desarrollo y alojamiento de repositorios tampoco supone ningún coste para la compañía al ser gratuitos.

2.2. Metodologia y planificación

Una vez especificados los objetivos generales del apartado 1.2.1, se asigna un tiempo aproximado para la documentación y formación necesaria para cada una de las tareas. La especificación de las fases y la planificación inicial se realiza mediante un diagrama de Gantt (Figura2.1). En éste se ve el reparto de las tareas entre las semanas de la estancia. En la figura 2.2 se observa el reparto de horas entre las tareas.

La planificación se siguió correctamente hasta la parte de la implementación. El análisis del código de las interfaces gráficas requirió menos tiempo del estimado para su comprensión. Sin embargo, se necesitó una mayor cantidad de tiempo para realizar pruebas que el que se había estimado al inicio. Ésto hizo que la integración del módulo desarrollado en la aplicación principal no se pudiera realizar en el plazo de la estancia en la empresa.

Todos los cambios llevaron a modificar el final de planificación, eliminando la parte de la integración e incrementando las horas dedicadas a las pruebas. Las figuras 2.3 y 2.4 representan la planificación final del proyecto.

Id	Nombre de tarea	Comienzo	Fin
1	**1. Propuesta Técnica**	**mar 24/03/15**	**jue 26/03/15**
2	**1.1. Inicio**	**mar 24/03/15**	**mar 24/03/15**
3	Definir proyecto con tutor y supervisor	mar 24/03/15	mar 24/03/15
4	Definir método de trabajo y documentación	mar 24/03/15	mar 24/03/15
5	Definir formato y estándares de trabajo	mar 24/03/15	mar 24/03/15
6	**1.2. Documentación y planificación**	**mar 24/03/15**	**mié 25/03/15**
7	Identificación de las tareas y tecnologías necesarias	mar 24/03/15	mié 25/03/15
8	Revisar contexto y buscar información	mié 25/03/15	mié 25/03/15
9	**1.3. Planificación del proyecto**	**mié 25/03/15**	**jue 26/03/15**
10	Definir tareas y estimar fechas	mié 25/03/15	mié 25/03/15
11	Crear diagrama de Gantt	mié 25/03/15	jue 26/03/15
12	Documentar propuesta del proyecto	jue 26/03/15	jue 26/03/15
13	Entregar propuesta técnica	jue 26/03/15	jue 26/03/15
14	**2. Desarrollo técnico del proyecto**	**jue 26/03/15**	**dom 05/04/15**
15	**2.1 Definir requisitos del proyecto**	**jue 26/03/15**	**sáb 28/03/15**
16	Evaluación de las necesidades	jue 26/03/15	vie 27/03/15
17	Definir y documentar requisitos de datos	vie 27/03/15	vie 27/03/15
18	Definir requisitos tecnológicos y de plataforma	vie 27/03/15	sáb 28/03/15
19	**2.2. Análisis**	**sáb 28/03/15**	**lun 30/03/15**
20	Estudio del funcionamiento de la aplicación global	sáb 28/03/15	sáb 28/03/15
21	Estudio de las interfaces gráficas proporcionadas	sáb 28/03/15	sáb 28/03/15
22	Análisis del código base	sáb 28/03/15	dom 29/03/15
23	Realizar esquemas de clases	dom 29/03/15	lun 30/03/15
24	**2.3. Diseño**	**lun 30/03/15**	**lun 30/03/15**
25	Estudiar diseño y estudiar cambios	lun 30/03/15	lun 30/03/15
26	Validar diseño	lun 30/03/15	lun 30/03/15
27	**2.4. Desarrollo del proyecto**	**lun 30/03/15**	**dom 05/04/15**
28	Programación	lun 30/03/15	vie 03/04/15
29	Pruebas	vie 03/04/15	dom 05/04/15
30	**2.5. Integración y entrega**	**dom 05/04/15**	**dom 05/04/15**
31	Integración del módulo con el programa principal y revisión.	dom 05/04/15	dom 05/04/15
32	Entrega final	dom 05/04/15	dom 05/04/15

Figura 2.1: Planificación inicial

Id	Nombre de tarea	Duración
1	**1. Propuesta Técnica**	**56 horas**
2	**1. 1. Inicio**	**11 horas**
3	Definir proyecto con tutor y supervisor	1 hora
4	Definir método de trabajo y documentación	5 horas
5	Definir formato y estándares de trabajo	5 horas
6	**1.2. Documentación y planificación**	**15 horas**
7	Identificación de las tareas y tecnologías necesarias	5 horas
8	Revisar contexto y buscar información	10 horas
9	**1.3. Planificación del proyecto**	**30 horas**
10	Definir tareas y estimar fechas	10 horas
11	Crear diagrama de Gantt	5 horas
12	Documentar propuesta del proyecto	15 horas
13	Entregar propuesta técnica	0 horas
14	**2. Desarrollo técnico del proyecto**	**245 horas**
15	**2.1 Definir requisitos del proyecto**	**40 horas**
16	Evaluación de las necesidades	10 horas
17	Definir y documentar requisitos de datos	15 horas
18	Definir requisitos tecnológicos y de plataforma	15 horas
19	**2.2. Análisis**	**45 horas**
20	Estudio del funcionamiento de la aplicación global	5 horas
21	Estudio de las interfaces gráficas proporcionadas	5 horas
22	Análisis del codigo base	20 horas
23	Realizar esquemas de clases	15 horas
24	**2.3. Diseño**	**10 horas**
25	Estudiar diseño y estudiar cambios	5 horas
26	Validar diseño	5 horas
27	**2.4. Desarrollo del proyecto**	**140 horas**
28	Programación	100 horas
29	Pruebas	40 horas
30	**2.5. Integración y entrega**	**10 horas**
31	Integracion del módulo con el programa principal y revisión.	10 horas
32	Entrega final	0 horas

Figura 2.2: Planificación inicial (horas)

Figura 2.3: Planificación final

Id	Nombre de tarea	Duración
1	**1. Propuesta Técnica**	**56 horas**
2	**1. 1. Inicio**	**11 horas**
3	Definir proyecto con tutor y supervisor	1 hora
4	Definir método de trabajo y documentación	5 horas
5	Definir formato y estándares de trabajo	5 horas
6	**1.2. Documentación y planificación**	**15 horas**
7	Identificación de las tareas y tecnologías necesarias	5 horas
8	Revisar contexto y buscar información	10 horas
9	**1.3. Planificación del proyecto**	**30 horas**
10	Definir tareas y estimar fechas	10 horas
11	Crear diagrama de Gantt	5 horas
12	Documentar propuesta del proyecto	15 horas
13	Entregar propuesta técnica	0 horas
14	**2. Desarrollo técnico del proyecto**	**260 horas**
15	**2.1 Definir requisitos del proyecto**	**40 horas**
16	Evaluación de las necesidades	10 horas
17	Definir y documentar requisitos de datos	15 horas
18	Definir requisitos tecnológicos y de plataforma	15 horas
19	**2.2. Análisis**	**35 horas**
20	Estudio del funcionamiento de la aplicación global	5 horas
21	Estudio de las interfaces gráficas proporcionadas	5 horas
22	Análisis del codigo base	10 horas
23	Realizar esquemas de clases	15 horas
24	**2.3. Diseño**	**10 horas**
25	Estudiar diseño y estudiar cambios	5 horas
26	Validar diseño	5 horas
27	**2.4. Desarrollo del proyecto**	**170 horas**
28	Programación	110 horas
29	Pruebas	60 horas
30	**2.5. Revisión y entrega**	**5 horas**
31	Revisión de todas las partes del proyecto	5 horas
32	Entrega final	0 horas

Figura 2.4: Planificación final (horas)

Capítulo 3

Análisis y diseño del software

3.1. Análisis de requisitos

3.1.1. Requisitos funcionales

Al analizar los objetivos del proyecto, se han extraído los requisitos funcionales. Debido a la diversidad de las tareas que realiza la aplicación, se han dividido dichos requisitos por las diferentes opciones que la componen:

- Localizar el teléfono mediante GPS.
- Bloquear el teléfono y/o hacer que suene una alarma cuando se realicen determinadas acciones.
- Controlar la ubicación del dispositivo mediante una delimitación geográfica. El teléfono realizará acciones específicas (se bloqueará, sonará una alarma y enviará su ubicación) cuando se encuentre fuera de un perímetro delimitado por una posición inicial y un radio, datos que serán leídos desde el GPS.
- Controlar cuando se introduce la contraseña de forma errónea. Si el móvil tiene cámara frontal se enviará una foto al e-mail del usuario de quien está intentando acceder y desde donde, es decir, geolocalizado.
- Controlar cambios de tarjeta SIM en el teléfono.
- Notificar al usuario por correo electrónico ante determinados eventos en cualquiera de las reglas configuradas.
- Poder localizar, bloquear, hacer que suene una alarma o borrar los datos del teléfono desde una aplicación web.

Una vez especificados los requisitos funcionales se pueden definir los requisitos de datos.

3.1.2. Requisitos de datos

Los requisitos de datos del proyecto son:

- Correo electrónico y contraseña para el registro y el acceso a la página web.
- Contraseña para la protección el teléfono.
- Numero de serie de la tarjeta SIM del teléfono, proporcionado por el mismo.
- Latitud y longitud, recibidas por el GPS.

3.2. Diseño del sistema

3.2.1. Contexto

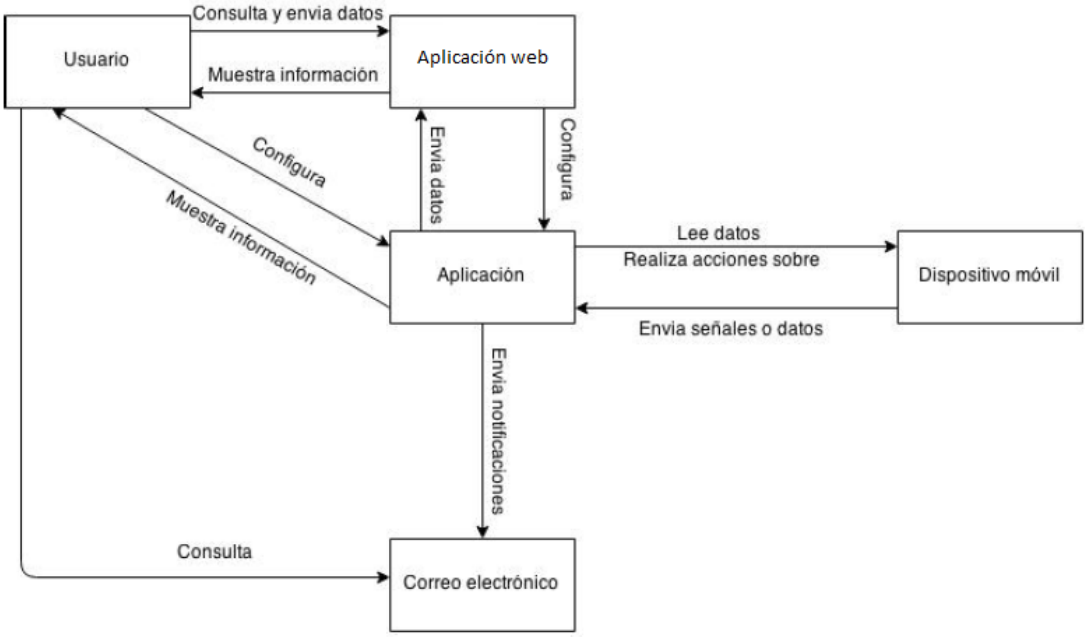

Figura 3.1: Diagrama de contexto.

El diagrama de la Figura 3.1 muestra como interaccionan los dos componentes que se desarrollan en el proyecto entre ellas, con el usuario y con el dispositivo.

El usuario puede configurar diferentes opciones en la aplicación. Ésta consulta datos continuamente del dispositivo y, cuando se producen determinados eventos, recoge información y la envía por correo electrónico al usuario.

Por otro lado, el usuario tiene acceso a una pagina web en la que puede consultar datos, como la localización, y que utilizará para enviar notificaciones a la aplicación y que ésta actúe

en consecuencia sobre el dispositivo.

3.2.2. Casos de uso

Como se puede observar en la Figura 3.2, el usuario utiliza tanto la aplicación móvil como la página web para la localización del teléfono. En el proceso de configuración de la aplicación se permite trabajar por separado cada una de las opciones, por lo que no se requiere configurarlas todas. Las opciones del teléfono que se modifican desde la página web son diferentes a las de la aplicación, teniendo la opción exclusiva del borrado de datos.

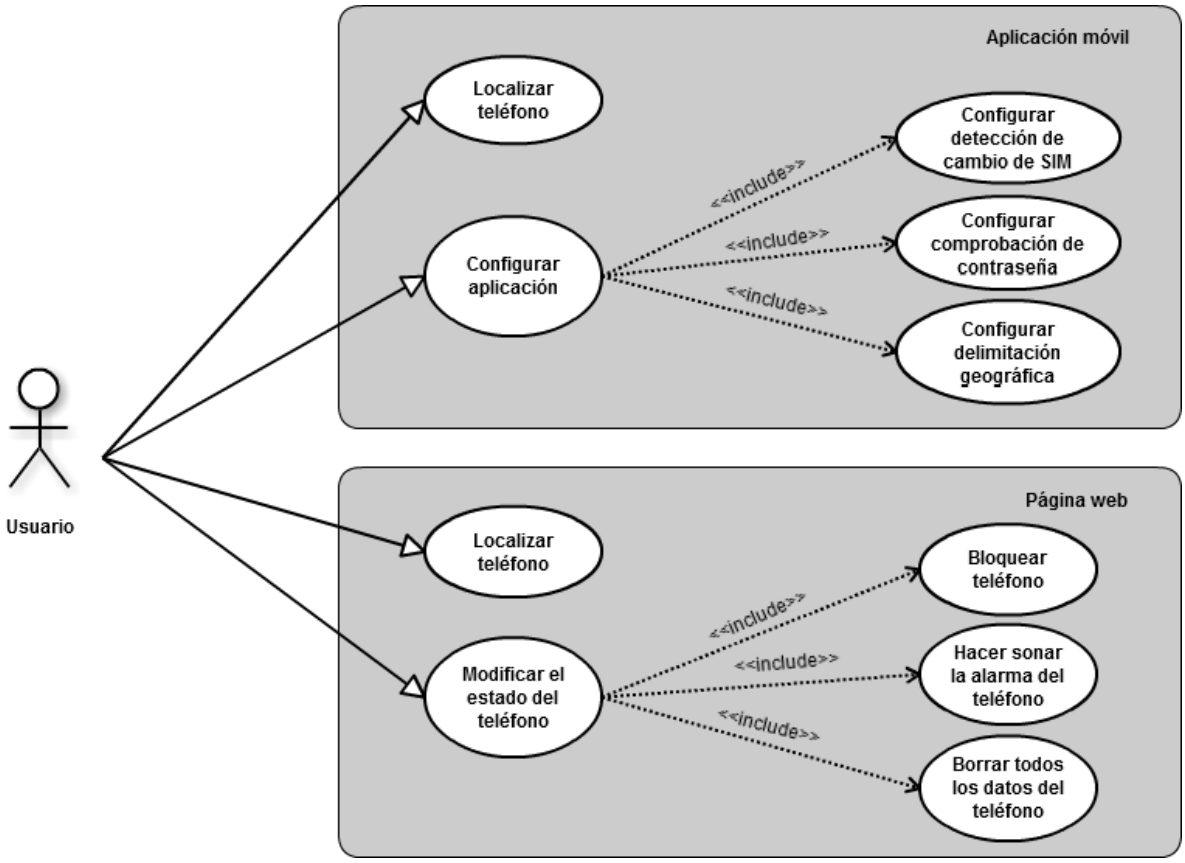

Figura 3.2: Diagrama de casos de uso

3.2.3. Estructura del proyecto

Aplicación móvil

Entre los principales componentes de una aplicación para Android se encuentran las actividades y los servicios. En la aplicación que se desarrolla en este proyecto se decide seguir el esquema presentado en la Figura 3.3. Utilizando este esquema, existe una actividad principal se encarga de declarar las variables necesarias y establecer los controladores. Uno de estos controladores se encargará de recibir los eventos en la interfaz gráfica, guardar los parámetros asociados

y ejecutar los servicios correspondientes. De forma similar, el segundo controlador estará a la espera de recibir notificaciones por Internet y arrancar los servicios que le sean indicados. La implementación de este esquema se estudiará en detalle en el apartado 4.

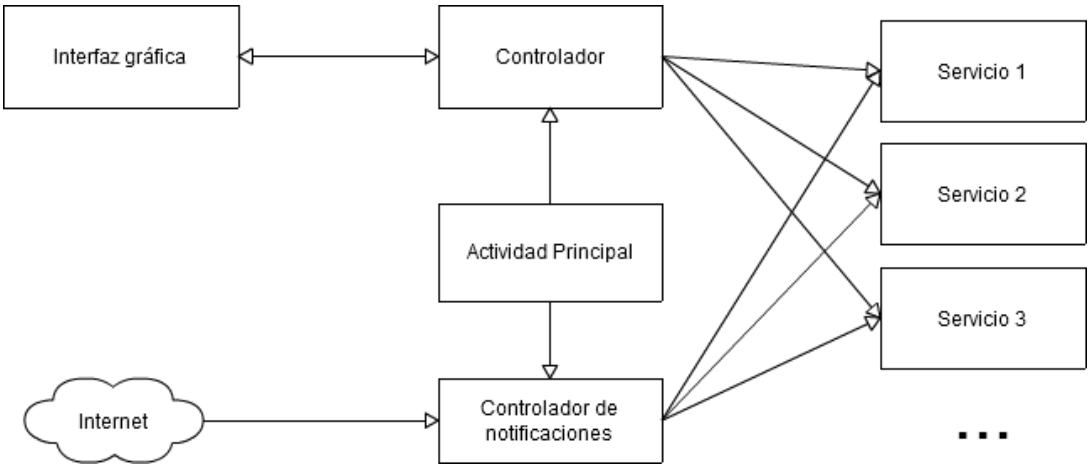

Figura 3.3: Esquema de la aplicación

Aplicación web

Para la implementación de la aplicación web de administración se decide utilizar el esquema presentado en la Figura 3.4. El servidor ofrece al usuario una serie de paginas web desde las que puede configurar a distancia la aplicación. Para ello utiliza una base de datos en la que almacena la información del usuario y de la aplicación que será utilizada para contactar con el teléfono mediante el servicio de notificaciones. Por último, la aplicación será la que se encargue de contactar con el servidor cuando quiera enviar datos como, por ejemplo, su localización. Todo ésto se verá en detalle en el apartado 4.

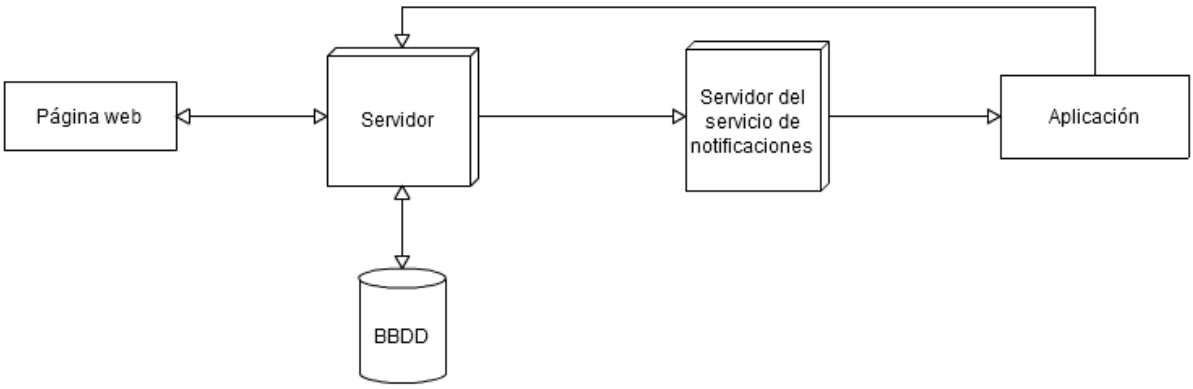

Figura 3.4: Esquema de la comunicación entre la aplicación web y la aplicación móvil

Capítulo 4

Desarrollo del proyecto

En este capítulo se mostrará el proceso de desarrollo, en orden cronológico, del proyecto. Cada apartado de la aplicación móvil y de la aplicación web será descrito en una sección, especificando lo que se debe cumplir, las herramientas y clases utilizadas, y las pruebas realizadas.

Debido al contrato de confidencialidad firmado por el autor de este proyecto con *Electronic Commerce Factory SL*, no es posible la inclusión del código fuente de la aplicación y de la aplicación web, sin embargo se le permite el uso de capturas de la interfaz. Debido a ésto, se utilizarán algoritmos a la hora de explicar ciertas partes del desarrollo.

La información sobre las clases que se mencionen en este capítulo y como utilizarlas se puede encontrar en el portal web de *Android Developers* (http://developer.android.com/index.html).

4.1. Desarrollo de la aplicación movil

Antes de comenzar a con el desarrollo se investiga la forma en la que se estructuraban las aplicaciones para móviles *Android*. Una vez hecho esto, se buscó una herramienta que permitiera trabajar con los archivos que componen una aplicación de forma cómoda. Una de las herramientas mas recomendadas fue *Android Studio*, por ello se aprendió a utilizarla antes de empezar.

Al inicio del desarrollo de la aplicación se utilizaba el simulador de dispositivos que ofrecía la propia herramienta, pero las limitaciones que tenia hicieron que no se tardara mucho en utilizar dispositivos reales.

Después de un periodo de práctica con la herramienta y teniendo en cuenta el esquema de la aplicación del apartado 3.2.3, se comienza el desarrollo de la aplicación por su actividad principal.

4.1.1. Actividad principal

La actividad principal es la que se ejecuta siempre al arrancar la aplicación del dispositivo, por eso se va a utilizar para realizar las preparaciones necesarias.

Variables

Existen muchas formas de almacenar variables en una aplicación Android. Las *SharedPreferences* son las que se utilizan en este proyecto por la conveniencia que supone poder compartir las variables en todas las clases que componen la aplicación.

Administrador de dispositivos

Las aplicaciones que administran el dispositivo a nivel de sistema necesitan tener una serie de permisos especiales. Desde Android 2.2 se añadió soporte a este tipo de aplicaciones proporcionando una API con la que se registra la aplicación como administradora del dispositivo, permitiendo así que pueda tener un mayor control sobre el mismo.

La mayoría de aplicaciones de seguridad, como la desarrollada en este proyecto, requieren crear un administrador de dispositivos para realizar tareas como el bloqueo de la pantalla, gestión de contraseñas, borrado de datos, etc.

En la aplicación que se desarrolla en este proyecto se crea el administrador de dispositivos en la actividad principal. Para ello se hace uso de la clase *DevicePolicyManager* con la constante *ACTION_ADD_DEVICE_ADMIN*. También se debe especificar que funciones se querrán poder realizar sobre el dispositivo mediante el administrador, sin indicar funciones innecesarias. La creación solo se efectuará en la primera ejecución y debe ser aceptada para continuar (Figura 4.1).

Creación de la interfaz gráfica

Android permite la creación de las interfaces gráficas de sus aplicaciones de varias formas diferentes. Incluye un gran número de layouts y elementos para construir una interfaz gráfica funcional. Sin embargo, a la hora de añadir estilos no ofrece tanta variedad.

Debido a esto, muchos desarrolladores de aplicaciones utilizan un tipo de vista que ofrece Android que permite poder diseñar las interfaces gráficas de la misma forma que en una página web convencional(*HTML*, *CSS*, etc). Esta vista se llama *WebView*, y funciona como un navegador web interno a la aplicación que permite mostrar paginas web en la propia aplicación, sean paginas web internas o externas.

En el desarrollo de la aplicación del proyecto se decide utilizar dicha interfaz porque las interfaces gráficas habían sido proporcionadas en lenguaje *HTML*.

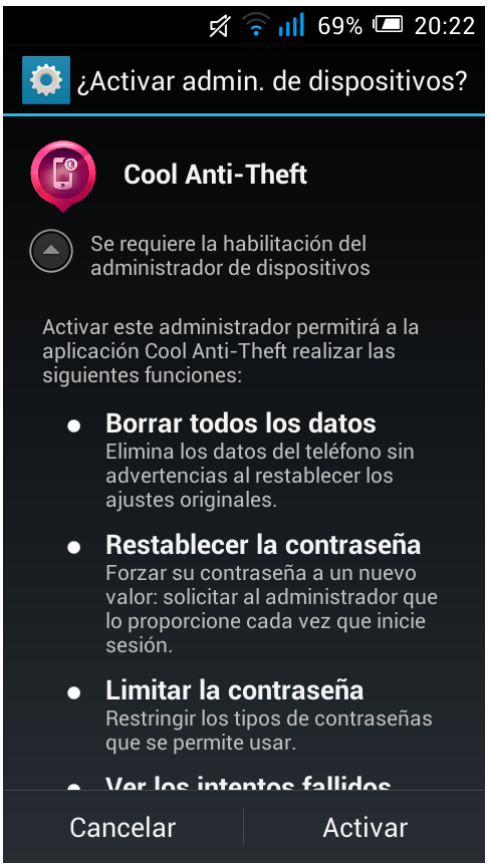

Figura 4.1: Pantalla de confirmación para crear el administrador de dispositivos.

La forma en la que *WebView* permite la interacción de la interfaz gráfica con la aplicación es mediante la asignación de una clase que actúa como medio de comunicación desde la vista hasta la actividad. El esquema de esta interacción se muestra en la Figura 4.2.

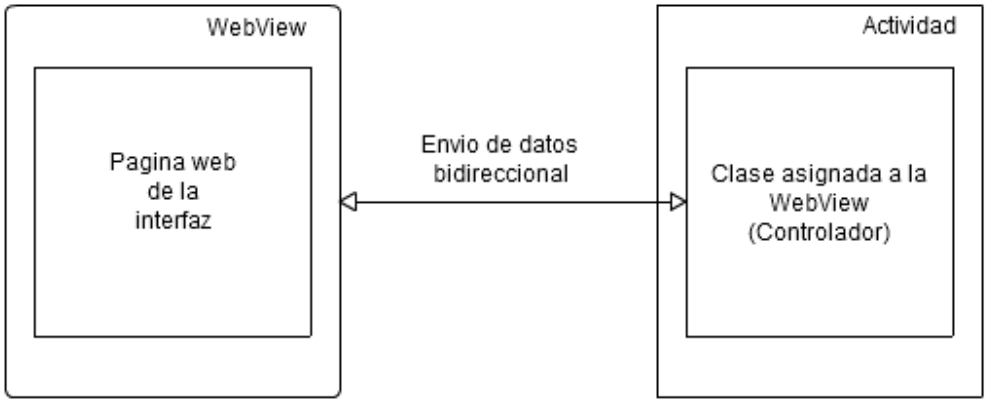

Figura 4.2: Esquema la comunicación entre la vista y el controlador.

Una vez establecida la clase que se relaciona con la vista, es importante saber que solo los métodos de esta clase con la etiqueta *@JavascriptInterface* encima de su declaración, podrán ser accedidos por la vista para consultar o enviar datos.

Finalmente, después de configurar los atributos de la vista, se debe indicar que pagina web se desea mostrar. En el propio código fuente de las páginas que mostremos, se pueden incluir enlaces a otras páginas. De esta forma se logra la navegación por las diferentes pantallas de la interfaz.

En la actividad principal de la aplicación es donde se realiza este proceso. En el caso de la aplicación que se está desarrollando, salvo contadas excepciones, la pagina web que se muestra al iniciar la aplicación será el menú principal con las opciones (Figura 4.3).

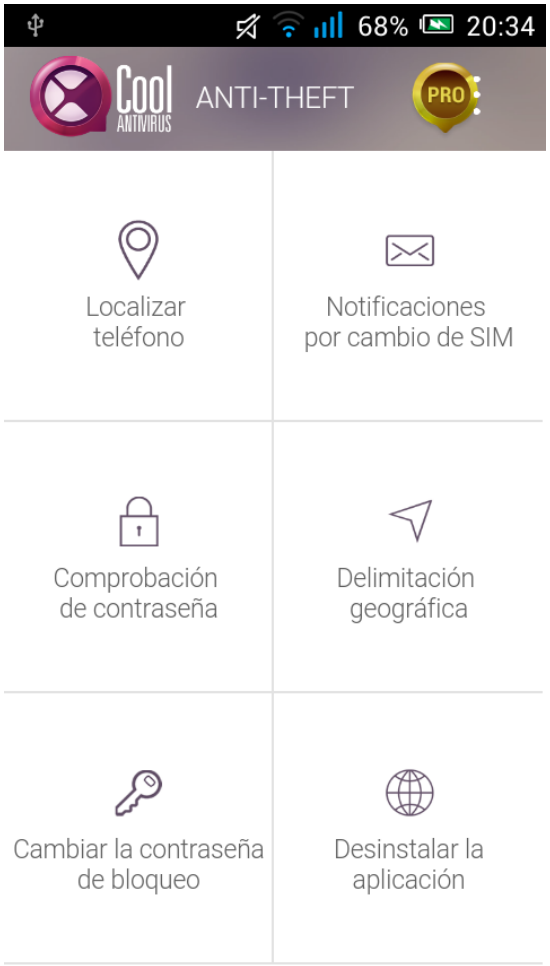

Figura 4.3: Menú principal de la aplicación.

Notificaciones push

Para la comunicación entre la aplicación web y la aplicación del dispositivo se utilizan las notificaciones push. Estas permiten que se pueda enviar información desde el servidor a cada dispositivo en particular. Se utiliza por la gran mayoría de aplicaciones que reciben notificacio-

nes desde un servidor. Un ejemplo pueden ser las aplicaciones de las redes sociales o mensajería instantánea como *WhatsApp* o *Telegram*. En ese tipo de aplicaciones los usuarios reciben notificaciones personalizas para cada dispositivo, es decir, solo reciben las notificaciones destinadas a ellos o a todos, pero nunca las de otro usuario. Esto se debe a que las notificaciones push funcionan asignando una IP al dispositivo que está usando la aplicación para poder enviarle la información que le corresponda.

Existen diversos servidores que ofrecen servicios de notificaciones push. El servicio que se utilizará en este proyecto es el que proporciona *Google*, el *Google Cloud Messaging* (ver Apendice C.1).

Los pasos que se deben seguir para la implementación de las notificaciones push son los siguientes:

1. Registrar a la aplicación como cliente en *GCM* para poder recibir notificaciones.
2. Almacenar el ID del registro recibido.
3. Enviar el ID del registro a la aplicación web para que la pueda utilizar para enviar mensajes.
4. Procesar las notificaciones recibidas desde el servidor *GCM*.

Para poder registrar a nuestra aplicación como cliente del *GCM*, se debe crear antes un proyecto en *Google Developers Console* (ver Apendice B.2).

Después de obtener el *número del proyecto* proporcionado al crear el proyecto en *Google Developers Console* , en la actividad principal de la aplicación móvil se solicita el ID de registro a *GCM* aportando ese número como parámetro de la petición. Eso hará que el dispositivo quede registrado dentro del proyecto que se ha creado y quede listo para recibir notificaciones.

Una vez obtenido el ID de registro, se almacena en las *SharedPreferences* para no tener que registrarse a cada ejecución de la aplicación. Posteriormente será enviado a la aplicación web para que pueda enviar las notificaciones (ver apartado 4.1.6).

La clase que procesará las notificaciones que se reciban mediante *GCM* se verán en el apartado 4.1.8.

Para que no se pueda registrar cualquier aplicación en el proyecto de la *Developers Console*, se especificará que solo se pueda registrar en el proyecto desde la aplicación que se está desarrollando. Para ello se genera una *API Key* el menú de credenciales del proyecto. Utilizando la clave, la aplicación accederá a las APIs que la requieran.

Pruebas

Lo primero que se comprueba es si el administrador de dispositivos se crea correctamente. Para saber si ha sido asi, después de aceptar en la pantalla de la Figura 4.1, se mira el apartado de *Administradores de dispositivos* para asegurarse de que se ha creado (Figura 4.4).

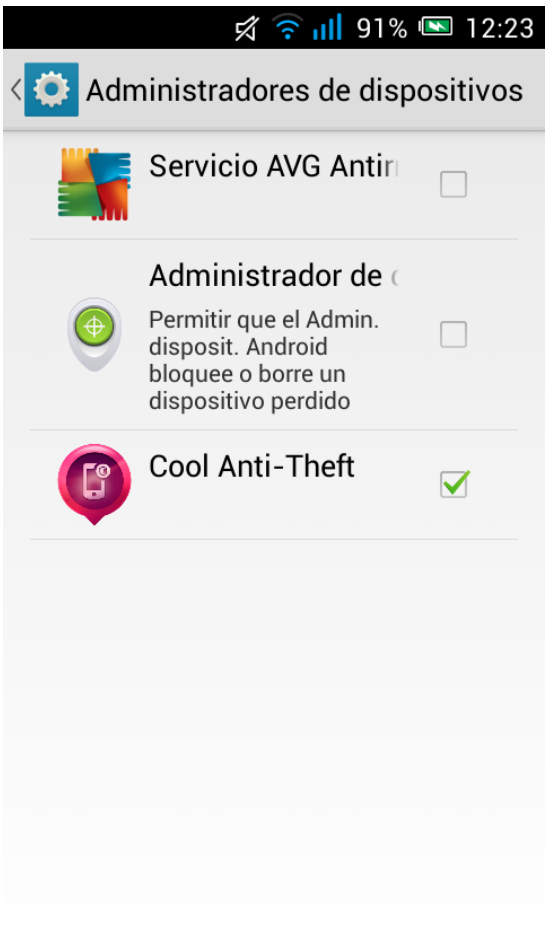

Figura 4.4: Administradores de dispositivos.

Lo siguiente que se comprueba es que se le haya asignado un *ID de Registro* al dispositivo para *Google Cloud Messaging*. Ésto se puede monitorizar al compilar la aplicación en el teléfono mediante la consola del *Android Studio*. Se observa que se ha recibido correctamente el código.

Las ultimas pruebas realizadas en este apartado corresponden a la creación de la interfaz gráfica. Al iniciar la aplicación se muestra la pantalla que hemos especificado y se puede comprobar que los enlaces a las pantallas de las diferentes opciones funcionan.

4.1.2. Localización

La primera opción de la aplicación que se implementa es la encargada de la localización del teléfono. Se decide emplear la *Google Maps Android API* (ver Apendice C.2) utilizando los mapas que ofrece para su uso en las aplicaciones móviles, combinándolos con las coordenadas que se obtienen del GPS del teléfono.

Preparación del mapa

Lo primero que se hace es incluir esta nueva API en el listado de las APIs habilitadas en el proyecto de *Google Developers Console*, para poder utilizarla. Lo siguiente es preparar la pantalla de la interfaz gráfica para mostrar el mapa. En este caso la interfaz gráfica está constituida por paginas web, por lo que incorporar el mapa no supone mucha complicación.

Como el mapa de la API debe ser descargado al entrar en el apartado de localización, si el usuario no dispone de conexión a Internet no podrá visualizarlo. La clase *ConnectivityManager* permite realizar esta comprobación para poder mostrar un mensaje al usuario cuando quiera localizar el teléfono y no disponga de acceso a Internet (Figura 4.5).

Figura 4.5: Pantalla de la localización cuando no se dispone de conexión a Internet.

Con lo anterior realizado, se dispone de un mapa que mostrar sin ningún marcador sobre él. Para colocar el marcador primero se deben obtener son las coordenadas desde el GPS del teléfono, por lo que se ha de comunicar la vista con el controlador e intercambiar los datos.

Obtención de las coordenadas por GPS

Como se comentó en la lista de objetivos específicos del apartado 1.2.2, la localización se realiza exclusivamente por GPS.

La clase que proporciona los servicios de localización del sistema *Android* es *LocationManager*. Lo primero que se hace es comprobar si el GPS del teléfono está activado utilizando esta clase. Si no lo está se le muestra al usuario un mensaje de forma similar a cuando no dispone de conexión a Internet (Figura 4.6).

Figura 4.6: Pantalla de la localización cuando no está activado el GPS del dispositivo.

Una vez comprobada la disponibilidad del proveedor de GPS, se le indica al *LocationManager* que proporcione las coordenadas. Lamentablemente, no existe ninguna forma de obligar al GPS a leer las coordenadas en el momento deseado. Solamente se tiene acceso a la última posición conocida (que puede ser reciente, pasada, o no tener ningún sentido). Lo único que podemos hacer es establecer un "escuchador"de eventos que detecte cuando se actualiza esa última posición conocida. La clase que se utiliza para ello es *LocationListener*, cuyo método *onLocationChanged* nos permite realizar acciones cuando la posición se actualiza.

La forma en la que se realiza el proceso es la siguiente:

1. Indicar mediante el *LocationManager* al dispositivo que se requiere una actualización de la posición, pasándole como parámetro el "escuchador".

2. Esperar a que se actualice a posición.

3. Utilizar el método *onLocationChanged*, que se ejecuta cuando se actualiza la posición, para almacenar la latitud y longitud e indicar que ya pueden ser enviadas a la vista.

4. Indicar que ya no se requieren mas actualizaciones de posición al GPS.

Mientras este proceso se realiza, la vista está contactando periódicamente con el controlador para preguntarle si ya tiene las coordenadas disponibles, mientras muestra un mensaje de espera (Figura 4.7). Una vez estén disponibles, la vista las consultará, creará un marcador en el mapa y mostrará el resultado de la localización (Figura 4.8). Cuando se pulse el botón "Actualizar"de la interfaz, se repetirá el proceso de obtención de coordenadas y se mostrará la nueva posición cuando estén disponibles.

Figura 4.7: Pantalla de la localización cuando se esperan las coordenadas.

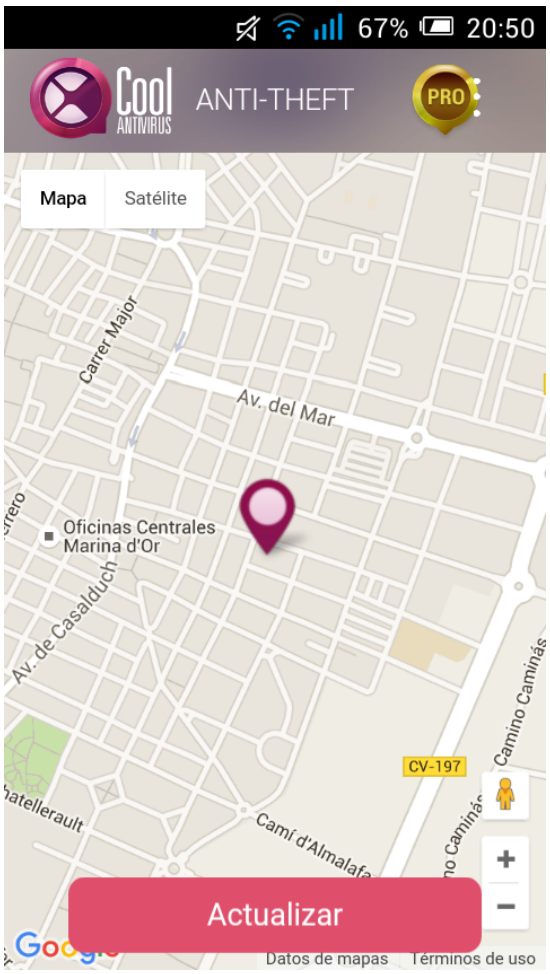

Figura 4.8: Pantalla de la localización con la posición encontrada sobre el mapa.

Pruebas

Las pruebas realizadas sobre esta opción de la aplicación consisten en comprobar el funcionamiento en diferentes situaciones. Al desactivar el GPS se observa que se muestra el mensaje de la figura 4.6. Del mismo modo, al desactivar la conexión a Internet se muestra el mensaje de la figura 4.5.

Se observa que, dependiendo del entorno, el GPS tarda un tiempo diferente en obtener las coordenadas. Aunque esto escapa al control que se puede ejercer desde la aplicación sobre el GPS, se establece una prioridad alta en la solicitud de la actualización de la posición.

Se ajustó el tamaño del marcador en el mapa y el zoom para que fuera proporcional a la interfaz y se probó en diferentes dispositivos.

4.1.3. Comprobación de la tarjeta SIM

Después de finalizar con la localización, se pasa a implementar el servicio que comprueba la tarjeta SIM del teléfono. El objetivo de esta opción es notificar al usuario cuando se haya retirado la tarjeta SIM del teléfono o se haya cambiado por otra diferente. Si ésto ocurre, la aplicación envía un correo electrónico al usuario notificándole lo ocurrido, adjuntando el numero de serie de la tarjeta nueva y la localización del teléfono.

La pantalla de la interfaz gráfica de esta tarea dispone únicamente de un interruptor para activar el servicio (Figura 4.9). El único requisito para poder activar el servicio es tener una tarjeta SIM colocada. Si no la tenemos, se avisa al usuario con un mensaje.

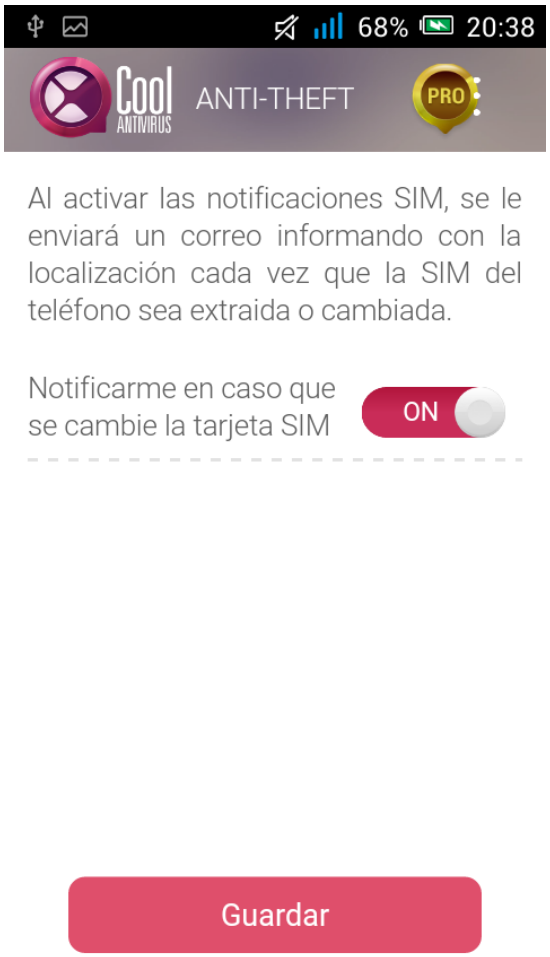

Figura 4.9: Pantalla de la opción de comprobar el estado de la tarjeta SIM.

El proceso que se sigue una vez iniciado el servicio es el siguiente:

1. Se almacena el número de serie de la tarjeta SIM actual.
2. Se comprueba periódicamente el estado de la tarjeta SIM hasta que se cambie por otra o se retire. Cuando esto ocurre, se continua con el paso 3.
3. Se bloquea el teléfono con una contraseña.

4. Se localiza el teléfono si el GPS está activado.

5. Se envía un correo con toda la información al usuario.

En los siguientes apartados se comenta como se implementa cada parte del proceso y las pruebas que se realizan.

Lectura del estado de la SIM y bloqueo del teléfono.

La clase que es capaz de proporcionar información sobre la tarjeta SIM es *TelephonyManager*. Se utiliza tanto para obtener el numero de serie del SIM como para conocer su estado.

Al activar el servicio, lo primero que se ha de hacer es almacenar el número de serie del SIM actual para tener una referencia a la hora de comparar. El método que lo proporciona es *getSimSerialNumber*.

Una vez conocido el numero de serie actual, se crea un hilo que se encarga de comprobar periódicamente el estado del la tarjeta SIM. La comprobación observa mediante *getSimSerialNumber* que el numero que se obtiene no sea diferente al que se leyó en un principio. También comprueba si hay alguna tarjeta SIM colocada con *getSimState*, comprobando que el numero que devuelva no sea igual al de la constante *SIM_STATE_ABSENT* de la clase *TelephonyManager*.

Si alguna de estas condiciones no se cumple, se realizan las acciones descritas a continuación.

Bloqueo del teléfono

Bloquear el dispositivo es la primera acción que se realiza cuando se detecta una alteración en la tarjeta SIM del teléfono. Se hace uso de la clase *DevicePolicyManager* junto a su método *resetPassword* para establecer la contraseña que el usuario introdujo cuando se registró en la aplicación. Después se utiliza el método *lockNow* para bloquear el teléfono inmediatamente.

Localización

Se desea enviar la localización del teléfono al usuario para que éste pueda saber donde se encuentra el teléfono al que le han modificado la tarjeta SIM. Aprovechando que el servicio de obtención de coordenadas desde el GPS implementado en el apartado 4.1.2, solo se necesita añadir una variación al mismo para adaptarlo a este nuevo servicio.

La principal modificación se realiza en el método *onLocationChanged*, que es el que se ejecuta al encontrar una nueva posición, haciendo que una vez la encuentre inicie la tarea de enviar el correo con toda la información.

Preparación y envio de correos

Lo primero que se hace es investigar la forma en la que *Android* puede enviar correos en segundo plano. Es necesario que se pueda enviar sin pedir confirmación del usuario ya que obviamente el supuesto ladrón no aceptaría enviarlo.

Tras un periodo de búsqueda, se decide utilizar el servidor de *Simple Mail Transfer Protocol* (ver Apendice D.1) de *Gmail*. Para poder enviar el correo de esta forma se necesitan realizar los siguientes pasos:

1. Iniciar una sesión en el servidor *smtp.gmail.com* mediante usuario y contraseña en la cuenta de correo electrónico desde la que se va a enviar el correo. Hay que indicarle que se desea utilizar TLS para cifrar los datos y la petición ha hacerse al puerto 587 del servidor, que es el destinado a este propósito.
2. Construir y asignar a la sesión el mensaje que se desea enviar, incluyendo el origen, destinatario, asunto y cuerpo del mensaje.
3. Enviar el mensaje.

Para iniciar la sesión en la aplicación, se utiliza un objeto de la clase *Session* en el que se indica que se ha de se ha de iniciar sesión en el servidor *smtp.gmail.com*, que el puerto de esa conexión es el 587 y que se utiliza TLS. Se utiliza el método *getInstance*, pasandole el usuario y contraseña encapsulados en una clase que implementa la interfaz *Authenticator*, para que los utilice en el inicio de la sesión.

Para la construcción del mensaje del correo se utiliza la clase *MimeMessage*, pasandole como parámetro el objeto de la sesión que hemos creado. En el destinatario del mensaje se introduce el correo con el que el usuario se registró en la aplicación. En el cuerpo del mensaje incluimos un enlace a *Google Maps* con las coordenadas que se han obtenido anteriormente (si el GPS está activado).

Dependiendo de si se ha retirado la tarjeta SIM o si se ha cambiado por otra diferente, se añadirá una linea u otra al mensaje del correo. Si la tarjeta se ha cambiado por otra se indicará el número de serie de la nueva.

Una vez el contenido del mensaje está completado, se envía el correo en segundo plano utilizando una *AsyncTask* y se finaliza el servicio.

Reinicio del teléfono

El servicio de notificaciones ante cambios en la SIM se deja de ejecutar cuando el teléfono se reinicia. Para que se siga ejecutando debemos hacer que el sistema lo incluya entre las aplicaciones que arranca al encender el teléfono.

La clase *BroadcastReceiver* permite a la aplicación revisar ciertos eventos que ocurren en el teléfono. En este caso se ha de observar el evento que se corresponda con la constante

BOOT_COMPLETED, que indica que el teléfono se ha acabado de iniciar. Después se ha de comprobar si la aplicación tiene marcado el servicio como activo. Al cumplirse ambas condiciones se arranca el servicio de comprobación de la SIM.

Pruebas

Las pruebas que se realizan sobre esta parte de la aplicación consisten en comprobar las diferentes situaciones que se pueden dar. Se prueba a retirar la tarjeta del teléfono, a cambiarla por una diferente, a reiniciar el teléfono con una nueva SIM y a reiniciarlo sin ninguna. En todas las ocasiones se consigue que el teléfono se bloquee y que se envie el correo.

Un problema que se detecta realizando las pruebas es que el tiempo que tarda el GPS en encontrar la posición varía bastante entre lugares y momentos, hecho que hace que el correo tarde demasiado en enviarse con la localización. Esto supone el riesgo de que el culpable apague el teléfono antes de que la localización se complete y se envíe el correo. Como no se puede controlar el momento en el que estará localizado el teléfono, se decide enviar dos correos. El primero se envía inmediatamente tras detectar el cambio o ausencia de la tarjeta SIM y una vez se haya obtenido la posición, se envía un segundo correo con los datos de localización restantes.

4.1.4. Comprobación de contraseña

La siguiente tarea de la lista es la comprobación de la contraseña de desbloqueo del teléfono. El objetivo de este servicio es establecer una contraseña para la pantalla de bloqueo del teléfono y notificar al usuario si se introduce incorrectamente durante tres intentos. Si esto ocurre, el servicio envía un correo al usuario con la localización del teléfono y, si el dispositivo dispone de una cámara delantera, una foto del culpable.

La pantalla de la aplicación que se utiliza para activar el servicio consta de un interruptor para indicar si lo queremos activar o desactivar (Figura 4.10). Cuando lo activemos bloqueará el teléfono con la contraseña que el usuario introduce al registrarse en la aplicación (apartado 4.1.6).

El proceso que sigue el servicio que se ejecuta después del tercer intento fallido es:

1. Comenzar el proceso de localización si el GPS está activado.

2. Hacer la fotografía si la cámara dispone de una cámara delantera y guardarla.

3. Construir el correo con los datos de los pasos anteriores

4. Enviar correo.

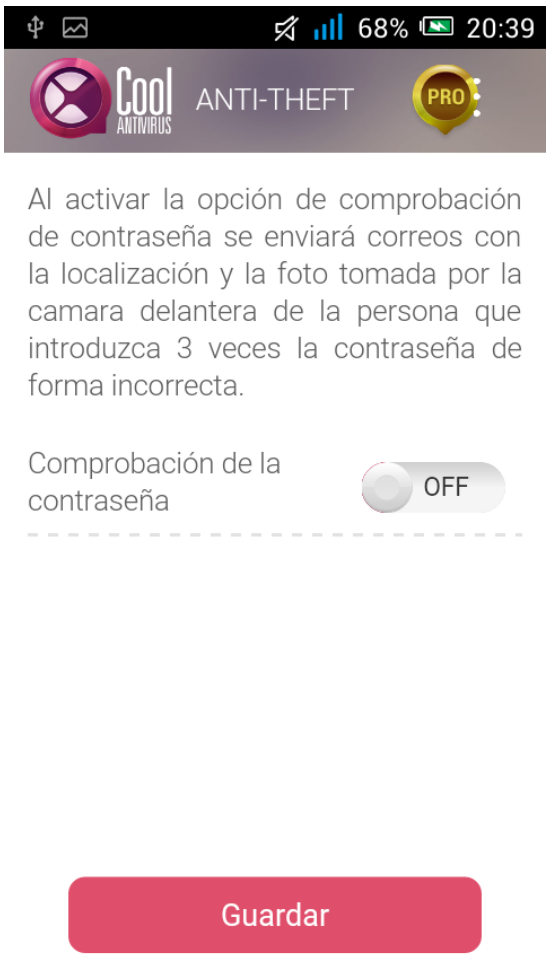

Figura 4.10: Pantalla de la opción de comprobación de la contraseña.

Detección de fallos en la introducción de la contraseña

El servicio que se está implementando se debe ejecutar una vez el usuario haya introducido incorrectamente la contraseña tres veces.

Android proporciona una clase que permite llevar un control sobre los intentos y la posibilidad de actuar cuando se falla al introducir la contraseña. Esta clase es *DeviceAdminReceiver*. Para este propósito se debe sobrescribir el método *onPasswordFailed*, que se ejecuta cada vez que el usuario falla la contraseña, para que realice la acción que se desea.

En este caso utilizaremos una clase propia que herede de *DeviceAdminReceiver* y que sobrescriba el método de forma que, al tercer fallo de introducir la contraseña, se ejecute el servicio que se ha creado. Para ello se utilizará el método *getCurrentFailedPasswordAttempts* de la clase *DevicePolicyManager*, que lleva la cuenta de los intentos fallidos. Cuando este contador llegue a tres, se ejecutará el servicio.

Fotografía del culpable

Android ofrece una gran cantidad de clases destinadas al manejo de los periféricos del dispositivo. Entre ellas se encuentran las destinadas a controlar las cámaras del dispositivo desde la aplicación.

Lo primero que se hace a la hora de realizar la fotografía es comprobar si el dispositivo tiene una cámara delantera. Para ello utilizamos la clase *Camera* que permite, entre otras cosas, conocer el numero de cámaras del que se dispone y las características de cada una de ellas. Si no se encuentra una cámara delantera en el dispositivo, se cancela la toma de la fotografía y se realiza el envío del correo. Una vez se conoce que el dispositivo tiene una cámara delantera se procede a tomar la fotografía.

El problema a la hora de realizar la fotografía es la forma de hacerla. Las clases y métodos que proporciona *Android* no permiten hacerla sin mostrar una previsualización de lo que se está fotografiando. Esto supone un problema ya que queremos que la fotografía sea tomada en segundo plano sin ningún tipo de interacción y confirmación por parte del usuario. Para solucionar esto se crea una previsualización personalizada mediante la clase *SurfaceView* y su método *getHolder*. Utilizando ésto, la clase *Camera* nos permite capturar imágenes por la cámara en segundo plano de forma instantánea.

Una vez obtenida la imagen, se intenta almacenarla en la tarjeta de almacenamiento externo si es posible, ya que es donde se suele disponer de mas espacio.

Finalmente se cierra la previsualización de la cámara y el proceso de captura, dando paso al envío del correo.

Localización del dispositivo y envio del correo

Para la localización del dispositivo y el envío del correo se utilizan pequeñas variaciones en el código de los servicios tratados en el apartado 4.1.3.

Se cambian las condiciones que construyen el cuerpo del mensaje por unas adaptadas a este servicio. Se comprueba si se ha podido localizar el teléfono o no, si se ha podido tomar la fotografía o no, etc. La clase que utilizamos para construir el mensaje del correo es diferente ya que esta vez se adjunta una imagen en el correo. Por este motivo utilizamos la clase *MimeMultipart*, con la que se construye un correo dividido en dos partes: el texto y la imagen.

Pruebas

Para este servicio se sigue un proceso de pruebas similar a los de los anteriores servicios. Se comprueba que la el servicio envíe el correo tanto cuando puede localizar el teléfono como cuando no puede. También se observa que la imagen enviada al correo es la tomada por la cámara delantera.

Del mismo modo que ocurría en las pruebas del apartado 4.1.3, la localización del dispositivo varia dependiendo de la situación y el momento. La solución que se implanta para este servicio es la misma, se envían dos correos. En este caso, el primero de ellos contiene el texto de la alerta de fallos en la introducción de la contraseña junto con la imagen captada por la cámara delantera. El segundo de ellos contiene un enlace con la localización del dispositivo.

4.1.5. Delimitación geográfica

Después de completar el servicio de comprobación de contraseña, se comenzó a desarrollar el servicio de delimitación geográfica. El objetivo de este servicio es comprobar la ubicación del teléfono durante un periodo de tiempo. El usuario establece una posición inicial y un radio que limita la distancia a la que puede desplazarse el dispositivo desde esa posición. Si el teléfono se sale del área especificada, se realizan las acciones que el usuario haya indicado (Figura 4.11).

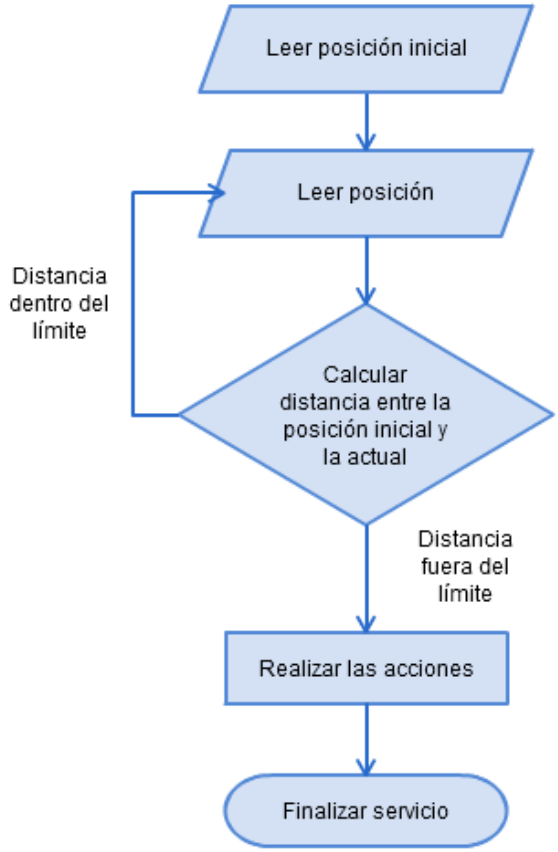

Figura 4.11: Diagrama de flujo del servicio de delimitación geográfica.

La pantalla de la aplicación destinada a este servicio (Figura 4.12) utiliza un interruptor para activar el servicio, un campo numérico para indicar el radio que se desea, y un botón que conduce a la pantalla que permite configurar las acciones a realizar en caso de que el teléfono se salga del radio (Figura 4.13).

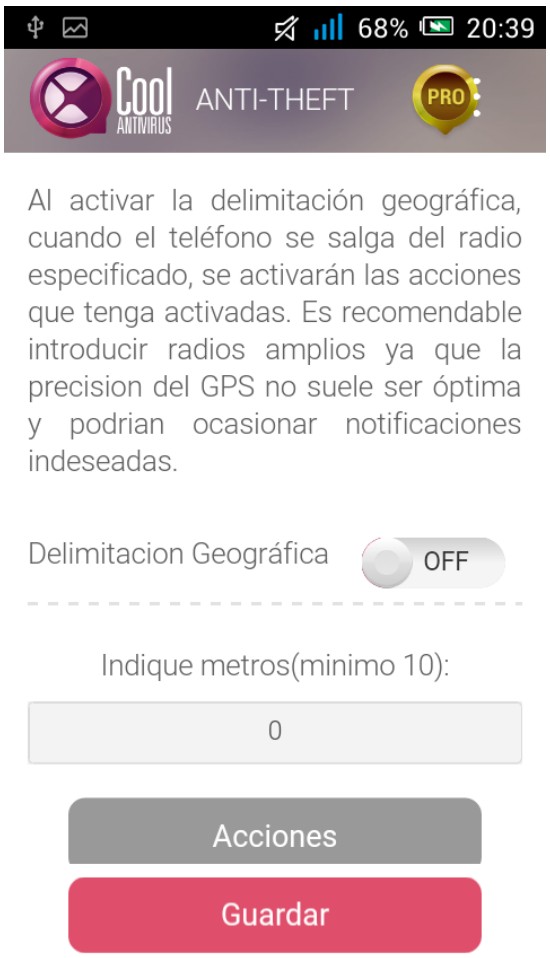

Figura 4.12: Pantalla de configuración de la delimitación geográfica.

Para este servicio se aprovechan las partes de localización y envio de correos implementadas en servicios anteriores. La parte nueva de este servicio es la activación de la alarma.

Activación de la alarma

Una vez el teléfono se sale del radio se activan las acciones que el usuario ha especificado. Si el usuario tiene activada la opción de hacer sonar una alarma, esta se reproduce al máximo volumen del dispositivo sin permitir desactivarla hasta que el usuario desbloquee el teléfono.

Las clases que se utilizan para configurar la alarma son principalmente las clases *AudioManager* y *Ringtone*. La primera permite, entre otras cosas, que se configure el volumen del teléfono. Con la segunda se elige el tono a reproducir y con el método *play* se reproduce. Combinando ambas clases se consigue hacer sonar el tono a máximo volumen.

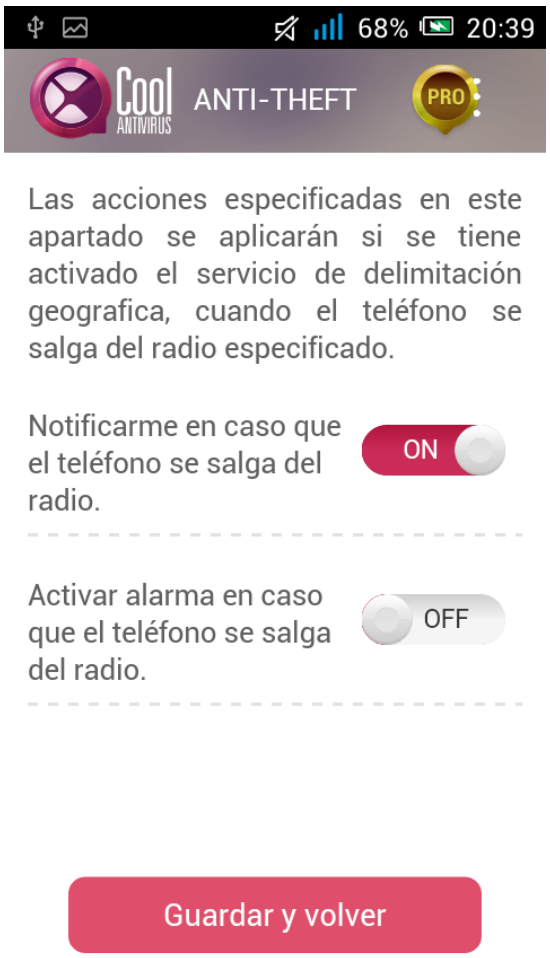

Figura 4.13: Pantalla de acciones de la delimitación geográfica.

Pruebas

En este apartado se comprueba que todas las opciones funcionen correctamente. Se prueba a activar y desactivar las acciones que se hacen tras salirse del radio y se comprueba que se ejecuten solamente cuando deben.

Se observa que se deben especificar radios relativamente amplios porque la precisión del GPS tiene margenes de error de decenas de metros.

4.1.6. Registro

La primera acción que el usuario realiza al ejecutar la aplicación por primera vez es registrarse. El objetivo es que el usuario envíe los datos necesarios para la administración remota del teléfono al servidor que se desarrolla en el apartado 4.2 y que establezca una contraseña que permita bloquear el teléfono cuando sea necesario. Este proceso solo se realiza una vez. Cuando el usuario se ha registrado de forma satisfactoria se le conduce al menú principal de la

aplicación.

La pantalla que se muestra al usuario para registrarse le pide introducir un correo electrónico y una contraseña (Figura 4.14), que se utilizarán para la autenticación en la página web.

Figura 4.14: Pantalla de registro.

Envio de datos al servidor

Cuando se llega a esta parte de la implementación, el servidor aun no ha sido creado. Lo que se hace es dejar el servicio listo para que cuando se cree el servidor solo se tengan que introducir pequeñas modificaciones.

Después de que el usuario introduce los datos del registro, éstos se envían desde la vista al controlador para comprobar que las contraseñas coincidan y que los campos no estén vacíos. Una vez comprobado, se envían los datos al servicio.

La tarea que se realiza en el servicio es el envío de los datos al servidor. Para ello se establece una conexión HTTPS mediante un objeto de la clase *HttpsUrlConnection*. Para que la conexión sea segura se debe de crear un certificado del servidor y se debe añadir al objeto de esta conexión

para que la aplicación acceda al servidor correcto.

Los datos que se envían desde este servicio al servidor son los que se recogen en la vista sumados al ID de registro que se obtiene al registrarse en *Google Cloud Messaging* cuando se configuran las notificaciones push en la actividad principal (apartado 4.1.1). Enviando este ID, el servidor reconocerá a que dispositivo hay que enviar las notificaciones cuando el usuario se autentique en la página web.

Después de configurar el servidor web del apartado 4.2 se hizo que, una vez se registrara el usuario correctamente en el servidor, se le mostrara un mensaje (Figura 4.15) y se procediera a la configuración de la contraseña de bloqueo.

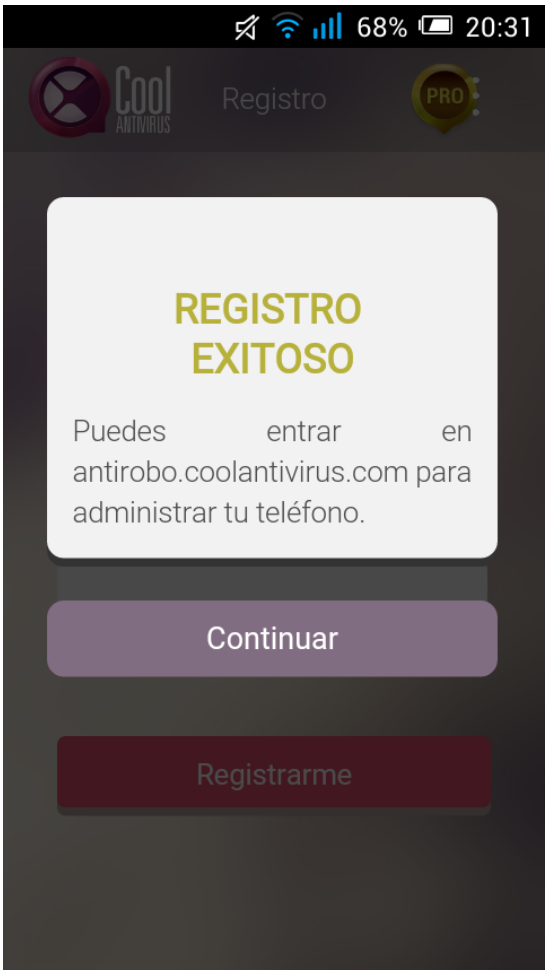

Figura 4.15: Pantalla de éxito al registrarse.

Configuración de la contraseña de bloqueo

Después del registro se le muestra una pantalla al usuario para que introduzca la contraseña con la que se bloqueará el teléfono en caso de robo o extravío(Figura 4.16). Esta contraseña podrá ser cambiada en una de las opciones de la aplicación. Una vez introducida al contraseña se envía al usuario al menú principal de la aplicación. Esto significa que el registro está completado.

En las siguientes ejecuciones se mostrará el menú principal y no la pantalla de registro.

Pruebas

Hasta que no se creó el servidor no se hicieron las pruebas de registro. Se configuró un servidor en la red de la empresa para realizar pruebas locales. La aplicación se conectaba correctamente y recibía los datos de la aplicación para almacenarlos en una base de datos. Cuando no era posible contactar con el servidor se le mostraba un mensaje al usuario indicándole que no tenia conexión a Internet o que el servidor no estaba operativo.

Se comprobó que la contraseña de bloqueo se creaba correctamente utilizando los servicios creados anteriormente, viendo como los bloqueos necesitaban esa contraseña para permitir el acceso.

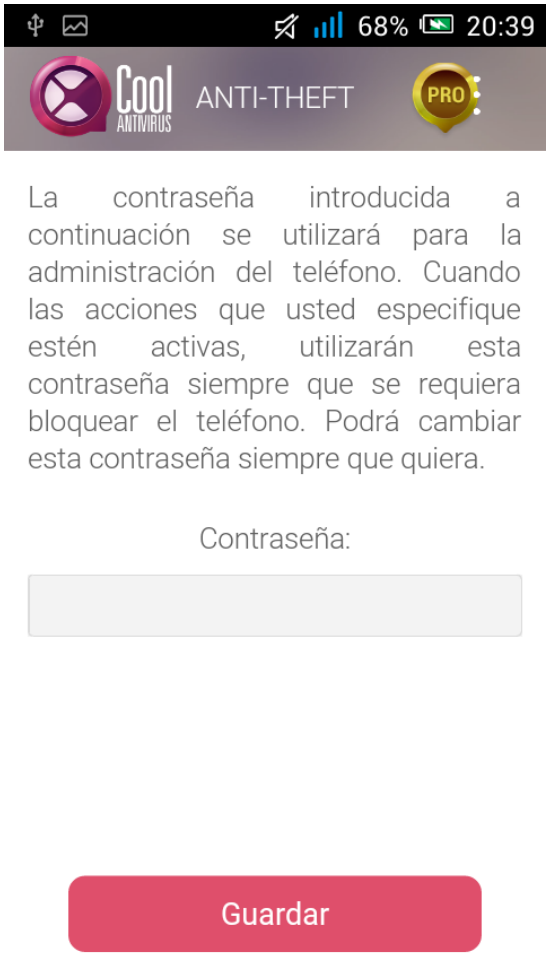

Figura 4.16: Pantalla de configuración de la contraseña de bloqueo.

4.1.7. Borrado de datos

Este es un servicio que se ejecuta exclusivamente cuando se recibe la notificación correspondiente desde la aplicación web (apartado 4.2.6). Consiste en restaurar el teléfono a su configuración de fábrica y borrar la información del almacenamiento externo que pueda tener.

El servicio utiliza *DevicePolicyManager* y su método *wipeData* para restaurar el dispositivo a su estado de fábrica sin confirmación. Con el flag *WIPE_EXTERNAL_STORAGE* se formatea también el almacenamiento externo.

4.1.8. Recibidor de notificaciones

En el apartado 4.1.1 se configuró un *ID de registro* mediante el que la aplicación recibe las notificaciones push desde *Google Cloud Messaging*. Pero eso no es suficiente para poder recibirlas y tratarlas. Es necesario crear una clase que diferencie los parámetros de la notificación que lleguen a la aplicación y arranque los servicios correspondientes.

La clase que se utiliza es una variante de *BroadcastReceiver* que se utiliza en el apartado 4.1.3 a la hora de arrancar el servicio al reiniciar el teléfono. La diferencia es que en esta clase ya no se comprueba si el evento se corresponde con el de haber completado el reinicio. Lo que se hace es comprobar los parámetros de la notificación para decidir que servicio hay que arrancar. Estos parámetros son los que se enviarán desde el servidor que se crea para la administración vía Internet en el apartado 4.2.

4.2. Desarrollo de la aplicación web

Lo primero que se hace al llegar a este punto del desarrollo es buscar los lenguajes que se van a utilizar para implementar el servidor y la base de datos. Dado que el autor del proyecto tiene experiencia en el uso del lenguaje *Python*, se decide utilizar *Tornado* (ver Apendice E.1), un framework y un conjunto de librerías escritas en dicho lenguaje que permiten, entre otras cosas, la creación de servidores. Para la base de datos que se crea para almacenar los datos de registro se utiliza *MySQL*.

Las clases utilizadas en este apartado pertenecen a *Tornado*.

4.2.1. Servidor

Se comienza la implementación configurando los parámetros del servidor. La clase *HTTPServer* permite que se cree uno. Al constructor de esta clase se le indican algunos de los parámetros del servidor como el certificado y la llave que le permite establecer conexiones cifradas con el cliente. Mediante el método *listen* se le indica el puerto en el que debe escuchar las peticiones. También se le pasa un objeto de la clase *Application* en el que se declararán las URLs de los

recursos del servidor a los que se puede acceder y los métodos que tratan las peticiones a estos recursos.

Para establecer la conexión con el servidor de *Google Cloud Messaging* se utiliza la librería externa *python-gcm* (https://github.com/geeknam/python-gcm). Esta librería nos proporciona la clase GCM en cuyo constructor se introduce una clave generada en el proyecto de *Google Developers Console*, que se creó al preparar las notificaciones push en el apartado 4.1.1, que permite identificar el servidor que se está creando en el servidor de *Google Cloud Messaging* para que éste sepa a que dispositivos tiene que enviar las notificaciones.

Lo ultimo que se configura al arrancar el servidor es la conexión a la base de datos. Para hacerlo se utiliza la clase *Connexion* de la librería *torndb* (https://github.com/bdarnell/torndb). Mediante esta conexión se accede a los datos que los usuarios enviaron a la aplicación en el proceso de registro del apartado 4.1.6, entre otros.

4.2.2. Autenticación

Una vez configurados los parámetros del servidor, se pasa a la implementación de los métodos que tratan las peticiones.

La primera petición que se implementa es la de autenticación. Se utiliza un sistema de sesiones gestionadas mediante cookies que son soportadas por *Tornado*. Este sistema se basa en utilizar cookies cifradas para guardar la información del usuario en el navegador. Los métodos que traten las peticiones del navegador hacia el servidor comprueban que el usuario esté autenticado leyendo la cookie correspondiente y, si no se encuentra, se redirige al usuario hacia la pantalla de autenticación.

Los métodos utilizados para tratar las peticiones se implementan extendiendo la clase *RequestHandler*. Cada una de las clases que extendamos responderá a alguna URL y tratará la petición asociada.

La clase que se utiliza para la autenticación recoge la información de la pantalla de *login* (Figura 4.17) y comprueba que en la base de datos exista una entrada con esos datos. Si existe se crea una cookie en el navegador mediante el método *set_secure_cookie*. Si no, se muestra un mensaje de error (Figura 4.18). Una vez autenticados, el resto de métodos comprobarán que la cookie esté presente antes de ejecutar su código.

Para cerrar sesión se crea otra clase que simplemente elimina la cookie y redirige a la pantalla de *login*.

4.2.3. Recuperación de contraseña

Se implementa un sistema para permitir al usuario cambiar de contraseña en caso de que se le olvide o quiera cambiarla. El sistema es similar al que se utiliza en la mayoría de páginas web que manejan cuentas. Primero se le pide al usuario que introduzca el correo de su cuenta (Figura

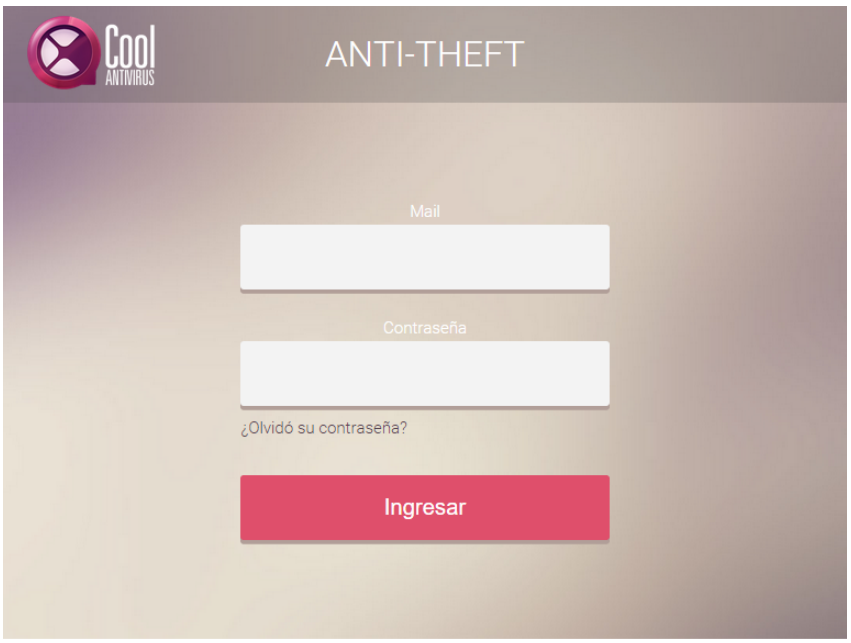

Figura 4.17: Página de autenticación de la aplicación web.

4.19). Cuando el usuario lo introduce, se crea un token que se le envía por correo mediante la librería *tornadomail* (https://github.com/equeny/tornadomail) y que se introduce además en su entrada de la base de datos.

En el correo que recibe el usuario se encuentra con un enlace a la página para cambiar su contraseña (Figura 4.20). En la URL viene incluido el token y en los campos introducirá la nueva contraseña. Si el token que recibe el servidor desde ese formulario se corresponde con el que contiene su entrada en la base de datos, se efectúa el cambio (Figura 4.22). Si no se corresponde, se muestra un mensaje de error (Figura 4.21).

4.2.4. Localización

Realizar la localización del teléfono desde la pagina web es muy similar a lo visto en el apartado 4.1.2. La diferencia radica a la hora de transmitir los datos, ya que no se muestran por pantalla sino que se envían al servidor. La pestaña de pagina web para la localización es similar a la pantalla que se utiliza para este servicio en la aplicación móvil (Figura 4.23).

El procedimiento que se sigue para tratar estas peticiones es muy similar al que se utiliza en el resto. Se consulta la cookie para utilizar la información de autenticación y encontrar la entrada correspondiente en la base de datos. Haciendo esto se obtiene lo necesario para enviar la notificación al teléfono del usuario, el *ID de registro* del teléfono en el servidor de *Google Cloud Messaging*.

Se empaquetan los datos que indican que se quiere localizar el teléfono y se envían mediante la librería de *python-gcm* al servidor de GCM para que éste lo reenvíe al teléfono correspondiente. El teléfono recibe los datos y actúa conforme a lo indicado en el apartado 4.1.8. Después los

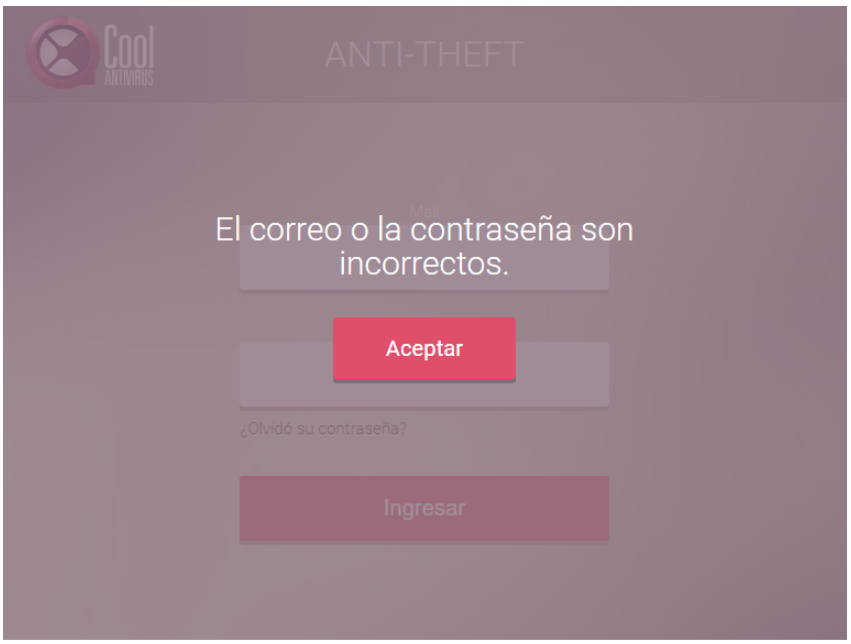

Figura 4.18: Página de fallo en la autenticación de la aplicación web.

datos son enviados desde la aplicación móvil al servidor y este los almacena en la base de datos, desde donde serán leídos para mostrarlos en la página web.

4.2.5. Bloqueo y alarma

Esta sección de la aplicación web permite al usuario enviar notificaciones al teléfono para que inicie el servicio que hace sonar la alarma y/o el servicio que bloquea el teléfono con una contraseña. En la página se muestra un conmutador para cada servicio, añadiendo en el de bloqueo un campo para especificar la contraseña con la que se desea bloquear el teléfono (Figura 4.24).

La forma de proceder es la misma que la utilizada en el apartado anterior, salvo que esta vez los datos que se envían al teléfono indican que los servicios que se han de arrancar son los correspondientes a estas acciones.

4.2.6. Borrado de datos

El borrado de datos se puede realizar exclusivamente desde la aplicación web. La página web dispone de un único conmutador que inicia este servicio (Figura 4.25). Como medida de protección se utiliza una pantalla intermedia de confirmación(Figura 4.26).

El proceso sigue el mismo patrón que los anteriores, enviando esta vez una notificación que le indique a la aplicación móvil que inicie el servicio de borrado.

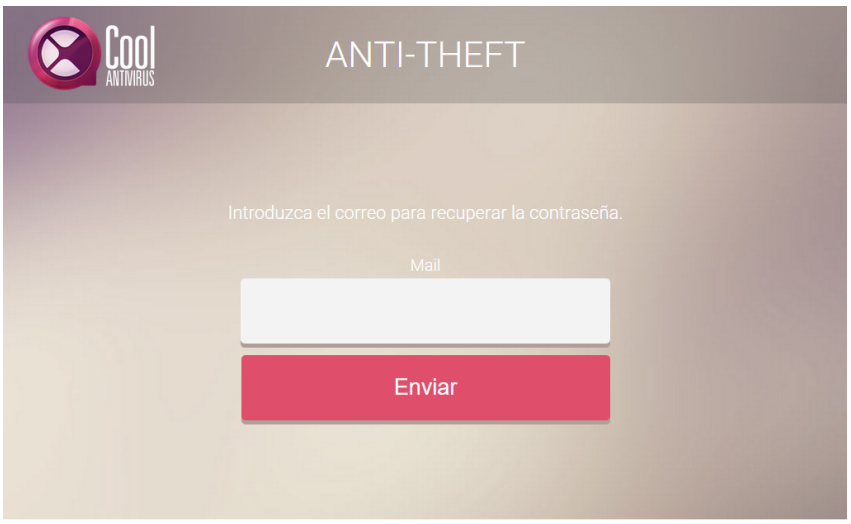

Figura 4.19: Página para solicitar cambio de contraseña.

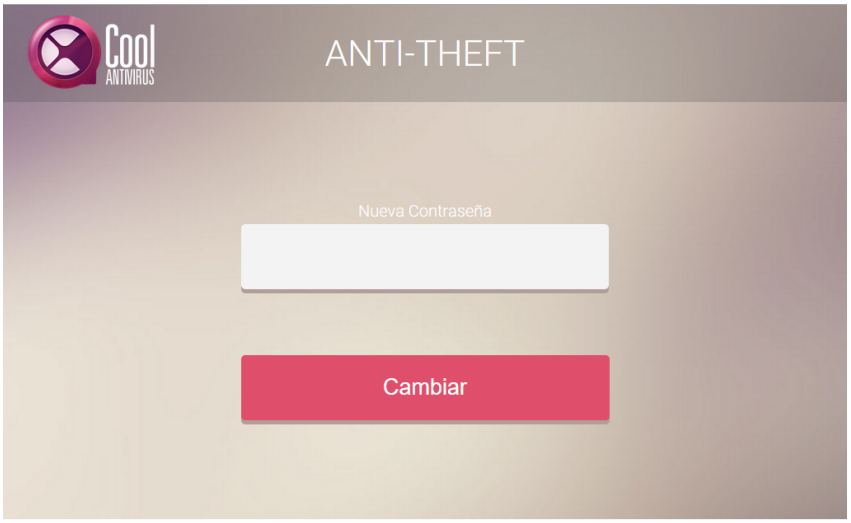

Figura 4.20: Página para cambiar la contraseña.

4.2.7. Pruebas

La aplicación web utiliza muchos de los servicios que se probaron cuando se desarrollaba la aplicación móvil, por lo que las pruebas realizadas en este apartado se centran en probar las conexiones con los teléfonos, comprobar las operaciones sobre la base de datos, probar que se funciona la recuperación de la contraseña de la autenticación web.

4.3. Resultados

Después de haber realizado las pruebas sobre la parte web se había llegado ya al limite de la estancia en la empresa. El resultado que se obtuvo fue, por un lado, una aplicación web

Figura 4.21: Mensaje de fallo en el cambio de contraseña.

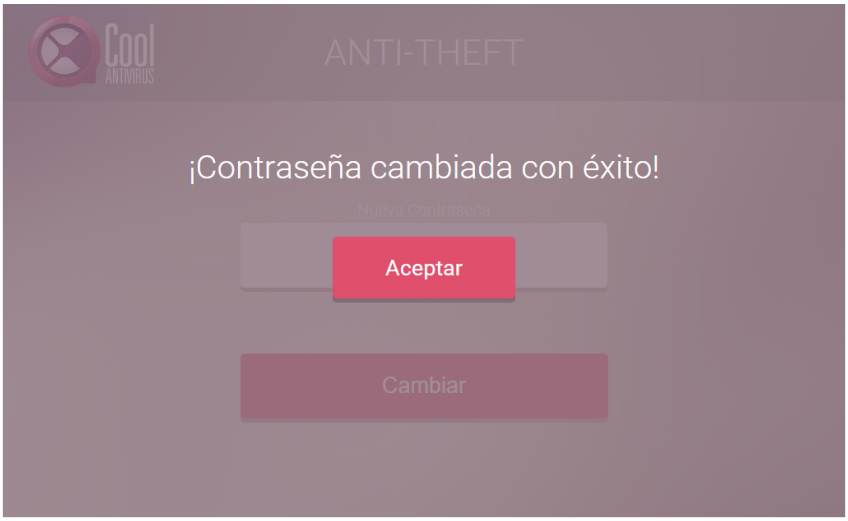

Figura 4.22: Mensaje de éxito en el cambio de contraseña.

que se comunica con el dispositivo para realizar ciertas acciones. Las pruebas que se hicieron utilizaron la red de la empresa como medio de comunicación. Es posible que la parte web estuviera sujeta a cambios al ponerla en funcionamiento en Internet al tener que manejar un número mas elevado de tráfico. Por otro lado, se completó la aplicación para móviles con todas las tareas que se habían especificado con algunas mejoras. No se pudo llegar a integrar con la aplicación principal, quedándose implementado un módulo independiente y funcional.

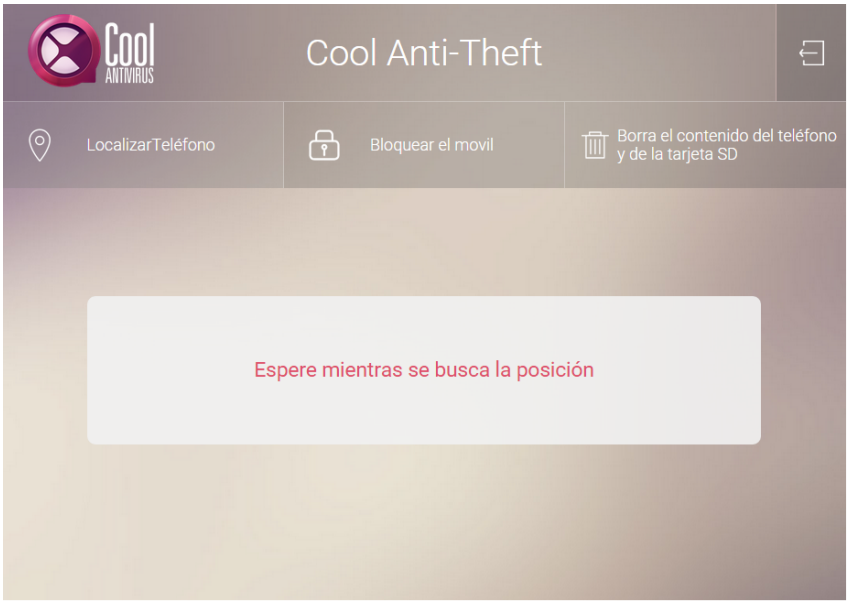

Figura 4.23: Página de localización de la aplicación web.

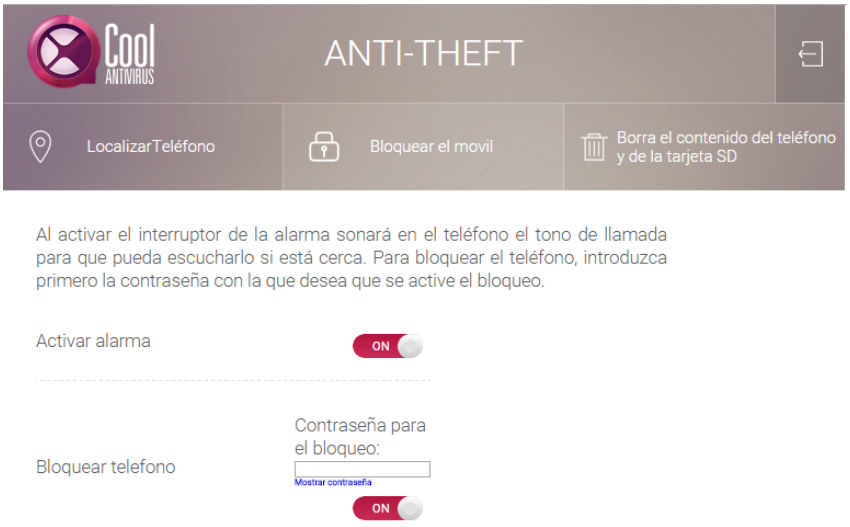

Figura 4.24: Página de administración de la alarma y el bloqueo.

Figura 4.25: Página de borrado de la aplicación web.

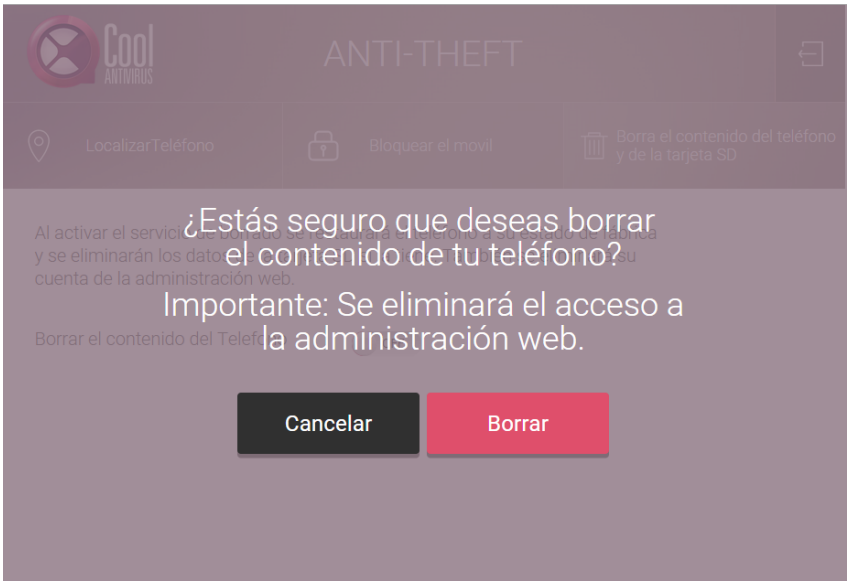

Figura 4.26: Mensaje de confirmación de borrado.

Capítulo 5

Conclusiones

En este capítulo se describirá lo que se ha aprendido en el transcurso del proyecto. Se tendrán en cuenta las asignaturas que han sido útiles y las experiencias que se han trabajado durante el desarrollo.

5.1. Consideraciones técnicas

El proyecto desarrollado en esta memoria ha supuesto una nueva experiencia para el autor. La diversidad de las tareas que debía realizar la aplicación han llevado a la ampliación de conocimientos en diversos temas. Entre las consideraciones mas importantes en el desarrollo se encuentran las siguientes:

- *Planificación*: La estimación de tiempo asignada a cada tarea fue la clave para poder realizar el proyecto dentro del plazo establecido. Si bien no siempre se pudo seguir tal estimación, sirvió de guía para conocer en todo momento el progreso y para realizar los ajustes necesarios en el desarrollo.
- *Solución autónoma de problemas*: Parte de aprender el funcionamiento de las tecnologías o herramientas que se utilizan incluye también ser capaz de solucionar los problemas. En este proyecto el autor ha programado por primera vez una aplicación para móviles Android, por lo que tuvo que ser capaz de solucionar todos los problemas que le surgieron durante el desarrollo debido a la falta de experiencia. Es importante saber solucionar los problemas que se presenten para poder tener éxito y saber afrontar futuros proyectos.
- *Pruebas*: Es una de las partes mas importantes en el desarrollo de software. No solo se deben realizar al final sino también durante el desarrollo. Realizar pruebas sobre cada tarea realizada supuso que no se repitieran los mismos errores en la implementación en otras tareas y, por lo tanto, se ahorró tiempo.
- *Aprovechar código*: Es una parte fundamental en la programación. Al programar diferentes tareas se puede observar que las tareas mas complejas suelen estar formadas por el conjunto de otras mas simples. En este proyecto hubo tareas que necesitaban los mismos servicios

repetidas veces, por lo que compartir esos fragmentos de código entre tareas fue clave para ahorrar en espacio y en trabajo.

- *Uso de APIs*: Durante el desarrollo del proyecto el autor comprendió la utilidad de utilizar APIs para realizar tareas que hubieran sido costosas de programar y hubieran requerido un mayor número de recursos. Su uso ha significado disponer de abundante documentación y soluciones a problemas típicos.

5.2. Consideraciones académicas

En el desarrollo del proyecto se utilizaron una gran cantidad de conceptos y procedimientos tratados en asignaturas cursadas en la carrera. Entre las mas relevantes se encuentran las siguientes:

- Asignaturas relacionadas con la programación como: *EI1004*, *EI1008*, *EI1013*, etc. Estas asignaturas enseñaron al autor como programar y diversos lenguajes de programación como Java, el que se utiliza principalmente en este proyecto.
- Asignaturas relacionadas con las redes como la *EI1015* y la *EI1051*. En estas asignaturas se adquirieron conocimientos que fueron útiles a la hora de conectar la aplicación móvil con la aplicación web.
- Asignaturas relacionadas con las bases de datos como la *EI1020* y la *EI1052* enseñaron conceptos necesarios en este proyecto para saber como manejar la base de datos del servidor.
- La asignatura de seguridad informática *EI1034* enseñó los fundamentos para crear conexiones seguras con certificados que se utilizan en este proyecto para conectar el dispositivo con el servidor.
- La asignatura de tecnologías web *EI1042* y la de tecnologías emergentes *EI1053* proporcionaron los fundamentos las mas útiles para el desarrollo de toda la parte web del proyecto.

5.3. Consideraciones finales

El proyecto ha supuesto el primer contacto del autor con el mundo laboral. Se ha aprendido una nueva forma de trabajar colaborando con personas de diferentes departamentos en el desarrollo de la aplicación.

También ha sido el primer contacto con la programación para dispositivos móviles. Se han aprendido una gran cantidad de conceptos durante la investigación que serán de gran utilidad para futuros desarrollos tanto de aplicaciones móviles como de aplicaciones en general.

Por último remarcar que utilizar gran parte de los conocimientos adquiridos en la carrera ha sido una experiencia muy satisfactoria y un factor clave para poder finalizar el proyecto.

Apéndice A

Plataformas

A.1. *Android*

Figura A.1: Logo de Android

Android es un sistema operativo basado en el núcleo de Linux. Está orientado principalmente para dispositivos móviles con pantalla táctil. Es capaz de procesar las entradas táctiles del dispositivo como tocar, deslizar, pellizcar, etc., para manipular los objetos de la pantalla y mostrar una respuesta al usuario.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientadas a objetos sobre el núcleo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo de ejecución.

Las aplicaciones pueden ejecutar servicios que llevan cabo tareas en segundo plano, sin interaccionar con el usuario. Para que un servicio está activo, también lo ha de estar la aplicación que lo ejecuta. La plataforma Android ofrece una gran cantidad de servicios predefinidos, disponibles regularmente a través de la clase Manager.

Apéndice B

Herramientas

B.1. *Android Studio*

Figura B.1: Logo de Android Studio

Android Studio es un entorno de desarrollo integrado para la plataforma Android. Fue desarrollado por Google y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. Utiliza una licencia de software libre Apache 2.0, está programado en Java y es multiplataforma. Sus principales características son:

- Renderización en tiempo real
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Soporte para construcción basada en Gradle.
- Refactorización especifica de Android y arreglos rápidos.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones, y otros problemas.
- Plantillas para crear diseños comunes de Android y otros componentes.
- Soporte para programar aplicaciones para Android Wear.
- Posibilita el control de versiones accediendo a un repositorio desde el que poder descargar Mercurial, Git, Github o Subversion.

- Vista previa en diferentes dispositivos y resoluciones.
- Editor de diseño que muestra una vista previa de los cambios realizados directamente en el archivo xml.

B.2. *Google Developers Console*

Es una herramienta para desarrolladores que permite controlar de tráfico de datos, la autenticación y la información facturación para las APIs de Google que utiliza un proyecto.

En la *Google Developers Console*, un proyecto es la colección de configuraciones, credenciales y metadata acerca de las aplicaciones con las que se trabaja que utilizan las APIs de desarrollo de *Google* o los recursos de *Google Cloud*. La consola permite el control sobre estos aspectos del proyecto, como generar credenciales para las APIs, activar ciertas APIs, etc.

Hay dos maneras de identificar un proyecto:

- El *ID del proyecto* es un identificador único y se utiliza solo en la Developers Console. Se puede elegir siempre que no se repita con el de otro proyecto existente o bien se puede utilizar el asignado por defecto.
- El *número del proyecto*, que se asigna automáticamente cuando se crea el proyecto.

Apéndice C

APIs de *Google*

C.1. *Google Cloud Messaging*

Figura C.1: Logo de Google Cloud Messaging

Google Cloud Messaging (GCM) es un servicio que permite a los desarrolladores enviar mensajes en diferentes plataformas, como Android, iOS y Chrome. Por ejemplo, un servidor puede enviar mensajes directamente a un único dispositivo, a un grupo de dispositivos o a dispositivos suscritos a determinadas normas. De la misma manera, la aplicación de un dispositivo puede enviar mensajes directamente a un servidor y a los dispositivos que pertenezcan al mismo grupo.

Una implementación utilizando GCM está compuesta por un servidor de conexión de Google, un servidor en un entorno propio que se comunica con el servidor de Google mediante el protocolo HTTP o el XMPP, y una aplicación cliente

C.2. *Google Maps Android API*

Es una de las APIs de *Google Maps* orientada a su uso en aplicaciones para dispositivos *Android* que permite incluir mapas basados en datos de *Google Maps*. La API gestiona automáticamente el acceso a los servidores de *Google Maps*, la descarga de los mapas, el mostrarlos en la aplicación y la respuesta a los eventos realizados sobre el mapa. También se pueden utilizar las llamadas a la API para añadir marcadores, polígonos y capas al mapa básico. Estos elementos

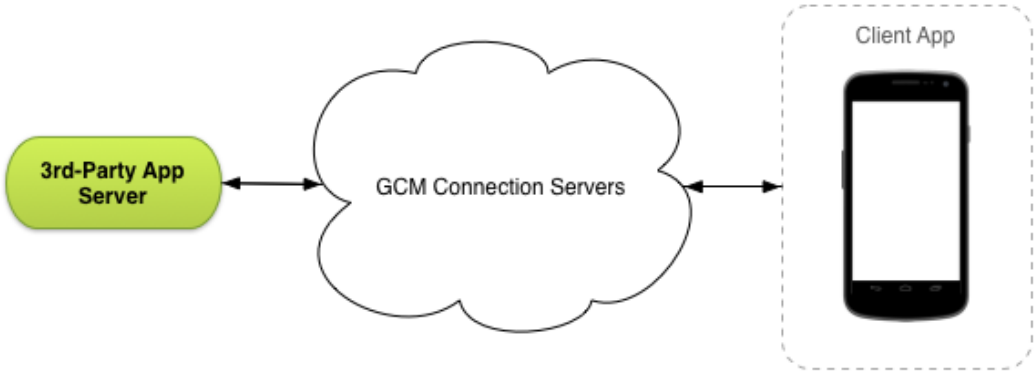

Figura C.2: Estructura de Google Cloud Messaging

proporcionan información adicional y permiten la interacción del usuario con el mapa.

La API permite añadiro los siguientes elementos gráficos:

- Iconos anclados a una determinada posición del mapa. (Marcadores).
- Conjuntos de segmentos de lineas. (Polílinea)
- Segmentos agrupados y cerrados (Polígonos)
- Imágenes en mapa de bits.
- Conjuntos de imágenes mostradas sobre las áreas que componen el mapa.

Apéndice D

Protocolos

D.1. *Simple Mail Transfer Protocol* (SMTP)

El *Simple Mail Transfer Protocol* (SMTP) o "protocolo para transferencia simple de correo", es un protocolo de red utilizado para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos (PDA, teléfonos móviles, etcétera).

El funcionamiento de este protocolo se da en línea, de manera que opera en los servicios de correo electrónico. Sin embargo, este protocolo posee algunas limitaciones en cuanto a la recepción de mensajes en el servidor de destino (cola de mensajes recibidos). Como alternativa a esta limitación se asocia normalmente a este protocolo con otros, como el POP o IMAP, otorgando a SMTP la tarea específica de enviar correo, y recibirlos empleando los otros protocolos antes mencionados (POP O IMAP).

SMTP es un protocolo orientado a la conexión basado en texto, en el que un remitente de correo se comunica con un receptor de correo electrónico mediante la emisión de secuencias de comandos y el suministro de los datos necesarios en un canal de flujo de datos ordenado fiable, normalmente un protocolo de control de transmisión de conexión (TCP). Una sesión SMTP consiste en comandos originados por un cliente SMTP (el agente de inicio, emisor o transmisor) y las respuestas correspondientes del SMTP del servidor (el agente de escucha, o receptor) para que la sesión se abra y se intercambian los parámetros de la sesión.

Apéndice E

Frameworks

E.1. *Tornado*

Tornado es un framework web escrito en Python y una librería para el manejo de redes de forma asíncrona. Mediante el uso de la red no-bloqueo de E / S, Tornado puede escalar a decenas de miles de conexiones abiertas, lo que es ideal para *LongPolling*, *WebSockets*, y otras aplicaciones que requieren una conexión de larga duración a cada usuario.

Figura E.1: Logo de Tornado

Tornado está dividido en cuatro componentes principales:

- Un framework web.
- Implementaciones HTTP del lado el cliente y del lado del servidor (*HTTPServer* y *AsyncHTTPClient*)
- Una librería de red asíncrona que sirve para construir las partes HTTP y crear otros protocolos.
- Una librería de corrutinas que permite escribir código asíncrono de forma directa que mediante callbacks.