Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de la aplicación móvil Android para la plataforma de monitorización y seguridad de redes redBorder

Autor: Ángel María de Miguel Meana

Tutor: Pablo Nebrera Herrera

Dep. Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de la aplicación móvil Android para la plataforma de monitorización y seguridad de redes redBorder

Autor:
Ángel María de Miguel Meana

Tutor:
Pablo Nebrera Herrera
Profesor Titular

Dep. Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado: Desarrollo de la aplicación móvil Android para la plataforma de monitorización y seguridad de redes redBorder

Autor: Ángel María de Miguel Meana
Tutor: Pablo Nebrera Herrera

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

RedBorder Mobile ha sido mi primer acercamiento al mundo laboral. No puedo estar más feliz tras esta experiencia, que me ha ayudado a aprender, me ha propuesto nuevos retos, y me ha hecho conocer personas con las que he entablado una gran amistad.

Primero agradecer a mi familia el apoyo que me han dado siempre, ayudándome cuando las cosas no han salido como esperaba y celebrando los pasos que he ido dando en estos años. Gracias a mis padres por su educación, dedicación y cariño, me habéis enseñado que con esfuerzo y ganas todo se consigue. Gracias por escucharme y aconsejarme siempre, sois mi ejemplo a seguir en esta vida.

También quiero mencionar a mi Tío Eloy, que ha sido como un segundo padre para mí, demonstrándome que es mejor pararte un momento a pensar las cosas y que es muy importante disfrutar. Ya sé que 18 años no se tienen otra vez, pero no te preocupes, tomé tu consejo y los he disfrutado al máximo.

Gracias Cristina, por estar siempre a mi lado y apoyarme en cualquiera de mis alocadas ideas. Es muy importante para mi el sentir ese apoyo y poder compartir contigo cualquier cosa que se me pase por la cabeza.

Gracias a mis amigos de Jerez, que aunque estemos en la distancia, seguimos encontrándonos y charlamos como si aún cada fin de semana echaramos las horas en la *escalerita*. También a mis compañeros de facultad, Antonio, Andrés, Carlos, Pedro, Merce, Pablo... con los que durante estos años he compartido experiencias, frustaciones y éxitos. La carrera es difícil pero con amigos cerca todo se lleva mucho mejor.

Gracias a los maestros, sin ellos no sería posible que a día de hoy pueda convertir mis ideas en algo tangible. En especial a Pablo, mi tutor y amigo, que confió en mi para desarrollar este proyecto.

Por último, quiero agradecer al equipo de redBorder su trato. Entrar en un equipo de trabajo siendo tan joven y ser escuchado es algo muy importante. Gracias a Jose Antonio, Clara y Carlos por enseñarme a comprender mejor Ruby entre charlas y tonterías. Me alegro enormemente de formar a día de hoy parte de esta familia.

Ángel María de Miguel Meana
Sevilla, 2015

Resumen

Los dispositivos móviles son parte del día a día de cualquier persona. Durante los últimos años, muchas empresas han ampliado su estrategia a estos dispositivos, ya que al llevarlos consigo durante la mayor parte del tiempo, las personas tienden a hacer un uso constante de ellos.

RedBorder® es un servicio de monitorización y gestión de eventos orientado a seguridad, y como otras empresas, ha incluido en su estrategia la necesidad de desarrollar una aplicación móvil para agregar más valor al producto.

De esta idea nace redBorder Mobile, una aplicación móvil para Android® integrada con su plataforma, para la visualización de datos y notificación en tiempo real de cualquier anomalía en el sistema monitorizado por redBorder®.

Índice Abreviado

Índice

1 Introducción

Redborder es un sistema de gestión de seguridad de código abierto basado en tecnologías de Big Data, desarrollado por ENEO Tecnología. Este proyecto se divide en dos versiones: una versión ‘‘Community’’ de código libre; y una versión ‘‘Enterprise’’ orientada a empresas.

RedBorder cuenta con un gestor web que permite tanto la visualización de datos como la gestión de la infraestructura. Este gestor es muy completo, ofreciendo mucha variedad opciones a la hora de representar los datos, lo que engloba una gran cantidad de casos de uso. No obstante, en dispositivos móviles el ofrecer demasiadas opciones puede tener un efecto negativo, ya que el espacio disponible es menor y por lo tanto es más complicado navegar.

1.1 Motivación y objetivos

Definición 1.1.1 (Proactividad) La proactividad es la capacidad de prevenir errores mediante detección temprana de anomalías, evitando que lleguen a convertirse en fallos.

Según Fault, Configuration, Accounting, Performance and Security (FCAPS) [1], en la gestión de fallos es imprescindible abarcar cualquier problema desde dos enfoques:

- *Proactividad*: según la Definición 1.1.1, la alerta temprana de problemas de seguridad como tráfico anómalo desde un equipo o alta intensidad de tráfico hacía un servidor, es importante a la hora de lidiar con ataques malintencionados a nuestra red.
- *Reactividad*: una vez se ha detectado un problema, las alarmas saltan y cada minuto es crucial para mitigar un ataque. Por ello es necesaria una rápida notificación a los encargados de la gestión de la red.

La falta de visibilidad y la lentitud a la hora de responder ante un problema puede acarrear problemas económicos y de imagen de la propia empresa. Las pérdidas ecónomicas pueden ser directas, debido a un robo de cuentas bancarias o indirectas, debido a la pérdida de confianza de los usuarios.

Es en este escenario cuando entra en juego la idea de `redBorder Mobile`. En la actualidad no hay persona que no salga de su casa con un dispositivo móvil inteligente en el bolsillo. Aprovechando esta ventaja, se presenta la posibilidad de desarrollar una aplicación móvil que permita al usuario tener un cierto control sobre los sistemas de gestión de redBorder.

RedBorder Mobile se enfoca desde el punto de vista de la visualización de datos y la explotación de las notificaciones en tiempo real. Entre los miembros del equipo de redBorder y el autor del proyecto diseñamos una interfaz adaptada a pantallas móviles y tabletas en la cual representar los datos más relevantes. Como se explicará más adelante, para que el usuario se familiarize con la aplicación móvil rápidamente, la estructura de las vistas es similar a la del gestor web.

El motivo de ignorar la configuración es debido a la complejidad de redBorder, la cual dificultaría su manejo a través de una interfaz móvil. Introducir gran cantidad de datos en una interfaz tan pequeña puede favorecer a que el usuario cometa fallos causando la inestabilidad del sistema.

De este modo, planteamos un claro objetivo: ofrecer a un usuario de redBorder la posibilidad de tener en su mano la información relevante de su infraestructura en los casos en los que este no disponga de un ordenador, y ser notificado en el momento en el que ocurra una irregularidad en su red.

[1] FCAPS es un modelo de gestión de redes de telecomunicaciones desarrollado por la ISO

1.2 Plataforma móvil

Dentro del marco de plataformas móviles encontramos una gran variedad con una clara dominancia de los sistemas operativos Android® e iOS®. La primera intención del equipo es el desarrollo de una aplicación multiplataforma utilizando el método de la encapsulación de aplicaciones web como aplicaciones móviles nativas.

Durante dos semanas, trabajamos en el desarrollo hasta que encontramos un problema de rendimiento. La aplicación debe de soportar un gran procesamiento de datos, por ejemplo, estimamos que para mostrar una gráfica deberíamos de aplicar escalas y procesamientos de fecha a más de 200 eventos por petición. Dada la velocidad de los procesadores de javascript, se hacía inviable desarrollar una aplicación ágil y rápida.

En este punto, tomamos la decisión de desarrollar una aplicación nativa. Barajamos la plataforma que escogeríamos para el desarrollo y tras una reunión, se pivotó la idea hacía el desarrollo de la aplicación para la plataforma de Android®. La principal causa de esta decisión fuero los comentarios de los clientes actuales de redBorder, entre los cuales, había una clara dominancia del sistema operativo Android®.

1.3 Metodología de trabajo

La integración de una aplicación móvil con una plataforma requiere del trabajo en equipo de los desarrolladores de ambas partes. Para poder realizar esta tarea de manera sincronizada, se acordó la manera de interconectar la aplicación móvil con el gestor web para poder acceder de manera rápida y eficiente a los datos. Los detalles se especifican en el tercer capítulo de este documento.

Para trabajar en equipo, se han realizado reuniones presenciales cada 15 días, para acordar nuevas vías de mejora y hablar sobre el estado del proyecto. También se ha utilizado la herramienta de comunicación vía voz sobre IP, `Skype`.

2 RedBorder

RedBorder es una aplicación compleja que engloba muchos conceptos. Es importante dedicar un poco de tiempo a definir algunos de estos, ya que serán necesarios posteriormente.

El gestor web está dividido en diversos módulos que se adaptan a las necesidades de cada cliente. Estas necesidades pueden ir desde la monitorización interna de una red empresarial para la prevención de problemas de seguridad; hasta la gestión de una red WiFi de un centro comercial.

Ambos ejemplos definen dos casos de usos bastante diferentes que son cubiertos gracias a las distintas partes de las que está compuesta la aplicación. En la Figura 2.1 se muestra una captura de pantalla de un supuesto escenario de gestión de puntos de acceso WiFi en la ETSI.

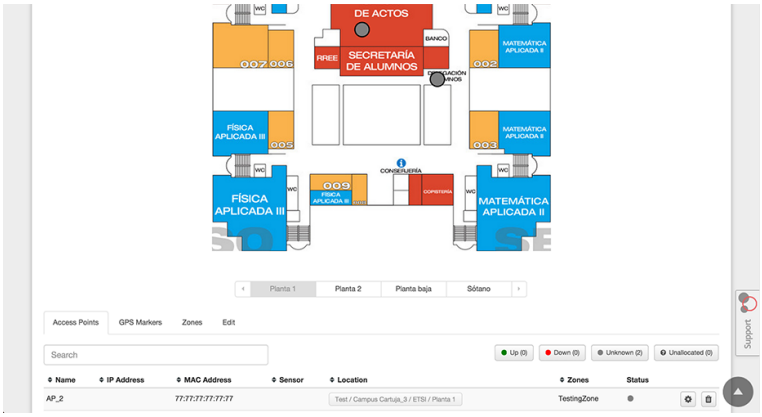

Figura 2.1 Ejemplo de la aplicación para la gestión de puntos de acceso WiFi.

Actualmente, los módulos que incluye RedBorder son:

- *redBorder IPS*: sistema IDS/IPS para detección y actuación frente a problemas de seguridad.
- *redBorder Flow*: colector de tráfico de red, correlando la información recibida con la proporcionada por los enrutadores WiFi (como posicionamiento). De esta manera obtenemos una visión tanto a nivel de red de datos como espacial.
- *redBorder Monitor*: monitor de recursos de las máquinas conectadas. Ofrece información de capa física como la carga de CPU o la velocidad de los ventiladores; y de la capa de red, como los paquetes recibidos y enviados.

2.1 Eventos

Definición 2.1.1 (Evento) conjunto de información asociada a un suceso real que se almacena y mantiene una estructura establecida. Se puede realizar una analogía con la programación orientada a objetos, siendo la estructura la clase y el evento un objeto instanciado.

Los eventos son la base de RedBorder, ya que representan la información recolectada, procesada y almacenada. Cada tipo de evento está organizado en columnas, ofreciendo cada una de ellas información adicional sobre el suceso.

Por ejemplo, podemos definir que cada evento de tráfico de red contendrá información sobre la dirección IP origen y la dirección IP destino, quedando así definida la estructura. Las direcciones son las columnas de este tipo de eventos, mientras que el conjunto de dirección origen igual a 192.168.1.1 y destino 192.168.1.2 sería un evento de tráfico.

2.1.1 Druid

Definición 2.1.2 (Granularidad) en almacenamiento de datos, se define como una unidad de detalle de los datos. Partiendo de datos que pueden ser agrupados, la granularidad especifica el nivel de agrupación de estos datos. A mayor granularidad, mayor agrupación y menor detalle. Cuanto más baja es la granularidad, menor es la agrupación y mayor el detalle.

Los eventos son almacenados en una base de datos de tipo On-Line Analytical Processing (OLAP), llamada Druid®. No es objetivo de este trabajo el describir este tipo de base de datos, no obstante, es interesante explicar las bases de Druid® para comprender algunas ideas.

La gran ventaja de este tipo de bases de datos, es que además de almacenar eventos, realizan cálculos sobre ellos creando resúmenes más rápidos de consultar. En Druid® el término de granularidad está relacionado con el tiempo. La granularidad mínima es el mínimo tiempo de separación entre eventos, los que define el detalle máximo que podemos obtener. Es decir, a menor granularidad, mayor coste computacional y mayor necesidad de almacenamiento.

En redBorder la granularidad mínima de los eventos es de un minuto, por lo que por su definición, podremos obtener como máximo el minuto en el que ocurrió un evento, pero no el segundo. Esta granularidad cubre la gran mayoría de los casos de uso y a su vez es suficientemente rápida para que las peticiones de datos no tarden excesivamente.

De esta manera, cuando solicitamos datos, especificamos el intervalo de tiempo en el que queremos obtener los eventos, así como la granularidad.

2.2 Gestor web

El gestor web es la herramiento principal para la consulta de datos y gestión de la infrastructura de redBorder. Este incluye numerosas vistas de configuración, aunque en este caso nos centraremos las dos vistas utilizadas en la aplicación móvil.

2.2.1 Vista de RAW

Definición 2.2.1 (RAW) formato de ficheros que representa información tal y como es recibida. Muy utilizado en el lenguaje fotográfico para referirse a imágenes sin compresión alguna.

Definición 2.2.2 (Vista RAW) vista en la cual se muestran los eventos tal y como se reciben, ordenados por el tiempo de llegada e incluyendo los datos asociados.

Una manera de visualizar los eventos almacenados en el sistema es tal y como llegan a este, sin nungún tipo de filtro visual ni agrupación. Partiendo de la Definición 2.2.2, este tipo de vista nos permite visualizar en tiempo real todo lo que esta sucediendo en nuestro sistema. En el momento en el que agrupamos los elementos por columnas o ejercemos una ordenación o filtro, perdemos la visión global de lo que ocurre en nuestro sistema. Por ello, es especialmente útil para conocer la evolución actual de nuestro sistema, permitiendo ver los eventos que estamos recibiendo.

En la figura Figura 2.2 tenemos un ejemplo de la vista RAW en el gestor web.

2.2.2 Vista de Tops

Definición 2.2.3 (Top) valor que está por encima de otros basándose en una determinada referencia.

Definición 2.2.4 (Vista Tops) vista en la que se agrupan los eventos por columnas, ordenándolos descendentemente en función del número de eventos o tráfico generado.

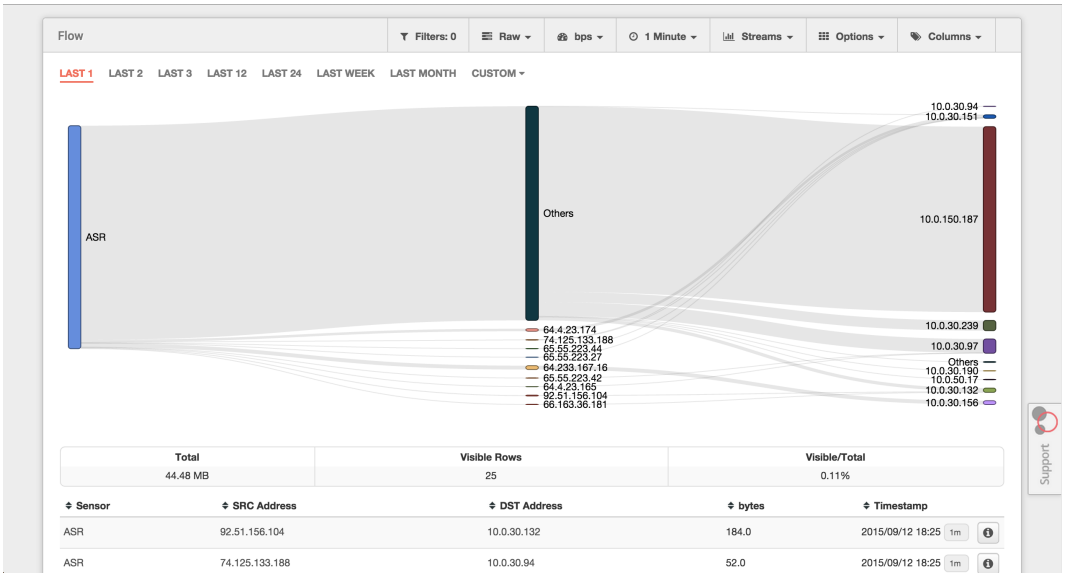

Figura 2.2 Vista de RAW del módulo Flow en el gestor web.

Otra forma de visualizar los datos consiste en priorizar aquellos que han destacado en un periodo de tiempo. Esto nos ofrece la posibilidad de comprobar rápidamente los eventos más destacados en nuestro sistema en función de la columna que deseemos.

En la figura Figura 2.3 tenemos un ejemplo de la vista RAW en el gestor web.

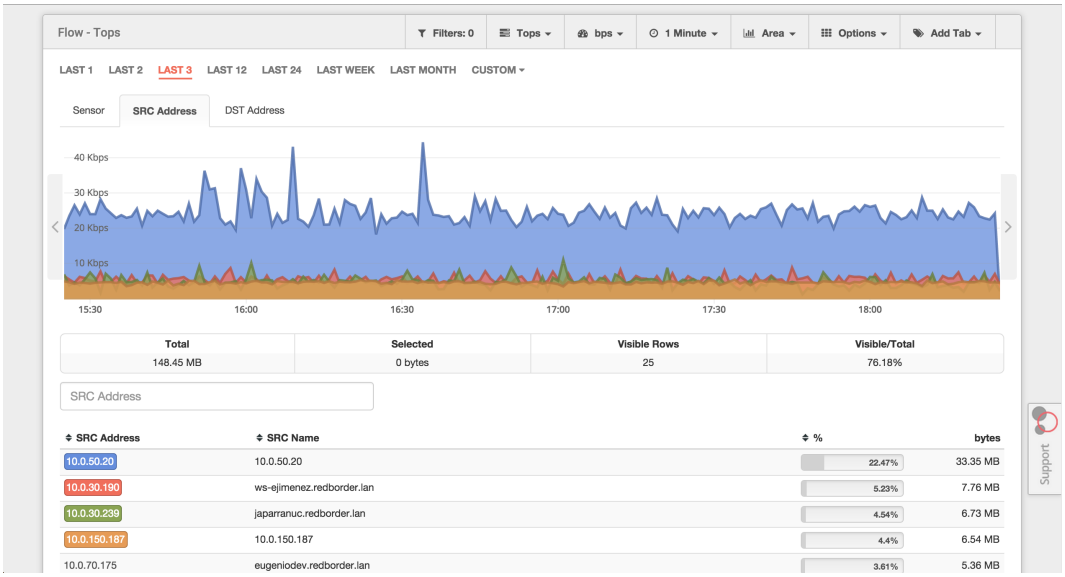

Figura 2.3 Vista de Tops respecto a la columna de dirección IP origen del módulo Flow en el gestor web.

3 Integración con el gestor de redBorder

La recepción de los datos en el dispositivo móvil debe de ser rápida y unificada. Para la definición de la interfaz de programación de la aplicación (más conocida en inglés por las siglas API), decidimos escoger una interfaz del tipo Representational State Transfer (REST).

3.1 REST

Definición 3.1.1 (Recurso) estructura de datos interrelacionados entre si que representan un ente relevante de información para un usuario de la aplicación.

Definición 3.1.2 (REST) define un conjunto de principios de arquitectura de información para diseñar servicios en línea basados en los recursos que ofrece dicho sistema y en los estados de estos. Para ello se basa en los distintos métodos definidos en el protocolo Hypertext Transfer Protocol (HTTP) (GET, POST, PUT, PATCH, DELETE) asignándolos las acciones de obtener, crear, actualizar y borrar los recursos. Los recursos son expuestos mediante una URL y ofreciendo su respuesta en formato XML o JSON.

La simpleza y la orientación a los propios recursos contenidos en el servidor, han sido las razones por las que se ha elegido esta guía para el desarrollo de la interfaz de acceso. Conviene destacar que gran parte de la potencia de REST se pierde en este proyecto, ya que solo se aprovecha la obtención de datos (peticiones de tipo GET) puesto que no es el objetivo de la aplicación móvil cambiar el estado en los recursos almacenados en el servidor. Se detallará más adelante algunas peticiones concretas que necesitan ser de tipo POST.

Existen otras alternativas como el lenguaje de descripción de servicios web (más conocido por su acrónimo en inglés, WSDL). Este tipo de soluciones se basan en el protocolo SOAP, definido por la organización W3C, actualmente en desuso debido a su complejidad.

3.2 Acceso a los recursos

Se define la interfaz de acceso a la aplicación a través de la dirección base `http(s)://<dirección del gestor>/api/<versión utilizada>/`. Un ejemplo de dirección base válida es:

```
http://gestor.empresa.com/api/v1/
```

Como se especifica en REST, las rutas web deben de hacer referencia a los recursos y a las posibles acciones estos.

Dentro de redBorder encontramos actualmente tres tipos de recursos: IPS, Flow y Monitor. Estos pueden devolverse de varias maneras en función del tipo de petición que se realice.

3.2.1 Identificación de usuarios en las peticiones

Definición 3.2.1 (Identificador) cadena alfanumérica aleatoria.

Para acceder a cualquier información en el gestor web es necesario iniciar sesión con el nombre de un usuario y su contraseña. En el caso de la aplicación móvil, para evitar la petición de la contraseña del usuario cada vez que se accede a un servidor se ha optado por una autenticación mediante un *identificador*.

La primera vez que un usuario inicia sesión en un servidor desde la aplicación móvil, se solicitan sus credenciales. Después de validar dichas credenciales y en caso de ser correctas, se retorna en la respuesta el identificador del usuario.

Para agregar una mayor seguridad al acceso a la API de redBorder, desde el gestor es posible volver a generar un nuevo identificador cuando sea necesario. Esta acción invalida cualquier petición posterior que contenga el identificador antiguo, por lo que el usuario deberá de volver a introducir las credenciales de acceso en la aplicación móvil.

Este parámetro se enviará con el identificador `auth_token` en la URL de la petición, quedando el formato de todas las peticiones como se indica en la Tabla 3.1.

Tabla 3.1 Formato general de las peticiones.

URL	?auth_token=<identificador>
Tipo de llamada	GET, POST, PUT, PATCH, DELETE
Parámetros	
Identificador	cadena de identificación del usuario

Al estar este parámetro presente en todas las peticiones a excepción de la de inicio de sesión, se obviará su presencia en el formato de las peticiones que se exponen a continuación.

Inicio de sesión

Este petición identifica al usuario dentro del gestor de redBorder. Si la autenticación es correcta, el usuario obtendrá su identificador. El formato de esta petición es el definido en la Tabla 3.2

Tabla 3.2 Formato de la petición de inicio de sesión.

URL	users/login?user[username]=<nombre usuario>&user[password]=<contraseña>&user[gcm_id]=<identificador de notificaciones>
Tipo de llamada	POST
Parámetros	
Nombre de usuario	Cadena de texto con el nombre de usuario o el email
Contraseña	Cadena de texto con la contraseña del usuario
Identificador de notificaciones	Identificador del dispositivo para el envío de notificaciones. Se explica su implementación en la Subsección 5.4.7

3.2.2 Vista RAW

Definición 3.2.2 (Vista) interfaz asociada a una pantalla de la aplicación, por ejemplo, un formulario para iniciar sesión o una lista de elementos. Cada vista está compuesta de multiples elementos gráficos.

En este tipo de visualización la organización de los eventos se realiza en función del tiempo de llegada de estos. En la aplicación móvil, el usuario puede seleccionar la granularidad con la que se mostrarán los eventos en función del detalle que este necesite.

Eventos en formato de lista

Las peticiones de eventos para la vista de lista de RAW siguen el formato expuesto en la Tabla 3.3. En este caso, los módulos de los que podemos obtener los datos son Flow e IPS.

El segundo parámetro es la `granularidad`. Esta se envía codificada, atribuyendo a cada granularidad de tiempo un símbolo comprensible por el gestor. Véase Tabla 3.4

El tercer y cuarto parámetro, `offset` y `end_time`, se utilizan para paginar. Para evitar una gran cantidad de datos en una sola petición, se ha limitado el número de elementos devueltos. Cuando el usuario llega al final de la lista, aparece un icono que indica que se están cargando más resultados mientras la aplicación espera la respuesta del servidor. Para ello el servidor segmenta el tiempo en función de la granularidad y retorna los eventos en grupos de veinticinco elementos.

Tabla 3.3 Formato de peticiones de eventos en lista de tipo RAW.

URL	modules/<modulo>/?granularity=<granularidad>&offset=<desfase>&end_time=<tiempo>
Tipo de llamada	GET
Parámetros	
Modulo	Nombre del módulo del que se desea obtener la información
Granularidad	Cadena de texto con la granularidad
Desfase	Numero de elementos ya recibidos
Tiempo	Tipo entero largo que representa el tiempo. El formato es UNIXTIME

Tabla 3.4 Correspondencia de granularidades y símbolos del gestor.

Granularidad	Símbolo
1 minuto	minute
5 minutos	pt5m
15 minutos	fifteen_minute
30 minutos	thirty_minute
1 hora	hour
8 horas	pt8h
1 día	pt24h
1 semana	pt168h
1 mes	pt720h

En cada respuesta, el servidor indica el instante de tiempo y cuantos elementos ha devuelto. Estos valores son los que han de enviarse en la siguiente petición como `offset` y `end_time`, para que el servidor tenga constancia del evento en el que se quedó.

Eventos como serie temporal

Estas peticiones son utilizadas para representar los datos de manera gráfica. En este caso, el servidor solo devuelve el número de eventos o tráfico generado. Al no retornar ninguna columna, la petición es considerablemente más rápida.

Las peticiones de este tipo tienen el formato definido en la Tabla 3.5.

Tabla 3.5 Formato de peticiones de serie temporal para RAW.

URL	modules/<modulo>/raw_serie?granularity=<granularidad>&start_time=<tiempo inicial>&end_time=<tiempo final>
Tipo de llamada	GET
Parámetros	
Modulo	Nombre del módulo del que se desea obtener la información
Granularidad	Cadena de texto con la granularidad
Tiempo inicial y final	Tipos de entero largo que representan el intervalo de tiempo en el que se debe de calcular la serie temporal. El formato de ambos es UNIXTIME

Detalles de un evento

Para obtener los detalles de un evento concreto, debemos de identificarlo en el servidor. Es necesario incluir en esta petición los distintos valores de las columnas del evento. Incluyendo múltiples valores y el tiempo en el que ha ocurrido dicho evento, podemos delimitarlo y obtener todos sus detalles. El formato de este tipo de peticiones es el definido en la Tabla 3.6.

3.2.3 Vista Tops

En este caso, las peticiones deben de especificar la columna y el intervalo de tiempo. En el gestor, el usuario puede modificar la granularidad, como ocurría en las peticiones de tipo RAW. No obstante, para aliviar la

Tabla 3.6 Formato de peticiones de detalle de un evento.

URL	modules/<modulo>/info?timestamp=<tiempo del evento>&columna_1=<valor>&columna_2=<valor>...
Tipo de llamada	GET
Parámetros	
Modulo	Nombre del módulo del que se desea obtener la información
Tiempo del evento	Tipo de entero largo en formato UNIXTIME con el instante en el que ocurrió el evento
Columnas y valores	Columnas con su correspondiente valor del evento que se quiere consultar

cantidad de opciones de esta vista, se ha obviado este parámetro tomándo un valor por defecto en función del intervalo de tiempo elegido.

Eventos en formato de lista

Para la vista de lista de Tops, el foramto de la petición es el definido en la Tabla 3.7.

Tabla 3.7 Formato de peticiones de eventos en lista de tipo Tops.

URL	modules/<modulo>/<columna>/tops?range=<rango>&offset=<desfase>
Tipo de llamada	GET
Parámetros	
Modulo	Nombre del módulo del que se desea obtener la información
Columna	Cadena de texto indicando la columna sobre la que se consultan los datos
Rango	Cadena de texto indicando el rango de tiempo
Desfase	Numero de elementos ya recibidos

El parámetro `rango`, se envía codificado, atribuyendo a cada intervalo de tiempo un símbolo comprensible por el gestor. Véase Tabla 3.8.

Tabla 3.8 Correspondencia de intervalos de tiempo y símbolos del gestor.

Intervalo de tiempo	Símbolo
Última hora	last_1_hour
Últimas 3 horas	last_3_hour
Últimas 24 horas	last_24
Última semana	last_week
Último mes	last_month
Últimos 3 meses	last_quarter
Último año	last_year

Eventos como serie temporal

En este caso, el formato de estas peticiones es el definido en la Tabla 3.9.

3.2.4 Vista de monitorización de nodos

En esta vista, la aplicación móvil solicita los nodos que puedan disponer de información de monitorización, como pueden ser sensores o los propios gestores.

Lista de nodos

Obtiene el nombre y el identificador de los nodos. El formato de la petición es el mostrado en la Tabla 3.10.

Información de un nodo

Obtiene información asociada a un nodo en concreto. El formato de la petición es el mostrado en la Tabla 3.11.

Tabla 3.9 Formato de peticiones de serie temporal para Tops.

URL	modules/<modulo>/<columna>top_serie?range=<rango>&values=<valor de columna>
Tipo de llamada	GET
Parámetros	
Modulo	Nombre del módulo del que se desea obtener la información
Columna	Cadena de texto indicando la columna sobre la que se consultan los datos
Rango	Cadena de texto indicando el rango de tiempo
Valor de columna	Este valor es opcional. Si se especifica se devuelve la serie temporal para un valor concreto de la columna.

Tabla 3.10 Formato de peticiones de la lista de nodos con información de monitorización.

URL	monitor/
Tipo de llamada	GET

Tabla 3.11 Formato de peticiones de la información sobre un nodo.

URL	monitor/<identificador>/info
Tipo de llamada	GET
Parámetros	
Identificador	Tipo entero con el identificador de dicho nodo en el gestor

4 Interfaz

Una interfaz de usuario es como una broma. No es buena si necesitas explicarla.

Anónimo

La interfaz de una aplicación define en gran medida la percepción que tendrán los usuarios sobre ella. Si una aplicación es estéticamente pobre o compleja de utilizar, probablemente acabe desinstalada. Por supuesto, todo ello debe de ir acompañado de una programación libre de errores.

A esto hay que añadir, que cada sistema operativo de móvil define un conjunto de elementos visuales y una manera de interactuar con las aplicaciones. Una clara diferencia entre los sistemas operativos Android® e iOS® se sitúa en la forma de navegar dentro de una aplicación. Como podemos ver en Figura 4.1, mientras que Android® apuesta por menús laterales y una barra de navegación superior, iOS® propone una barra de navegación inferior.

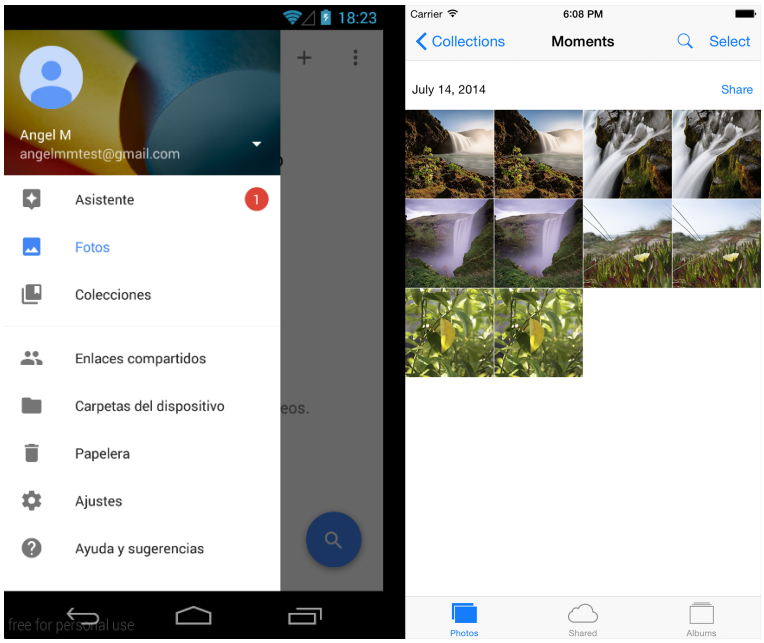

Figura 4.1 Diferencia en la forma de navegar en una aplicación para Android e iOS.

La interfaz planteada respeta la guía interacción de aplicaciones en Android®. Gracias a esta decisión, cualquier usuario acostumbrado a utilizar este sistema operativo se encontrará más cómodo trabajando con la aplicación y se habituará mejor a ella.

4.1 Conceptualización de las vistas

Definición 4.1.1 (Prototipo) en términos de diseño de páginas web y aplicaciones móviles, un *prototipo* es dibujo esquemático que representa la información que contendrá una vista y la disposición de los elementos gráficos de esta sin entrar en profundidad en detalles estéticos. El objetivo de estos es estructurar la información y obtener una visión global. Al obviar la parte estética, son muy fáciles de modificar por lo que son ideales para la conceptualización de ideas en fases tempranas de diseño.

El diseño de la interfaz de una aplicación es un proceso importante y su mejora debe de estar presente durante todo el ciclo de desarrollo de esta, especialmente en las primeras fases. La primera parte del proyecto se dedicó a establecer las necesidades de los usuarios con respecto a la aplicación, y como la interfaz las solventaría.

Para ello realizamos varias reuniones presenciales donde se hablaba entre varios miembros del equipo qué necesidades se pretendía cubrir con la aplicación, cómo lo haría, y qué expectativas debía cumplir. Durante las reuniones se dibujaban bocetos en papel sobre las vistas más importantes.

El siguiente paso fue afianzar las bases de la interfaz. Con la herramienta de dibujo vectorial Sketch®, se realizaron los prototipos de cada una de las vistas. Estos prototipos nos ayudaron a ver más claramente los pros y los contras de los diseños planteados, pudiendo realizar las correcciones necesarias antes de implementarlas en el código.

Cabe destacar que esta manera de trabajar nos ahorró mucho tiempo, ya que modificar un prototipo es una tarea muy sencilla, pero realizar el mismo cambio en el código puede conllevar una inversión considerable de tiempo.

4.2 Internacionalización

En las primeras reuniones que tuvimos sobre el enfoque de redBorder mobile, uno de los requisitos principales fue la traducción al inglés de toda la interfaz. De hecho, se priorizó el inglés antes que el castellano, ya que redBorder se enfoca a un público europeo y estadounidense, donde para la mayoría de las personas el inglés es el idioma nativo.

Dada la facilidad de traducción de las aplicaciones Android®, es muy sencillo realizar una traducción completa de cualquier aplicación. RedBorder mobile no es una excepción, y a pesar de que todas las capturas mostradas de la interfaz están en castellano, la aplicación está traducida integramente al inglés.

4.3 Estructura general

Cada módulo de redBorder tiene asociadas múltiples vistas donde los eventos se muestran de disinta manera para cubrir todas las necesidades de visualización de datos. En algunos casos necesitaremos tener una visión global mientras que en otros será más útil ver los eventos agrupados por columnas.

Siguiendo esta estructura, podemos distinguir cinco partes en nuestra aplicación: inicio de sesión, vista RAW, vista Tops, elementos monitorizados y alarmas; cada una de estas compuesta de una o más vistas. En la Figura 4.2 quedan resumidan las partes de la aplicación y las vistas asociadas a cada una de ellas.

4.3.1 Menu principal

Para el menú principal se ha seguido la guía de estilos de Android®, incluyendolo como un menú lateral izquierdo, recurso utilizado en la mayoría de aplicaciones Android®. Este permite la navegación entre los distintos módulos disponibles en la aplicación móvil, así como cerrar sesión en el servidor actual para conectarse a otro.

4.3.2 Menu lateral de configuración

Este menú está incluido en las vistas Top y RAW. El menú está dividido en dos partes:

- Tipo de vista: permite navegar entre las vistas Top y RAW de un módulo.
- Parámetros de la vista: depende del tipo de vista en la que nos encontremos. En esta parte podremos modificar valores que afectan a los datos recibidos.

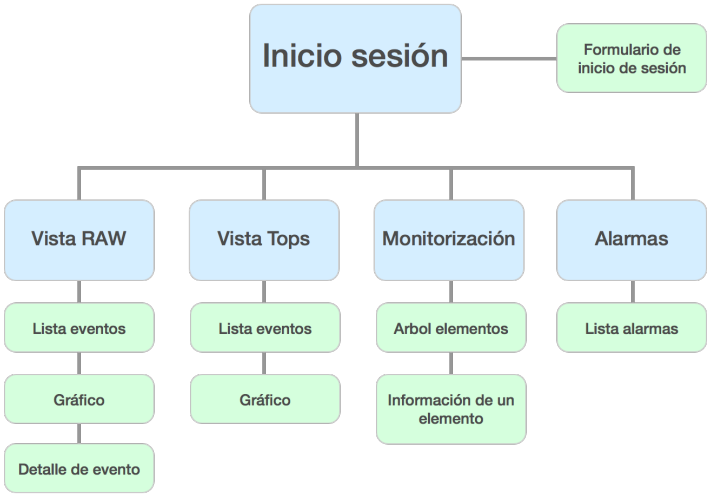

Figura 4.2 Estructura de las vistas de la aplicación redBorder Mobile. Las cajas con el fondo azul representan partes de la aplicación, y las verdes vistas.

La manera de trabajar con este menú, es decir, mostrarlo y ocultarlo, es igual a la del menú principal, solo que este aparece desde la zona derecha. Las vistas que lo incluyen, muestran un icono en la barra de navegación de Android®. Véase la Figura 4.3.

Figura 4.3 Icono de apertura del menú de configuración.

4.3.3 Gráficas

El gestor de redBorder cuenta con una gran diversidad de gráficos. La información puede representarse de múltiples maneras y cada una de ellas nos ayuda a enfocar la visualización de datos desde distintos puntos de vista.

En la aplicación móvil se han utilizado gráficos de línea con el fin de observar tendencias y picos actividad inusual en nuestro sistema. El primer requisito de un gráfico es que su visualización sea lo suficientemente amplia y buena para transmitir correctamente la información.

Para solventar el problema del tamaño, los gráficos se ha incluido en la aplicación en una vista a parte, a la que se accede colocando el móvil en posición horizontal. El posicionamiento horizontal de la pantalla deja un gran espacio para incluir el gráfico pudiéndose leer con facilidad, como vemos en la Figura 4.4

En estos gráficos, el eje de abcisas marca el tiempo en el que han ocurrido los eventos. Es por ello que deslizando las gráficas hacía derecha o izquierda, adelantamos o retrasamos respectivamente la ventana de tiempo actual. La información que representa el eje de ordenadas varía en función del tipo de vista en la que nos encontremos.

4.4 Inicio de sesión

Para consultar los datos de un gestor de redBorder, el usuario debe de autenticarse el sistema. Como se ha comentará en la Subsección 5.5.1, gracias al sistema de cuentas de Android® es posible almacenar las credenciales de un gestor para conectarse directamente sin necesidad de volver a introducirlas.

Por tanto, para iniciar sesión tendremos dos posibles vistas, una que contenga un formulario para introducir las credenciales de usuario y otra con los gestores almacenados en nuestro sistema. Por defecto, si no hay ningún gestor almacenado en el sistema se muestra el formulario para agregar uno. En cambio, si ya existen uno o más gestores almacenados, se muestra la segunda vista con un listado de estos.

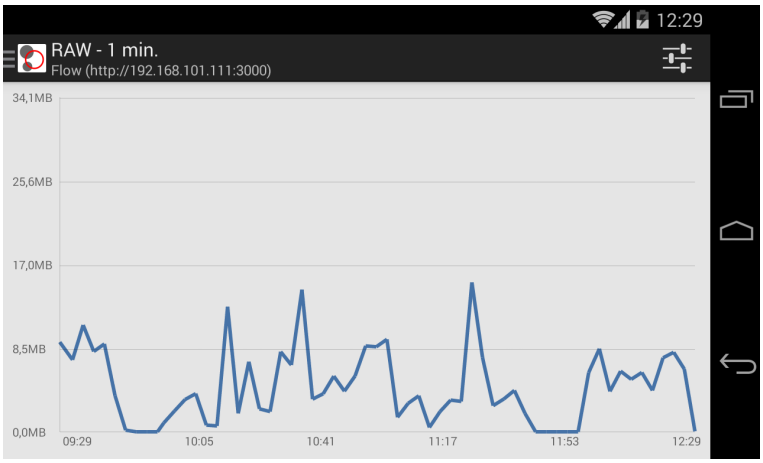

Figura 4.4 Vista de una gráfica con el móvil en posicion horizontal.

4.4.1 Iniciar sesión en un gestor nuevo

Esta sección permite la autenticación de un usuario en un nuevo gestor. Para ello, la vista dispone de un formulario que permite validar la identidad del usuario con el fin de acceder a los recursos en los que tiene permisos.

La identidad del usuario debe ser única, por lo que los campos del formulario deben de ser suficientes para poder identificarlo unívocamente. En primera instancia, la vista se diseñó con dos campos, como podemos ver en Figura 4.5. Los campos fueron:

- *Dirección del gestor*: dirección IP o URL del gestor redBorder a la que se quiere conectar.
- *Identificador de usuario*: cadena aleatoria generada en el gestor que identifica de manera unívoca a un usuario.

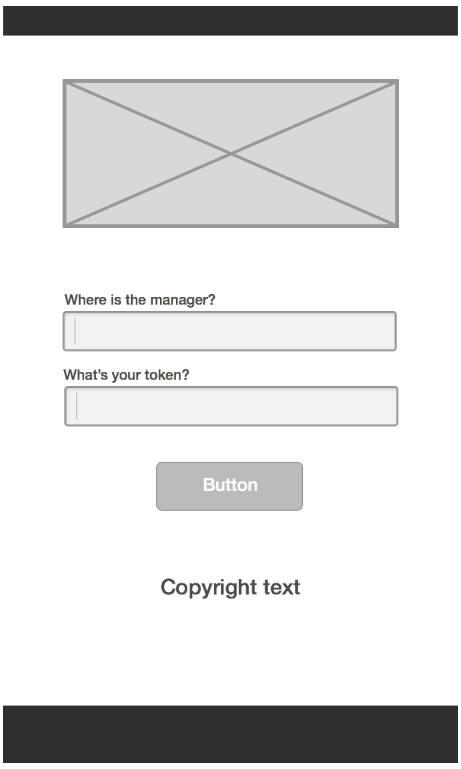

Figura 4.5 Esquemático de la primera versión de la vista de inicio de sesión.

Con estos campos un usuario quedaba totalmente identificado, pero el identificador es excesivamente largo para ser copiado a mano y muy difícil de recordar. No obstante, este identificador es necesario ya que todas las peticiones al gestor lo deben de incluir.

Es por ello que, como se ha comentado en el capítulo anterior, se optó por la autenticación por usuario y contraseña. Una vez que la autenticación es correcta, el gestor devuelve el identificador asociado a dicho usuario y se almacena en el dispositivo mediante el gestor de cuentas de Android®. El prototipo final de esta vista, así como su implementación es el mostrado en la Figura 4.6.

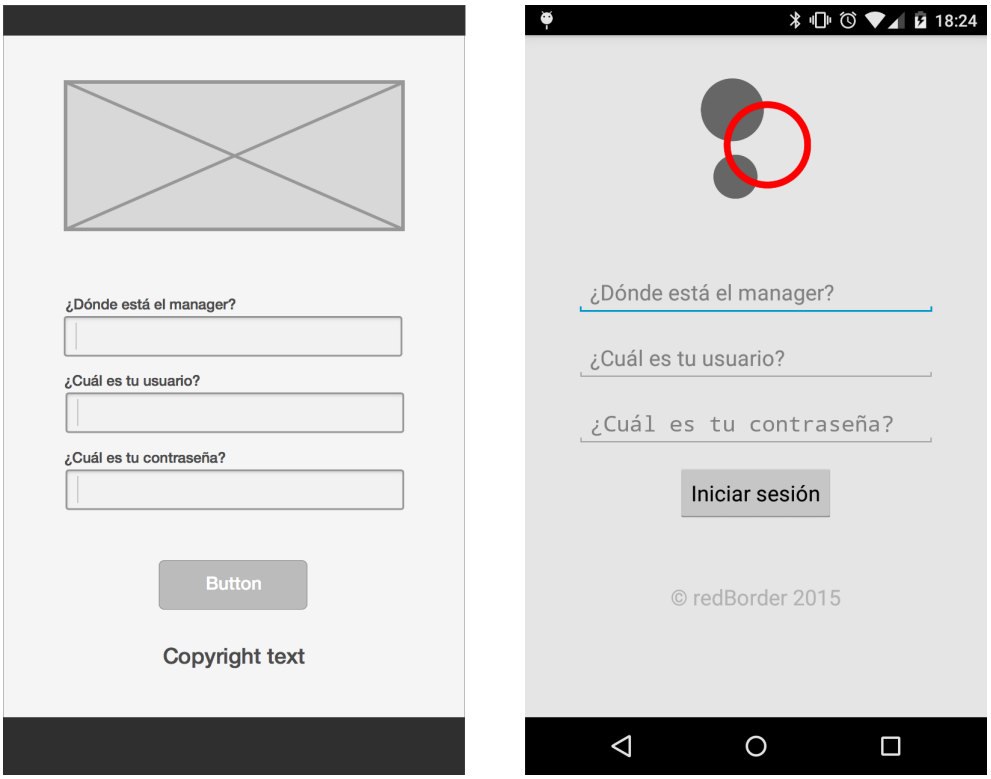

Figura 4.6 Prototipo junto con la implementación final del formulario de inicio de sesión.

Siguiendo la línea de otras aplicaciones, de cara a ayudar a los usuarios a manejar la aplicación se han planteado los títulos de los elementos del formulario como preguntas. Este formulario al tener pocos elementos es un buen escenario para utilizar preguntas como títulos, ya que en otros más extensos esta técnica puede fatigar al usuario.

4.4.2 Inicio de sesión con gestores almacenados

Esta vista mantiene la misma estética que el formulario de inicio de sesión, situando el logo en la parte superior y la información a continuación. Tenemos una lista con las credenciales de los gestores a los que se ha accedido previamente y en la zona inferior un botón para agregar nuevos gestores.

La información ofrecida sobre los gestores es la dirección de estos y el nombre de usuario con el que se accede. Se permite almacenar varios usuarios para un mismo gestor.

Una vez implementado el esquema en la vista, realizamos varias mejoras sobre esta. Se redujo el tamaño del logo para dejar más espacio a la lista y se eliminó el color de fondo de la lista ya que el resultado era más estético. Por otra parte, el botón se personalizó siguiendo la guía de colores de redBorder.

Podemos ver el prototipo y el resultado final de la lista en la Figura 4.7

4.5 Vistas de RAW

Dentro del gestor de redBorder, cada módulo posee su propia vista RAW donde se listan los eventos correspondientes. Esta vista la conforman dos componentes:

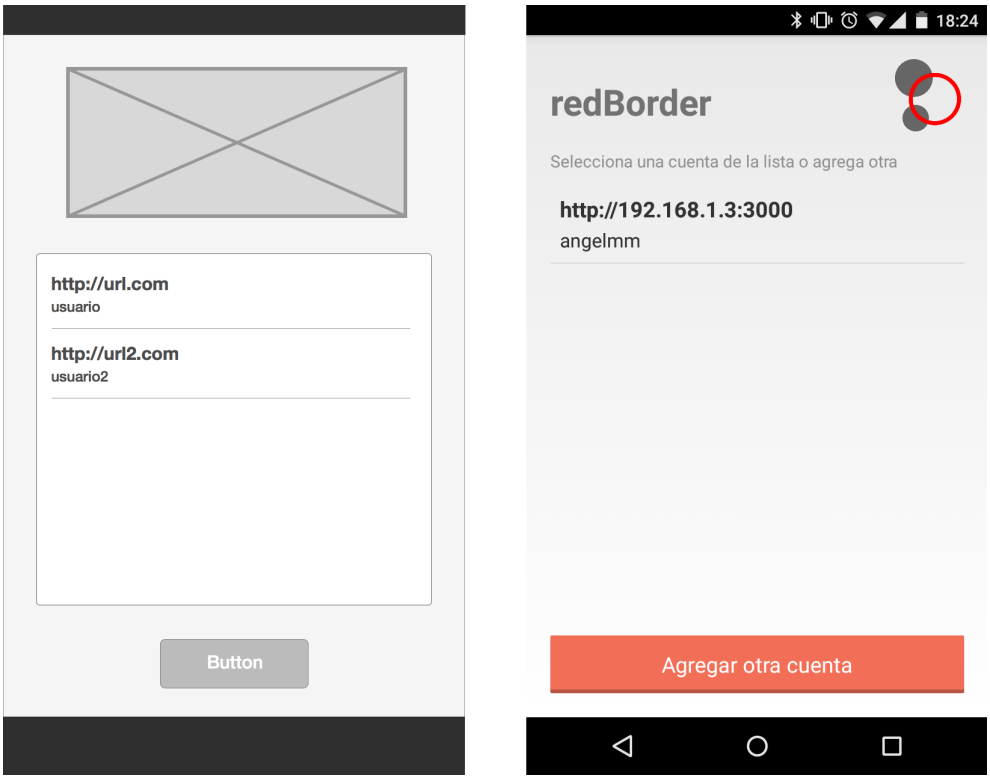

Figura 4.7 Prototipo junto con la implementación final de la vista de inicio de sesión con gestores almacenados.

- *Gráfica*: muestra una gráfica con la evolución del número de eventos con respecto al tiempo de llegada. Esta gráfica nos indica la evolución del sistema durante la ventana de tiempo que tengamos seleccionada.
- *Lista de eventos*: listado con los eventos recibidos.

Al igual que con la vista Tops, que se detallará más adelante, se ha realizado una adaptación de estas para mostrar solo la información más relevante, dejando la interfaz limpia y fácil de leer.

4.5.1 Menú de configuración

En la vista RAW, este menú nos permite modificar la granularidad de los eventos que recibimos, agrupando en mayor o menor medida los eventos. Las granularidades disponibles en la vista RAW son:

- 1 minuto
- 5 minutos
- 15 minutos
- 30 minutos
- 1 hora
- 8 horas
- 1 día
- 5 días
- 1 semana

4.5.2 Lista de eventos

La lista de eventos muestra la información relevante de cada uno de ellos. Dado que cada módulo tiene distintas columnas para sus eventos, la información ofrecida está personaliza en función de las columnas disponibles. No obstante, todos siguen la estructura definida en la Figura 4.8.

Para los eventos del módulo IPS, es decir, eventos de seguridad, se ha destacado la gravedad, traza y sensor que lo ha detectado. Resaltando estos elementos en la lista conseguimos de un vistazo la información

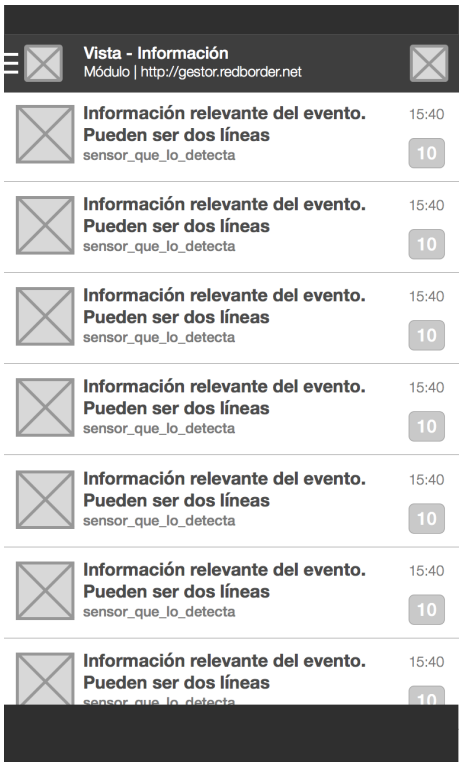

Figura 4.8 Estructura de la lista de eventos para la vista RAW.

más relevante. Como observamos en la implementación de esta vista en la Figura 4.9, la gravedad destaca visualmente, ya que consideramos que en primera instancia, es la información más importante asociada a un evento.

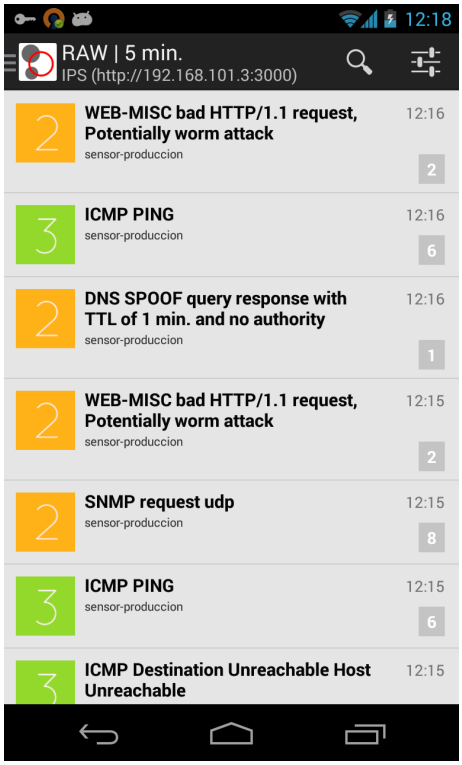

Figura 4.9 Implementación de la vista RAW para el módulo de IPS.

A continuación, tenemos la lista completa de información de cada evento del módulo IPS:

- *Severidad*: situada a la izquierda de cada fila de la lista, se representa con un número del uno al tres, dependiendo de la gravedad del evento.
- *Traza del evento*: descripción del evento de seguridad. Para resaltarlo se ha utilizado una tipografía en negrita.
- *Sensor que detecta el evento*: nombre del sensor.
- *Hora del evento*: fecha en la que se ha producido el evento. Si el día en el que ha ocurrido el evento es el actual, solo se muestra la hora y el minuto. En otro caso, se muestra también el día.
- *Número de eventos*: número de veces que se ha producido un evento en la granularidad seleccionada.

Para los eventos de tráfico del módulo de Flow, se ha destacado la información de la capa de red. En este caso, resaltamos el protocolo utilizado, así como la dirección origen y destino de paquete. Para especificar el sentido de la comunicación, se ha diseñado el icono situado a la izquierda de las direcciones IP. Podemos ver la implementación final de esta vista en la Figura 4.10

Figura 4.10 Implementación de la vista RAW para el módulo de Flow.

Las lista completa de las columnas de cada evento incluidas en esta vista para el módulo Flow es la siguiente:

- *Protocolo*: muestra el nombre del protocolo que se ha utilizado en la comunicación.
- *Dirección IP origen y destino*.
- *Hora del evento*: fecha en la que se ha producido el evento. Si el día en el que ha ocurrido el evento es el actual, solo se muestra la hora y el minuto. En otro caso, se muestra también el día.
- *Tráfico del evento*: número de bytes que ha generado el evento.

Para hacer transparente la paginación al usuario, esta lista implementa la técnica de *lista infinita*. Esta consiste en que cuando el usuario está llegando al final de la lista, se comienza a cargar el siguiente grupo de eventos. Claros ejemplos de uso de esta técnica son las aplicaciones de Twitter® e Instagram®.

También se ha querido agregar la funcionalidad de cargar nuevos eventos. Cuando entramos en esta vista, la lista comienza a pedir eventos a partir del momento actual hacia atrás. Pasado un tiempo, es posible que hayan llegado nuevos eventos, por lo que es muy importante poder recibirlos sin necesidad de volver a

cargar la vista entera. Para ello, se implementa otra téctica muy utilizada, *tirar y refrescar*. De este modo, la interacción para cargar nuevos elementos consiste en deslizar el dedo hacía abajo en el principio de la lista. Al levantar el dedo la lista comenzará a actualizarse. Véase la Figura 4.11.

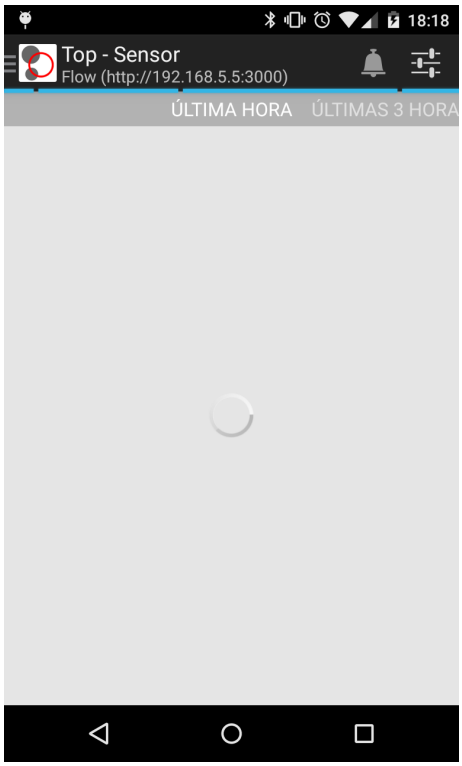

Figura 4.11 Vista cargandor tras realizar la acción de deslizar el dedo ‘‘arrastrando’’ la lista para obtener nuevos elementos.

4.5.3 Gráfica de evolución

Para la vista RAW, la gráfica muestra la evolución temporal de los eventos que han sido detectados. Estas gráficas ofrecen una visión de la tendencia del número de eventos en nuestro sistema. Gracias a este gráfico, es fácil visualizar comportamientos inusuales, ya que al poder cambiar la ventana temporal podemos comparar un rango de tiempo con situaciones pasadas.

Para el módulo de *Flow*, como muestra la Figura 4.12, el eje de ordenadas representa el total de tráfico en bytes en un instante determinado. En el caso del módulo *IPS*, , este eje representa el número de eventos que han sucedido en un instante.

4.5.4 Detalles de un evento

Como se ha comentado, en las vistas de la aplicación móvil se ha reducido la información ofrecida, pero cada evento tiene una gran cantidad de información asociada. Para satisfacer la necesidad de conocer todos los detalles, se ha diseñado una vista con todos los datos asociados.

Tras el prototipado y una primera versión, se incorporó un botón en la barra superior de navegación para compartir el detalle de un evento. Este botón genera la URL necesaria para identificar un evento en el gestor correspodiente. En la Figura 4.13 podemos ver el prototipo y la implementación final de esta vista.

4.6 Vistas de Tops

A diferencia de la vista RAW en el que establecemos una granularidad y vamos cargando nuevos eventos a medida que alcanzamos el final de la lista, en la vista Tops establecemos una ventada de tiempo. En el gestor existen unas ventanas de tiempo predefinidas, además de la opción de escoger cualquier período de tiempo. En la aplicación móvil hemos restringido las ventanas de tiempo a los siguientes valores:

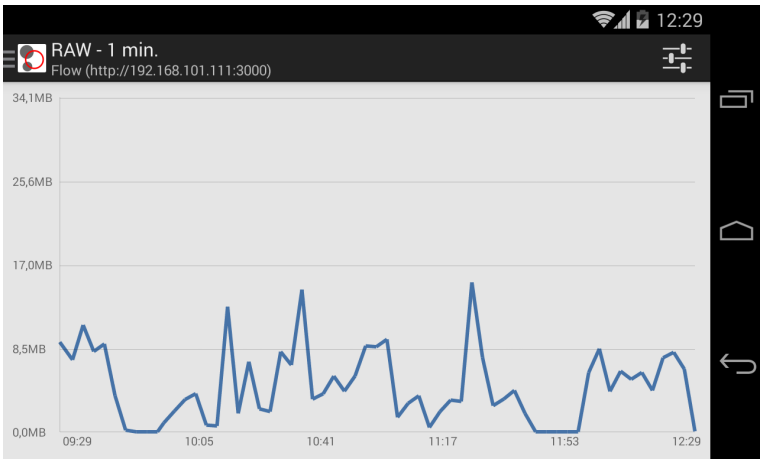

Figura 4.12 Gráfico de la vista RAW del módulo Flow.

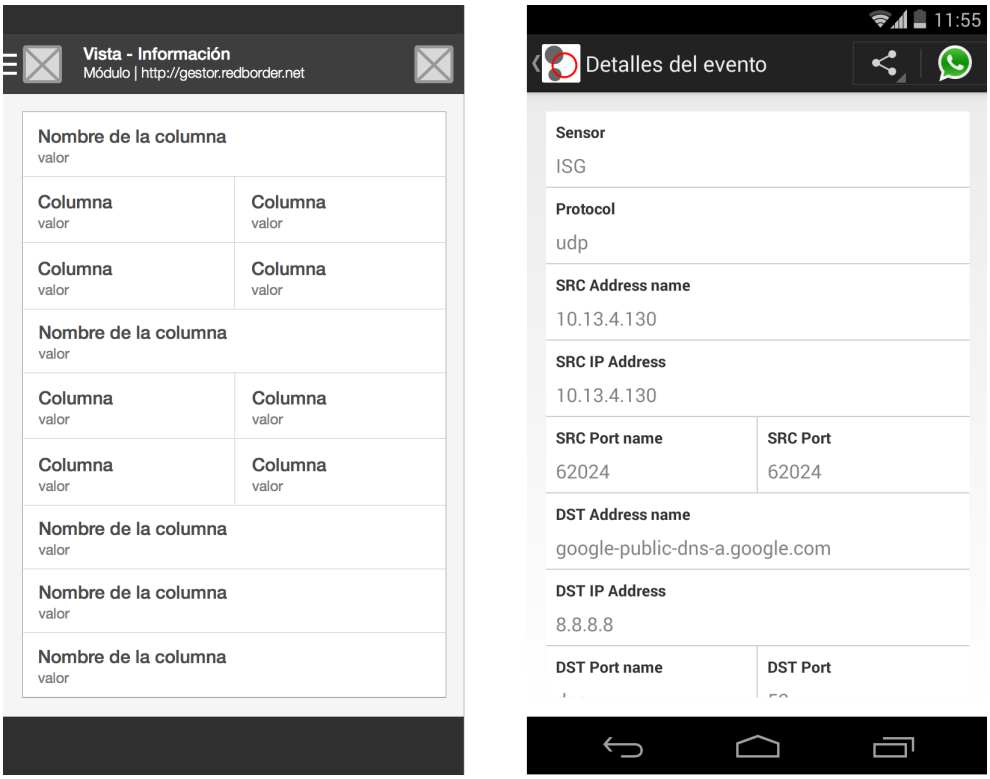

Figura 4.13 Prototipo y versión final de la vista de detalles de eventos RAW, incluyendo el botón de compartir.

- Última hora.
- Últimas tres horas.
- Últimas 24 horas.
- Última semana.
- Último mes.
- Último año.

4.6.1 Menú de configuración

En la vista Tops, este menú nos permite modificar la columna de la que queremos obtener la información. Las columnas varían en función del módulo, ya que vienen definidas por el tipo de evento que se recibe.

El el caso del módulo Flow, el número de columnas asociadas a cada evento es grande, por lo que se han agrupado estas bajo categorías para mejorar la navegación. Al pulsar en los títulos de las categorías estas se

expanden permitiendo seleccionar la columna que deseemos. La implementación final la podemos ver en la Figura 4.14.

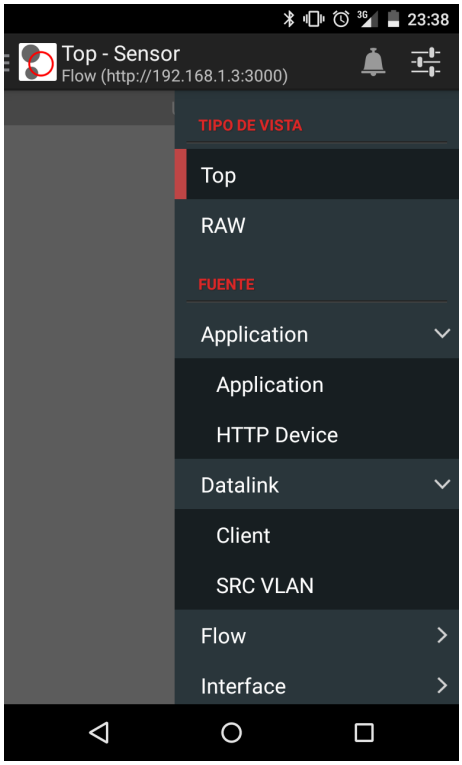

Figura 4.14 Menú de configuración de la vista Tops con algunas categorías expandidas.

4.6.2 Lista de eventos

Esta vista agrupa la información por columnas, por lo que en este caso, la estructura de los elementos de la lista de eventos son prácticamente idénticos. La única diferencia entre los módulos de IPS y Flow es el total. La información que representa cada elemento es:

- *Porcentaje*: porcentaje que representa dicho valor de una columna con respecto al total de eventos o tráfico generado por dicha column en un período de tiempo definido.
- *Valor de la columna*.
- *Total*: en el módulo IPS, el número de veces que se ha producido un evento con dicho valor para la columna seleccionada en la ventana de tiempo actual. En el caso del módulo Flow, este valor es el número de bytes generados con dichas condiciones.

En esta vista nos encontramos con la problemática de definir la ventana de tiempo que se quiere utilizar. Descartamos colocar esta opción en el menú de configuración, ya que para esta vista, el listado de columnas puede ser largo por lo que se prefirió no introducir seis elementos más al menú.

Finalmente diseñamos esta vista como un panel deslizante de Android®. Este tipo de interfaz nativa permite definir varias vistas accesibles de forma lateral, es decir, deslizando con el dedo a izquierda y derecha para movernos. Este tipo de vistas las utilizamos cada día, solo debemos de pensar en el escritorio de Android® en el que nos movemos de forma lateral.

Combinando estas ideas, en la Figura 4.15 podemos ver el prototipo final de la vista es el mostrado junto con su implementación final. En la implementación final, el porcentaje pasó de incluir el valor numérico, a ser indicado visualmente mediante las líneas azules posicionadas en la zona superior de cada evento.

4.6.3 Gráfica y filtrado

Al igual que en la vista RAW, al colocar el móvil en posición horizontal obtenemos una vista gráfica con la evolución temporal de los eventos de la ventana de tiempo que tengamos seleccionada.

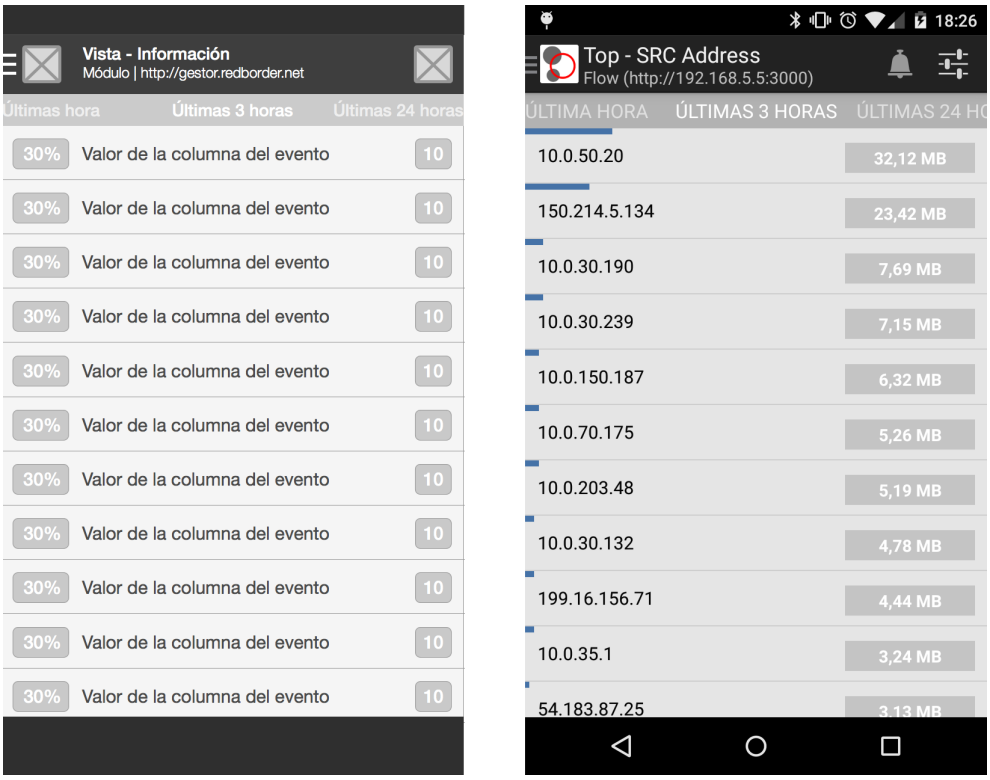

Figura 4.15 Prototipo de la vista Tops junto con la implementación final de esta.

Para este tipo de vista es interesante conocer la evolución de cada valor de las columnas de un evento por separado, pudiendo comparar varios valores de una columna en la ventana temporal elegida.

Con el fin de poder vistualizar estos valores por separado, se permite al usuario seleccionar varios elementos en la vista de lista (posición vertical). Una vez seleccionados, al girar el móvil obtenemos la gráfica de cada valor. Cada uno de ellos se identifica con un color, como se puede ver en la figura Figura 4.16.

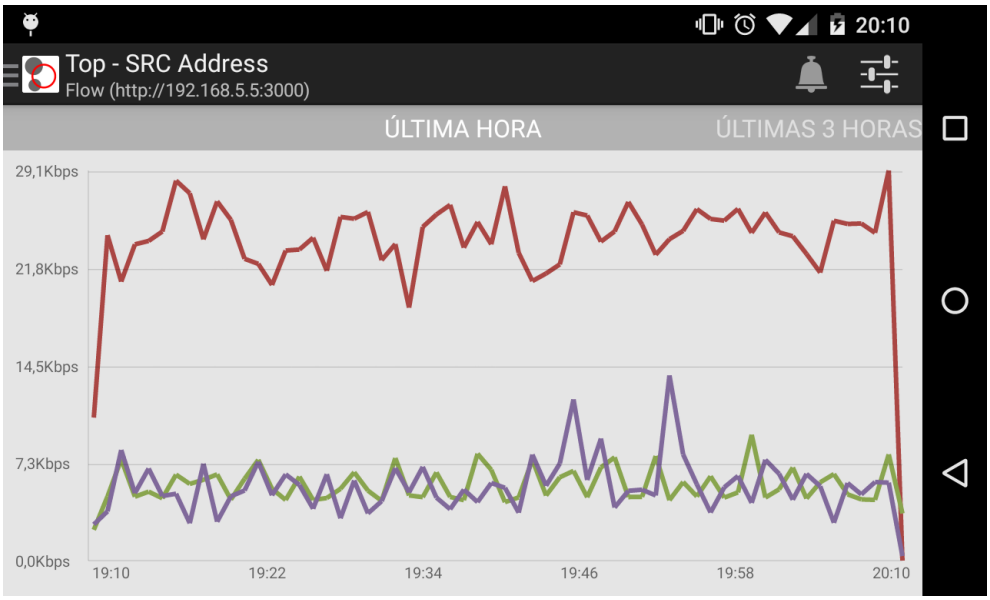

Figura 4.16 Gráfica de tops con varios valores para la columna de dirección IP origen en la última hora.

4.7 Alarmas

La sección de alarmas es muy importante en la aplicación móvil y por ello se ha incluido un icono para acceder a ella en la barra superior de navegación de las vistas Tops y RAW. Este icono despliega una lista con las alarmas actuales, destacando el fondo que las que aún no han sido leídas.

Las alarmas se definen en el gestor, donde se les atribuye un nombre y las condiciones para que sean lanzadas. Cada alarma incluye un límite superior o inferior, el cual se debe de sobrepasar durante un intervalo de tiempo, también definido en el gestor. Es posible aplicar filtros a los eventos para acotar más la alarma, ya que cuanto más concisa sea una alarma, menor será el número de falsos positivos.

En primera instancia se diseñó la vista de manera compacta, para poder visualizar varias alarmas sin tener que desplazarnos por la lista. Los datos mostrados sobre la alarma fueron: el nombre de esta y el valor numérico y la hora en la que fue lanzada.

Tras un tiempo de uso, decidimos que la información era confusa ya que no mostraba ninguna relación del valor numérico con respecto al límite definido. Por ello se incluyeron varios campos, quedando los siguientes campos de la alarma:

- *Nombre*: nombre de la alarma definido al crearla en el gestor.
- *Límite*: límite numérico definido en el gestor para esta alarma.
- *Valor*: valor númerico de la alarma en el momento en el que fue lanzada. También incluye el porcentaje que la alarma ha sobrepasado con respecto al límite.
- *Intervalo de lanzamiento*: se sustituye la hora por el intervalo, ya que en el gestor se establece el intervalo de tiempo durante el que el valor numérico debe sobrepasar el límite.
- *Gravedad de la alarma y tipo*: incluye el tipo de la alarma (si es de límite superior o inferior), sobre un fondo del color de la gravedad de la alarma. El gestor y la aplicación móvil comparte la gama de colores para indicar la severidad de una alarma.

Otro problema encontrado fue la gran cantidad de alarmas que se mostraban en la lista. Una vez que se ha solventado el problema por el que la alarma se ha lanzado, no tiene sentido seguir almacenando esta. Se agregó un botón al final de cada elemento de la lista para borrar la notificación.

Finalmente, en la figura Figura 4.17 podemos ver el prototipo inicial y como fue implementada.

4.8 Monitorización

La información que recibimos a través de sensores y distintos elementos de red es crucial, pero estos nodos también deben de ser monitorizados, ya que la desconexión o saturación de alguno de ellos puede provocar pérdidas en nuestro sistema.

4.8.1 Árbol de nodos

Muestra la estructura de sensores y dominios que el usuario autenticado puede visualizar. Al igual que el menú de configuración de la vista Tops, esta estructura es navegable permitiendo expandir el árbol al pulsar sobre los distintos elementos.

Al pulsar sobre un nodo con entidad física, como un sensor, la aplicación redirige al usuario a la vista de información sobre el nodo, donde entontramos distintas métricas sobre el estado de este.

4.8.2 Mapa de nodos

Desde el gestor se puede indicar la posición física de un nodo mediante sus coordenadas geográficas. Es por ello que la aplicación también incluye un mapa para visualizar los nodos de esta manera.

Al pulsar sobre un nodo en el mapa, se despliega una ventana con la información básica. Al pulsar sobre esta ventana nos muestra la vista de información sobre el nodo.

El problema de esta vista, es que los nodos pueden no disponer de su posición geográfica, por lo que no aparecerán en el mapa. Es por ello que esta vista es un complemento a la vista de árbol, ya que en esta si se muestran todos.

4.8.3 Información sobre el estado de un nodo

El gestor de redBorder, así como los sensores incluyen una serie de monitores físicos y virtuales. Existen múltiples tipos de monitores, siendo los más utilizados:

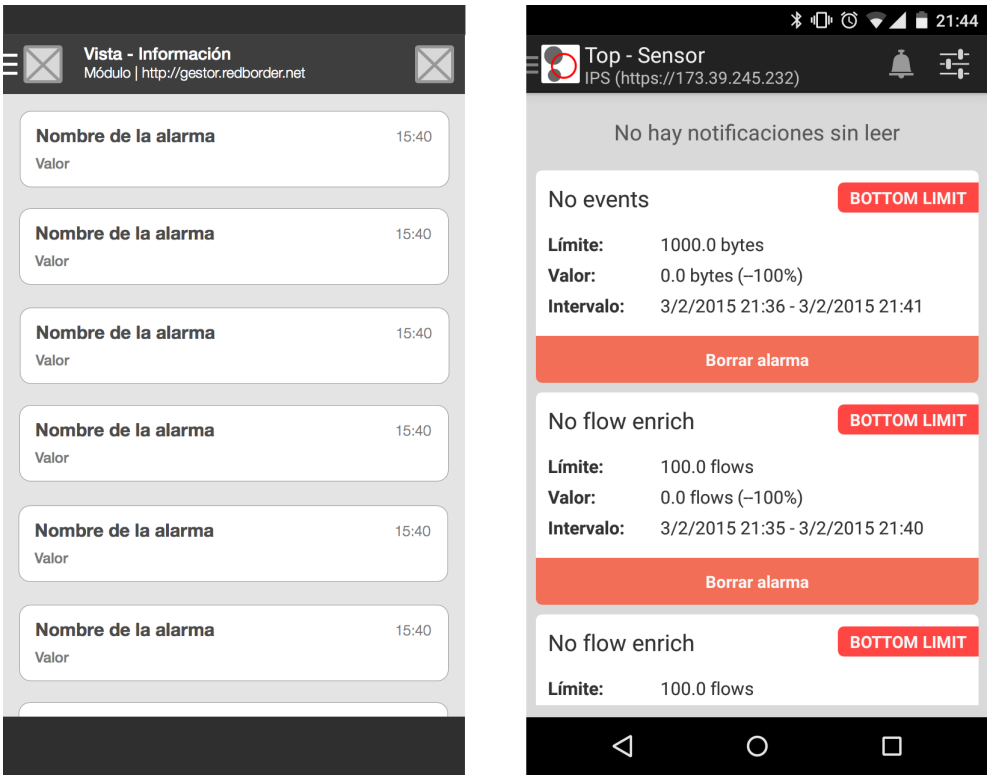

Figura 4.17 Prototipo y versión final de la vista de alarmas, incluyendo los cambios en los campos y el botón de borrar.

- *CPU*: porcentaje de CPU consumida.
- *Carga*: carga del sistema.
- *Memoria RAM*: memoria RAM usada, libre y en caché.
- *Temperatura*: temperatura del dispositivo.
- *Información de paquetes*: paquetes recibidos, pérdidos y enviados.

Para la aplicación móvil, se han incluido los monitores de:

- CPU
- Carga
- Memoria RAM usada
- IP del nodo
- Temperatura del dispositivo
- Velocidad del ventilador
- Estado completo de la memoria, incluyendo información sobre la memoria libre, usada y en caché.

Para alertar de una manera más directa sobre una sobrecarga en el nodo, se han mostrado los monitores de CPU, carga y memoria RAM como gráficas circulares con una gama de colores de verde a rojo en función de la saturación de dicho monitor. La vista final es la mostrada en la Figura 4.18.

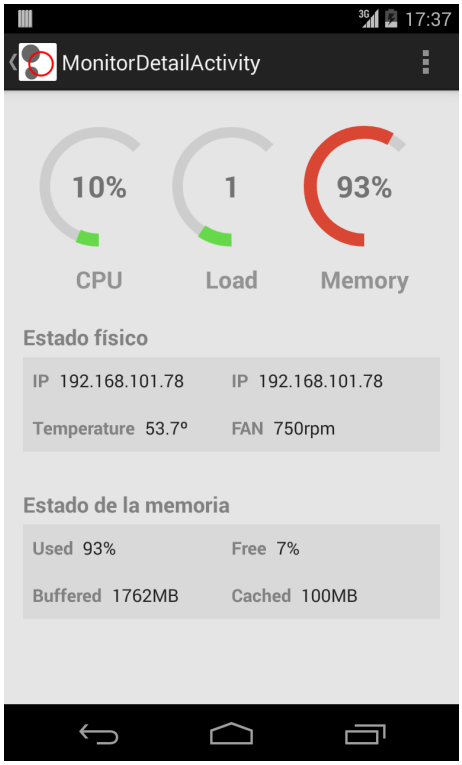

Figura 4.18 Vista del estado de un nodo, con toda la información relativa a su estado físico.

5 Implementación

Las aplicaciones desarrolladas para el sistema operativo Android® utilizan el lenguaje de programación Java® de una manera particular. Android® provee al programador de todo un conjunto de herramientas y librerías para trabajar fácil e intuitivamente con su sistema operativo.

5.1 Entorno de desarrollo

La aplicación ha sido desarrollada en un ordenador con sistema operativo Mac OS X, en el cual se instalaron las herramientas de desarrollo de Android, conocidas como Software Development Kit (SDK). Estas herramientas nos permiten hacer uso de todas las funcionalidades de nuestro dispositivo, como puede ser el acceso a localización vía GPS o navegar por los ficheros del dispositivo.

Este set de herramientas también incluye un emulador para simular un dispositivo en el que probar las aplicaciones en nuestro ordeador.

5.1.1 Android

Definición 5.1.1 (Fragmento) llamados *fragment* en inglés, es una porción de interfaz que realiza una función concreta. Estos son reutilizables y permiten modularizar la interfaz, de manera que una vista puede estar compuesta de varios fragmentos.

El sistema operativo Android® fue lanzado el día 23 de septiembre de 2008. Desde este momento se han sucedido una gran cantidad de versiones, mejorando la estabilidad y rendimiento y agregando nuevas funcionalidades.

Para esta aplicación, se ha limitado la versión de Android® a dispositivos con una versión igual o superior a JellyBean® (Versión 4.1). Tomamos la decisión aplicar la limitación en esta versión por la inclusión de manera nativa de fragmentos, muy utilizados en el desarrollo de la aplicación y la necesidad de determinadas librerías disponibles a partir de esta versión.

Toda la documentación utilizada para la implementación de este proyecto se encuentra en [2].

5.1.2 Android Studio

Como Entorno de desarrollo integrado (EDI) se ha escogido la aplicación desarrollada por Google®, Android Studio. Este entorno centrado en la programación para Android® está integrado con el conjunto de herramientas anteriormente citado, lo que permite desarrollar mucho más rápidamente, ya que no hace falta cambiar de aplicación para comprimir y firmar la aplicación o instalarla en un dispositivo móvil.

Otra característica destacada es el autocompletado de código que incluye todas las librerías porporcionadas por Android®, así como la correción de errores tipográficos lo que facilita en gran medida el desarrollo. Por ejemplo, partiendo de una clase podemos ver todos métodos públicos de esta.

También incluye un editor visual para el diseño e implementación de vistas, como podemos ver en la Figura 5.1. Con este podemos diseñar una vista sin introducir una sola línea de código, solamente arrastrando elementos y modificando sus atributos mientras comprobamos el resultado al instante. En algunos casos estos entornos no son suficientes para el diseño de una vista, como cuando incluimos elementos gráficos personalizados. Para estos casos, también podemos editar directamente el código XML de las vistas.

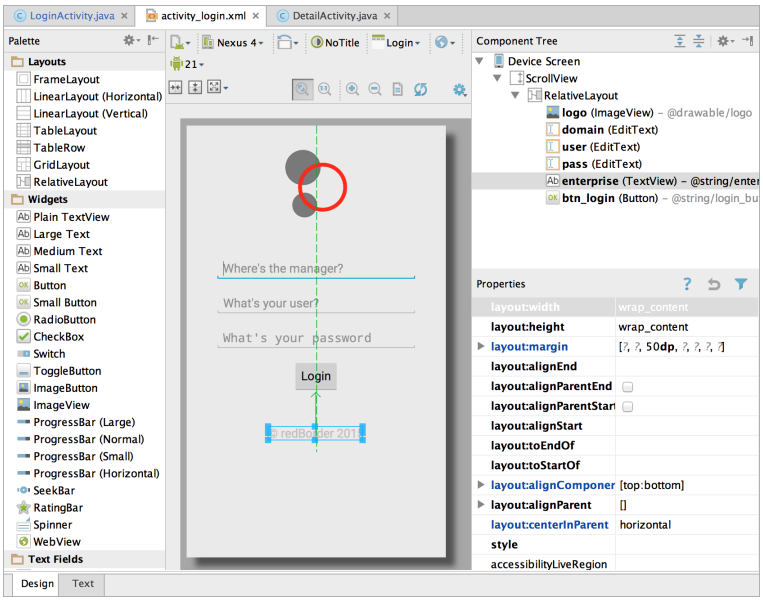

Figura 5.1 Modificando la vista de inicio de sesión con el editor visual de Android Studio.

Android Studio también incorpora un depurador para Android®. Este muestra la información de nuestro dispositivo, y para nuestra aplicación en desarrollo algunos parámetros tan interesantes como la memoria consumida o la carga que esta provocando en la CPU. Esta funcionalidad permite acotar problemas de rendimiento y memoria, un tema muy complejo e importante en el desarrollo de aplicaciones móviles.

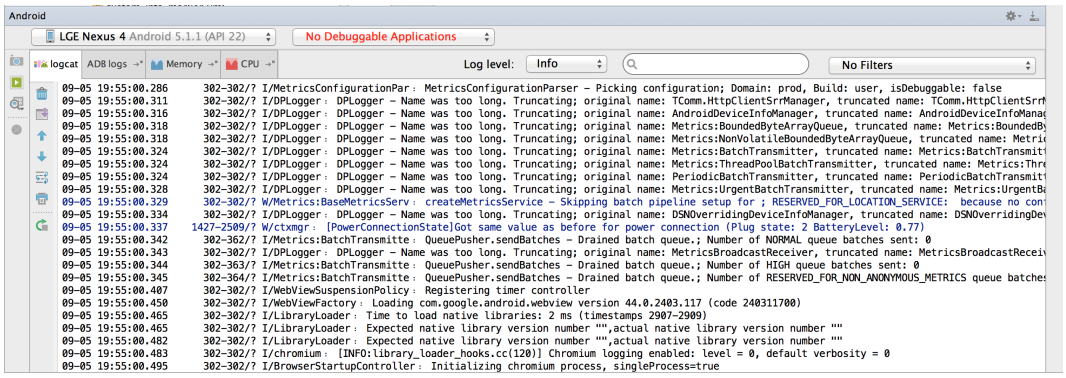

Figura 5.2 Depurador de Android Studio.

5.1.3 Gradle

Gradle® es un sistema de automatización de tareas centrado en la composición de aplicaciones, lo que engloba tareas como:

- Testeo de aplicaciones.
- Gestión de dependencias como librerías.
- Compilación.
- Empaquetado de aplicaciones.
- Publicación, en el caso de Android®, en *Play Store* [1].

Inspirado en los proyectos de Apache Maven y Apache Ant, Gradle agiliza todas las tareas monótonas de compilación y empaquetado. La gestión de dependencias es otra gran característica, ya que la utilización de librerías se convierte la inclusión de una línea de texto en su fichero de configuración. Esto es posible gracias a que Gradle utiliza Apache Maven para la descarga y gestión de librerías.

[1] *Play Store* es el nombre de la tienda de aplicaciones de Android®

Cada aplicación Android está cifrada y sellada para garantizar al creador de esta. Gradle provee herramientas para establecer entornos y utilizar las claves de cifrado necesarias. Generalmente se utilizan dos entornos, uno de producción, con la clave para publicar la aplicación en *Play Store*; y uno de desarrollo, donde se firma la aplicación con otra clave distinta.

El fichero de configuración de Gradle es `build.gradle`. En este se incluye toda la configuración de dependencias, firma y empaquetado de un proyecto. Cada librería utilizada incluye su propio fichero de configuración. Se adjunta el fichero de configuración de la aplicación en el Apéndice A.

5.1.4 Genymotion

Uno de los problemas del emulador de Android® es su lentitud. La tardanza en iniciar este, el consumo de recursos, así como la ralentización cuando se trabaja con el depurador hacen complicada la tarea de probar las aplicaciones en el ordenador.

Genymotion® es una aplicación que virtualiza imágenes de dispositivos Android® sobre la aplicación de virtualización VirtualBox®. Al trabjar sobre virtualización y no emulación, la velocidad de carga es considerablemente más alta, pasando de uno o dos minutos para iniciar el emulador de Android® a segundos para iniciar una imagen de este Genymotion®.

Para la depuración de la aplicación se utilizó la versión gratuita de Genymotion® y las imágenes de Android® de la versión 4.3 y 5.0.

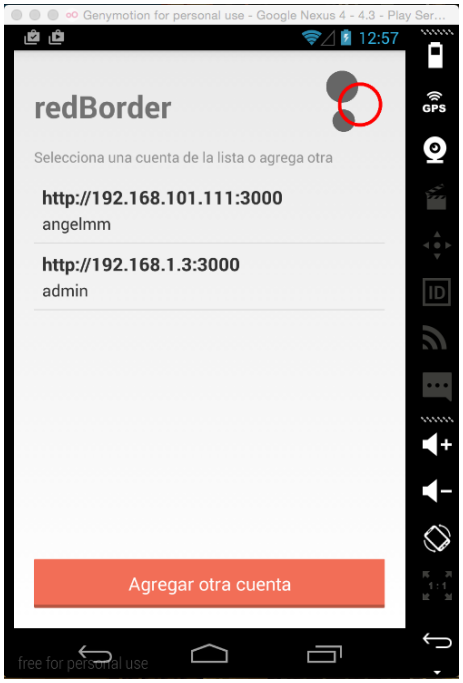

Figura 5.3 Virtualización de Android versión 4.3 por Genymotion®.

5.2 Librerías utilizadas

Desarrollar una aplicación en Android® es complejo, pero gracias a su filosofía de código abierto, existen una gran cantidad de librerías disponibles que facilitan tareas como el trabajo con la base de datos, o agregan funcionalidades, como la inclusión de gráficas.

Siguiendo con esta filosofía, durante el desarrollo de este proyecto se han utilizado varias librerías, colaborando conjuntamente con los autores de estas para la agregación de mejoras y discusión de problemas. También se ha creado una librería disponible públicamente.

5.2.1 ActiveAndroid

ActiveAndroid es una librería de conversión de objetos de Java a entidades de una base de datos relacional. Este tipo de librerías son más conocidas por sus siglas en inglés: Object-Relational mapping (ORM). Es de código abierto y su documentación está disponible en [7].

Esta librería abstrae al programador de la necesidad de definir la conexión, cadenas de comandos y creación de la base de datos, es decir, toda integración con esta queda relegada a la librería que incluye métodos para crear, borrar y editar tuplas.

Se utiliza para la gestión de las alarmas almacenadas en el dispositivo móvil.

Rendimiento en la inicialización de la aplicación

Se detectó un problema a la hora de iniciar la aplicación y es que esta librería comprueba todos los ficheros del proyecto. El problema es que esta comprobación incluía también a las librerías externas, como la de compatiblidad con versiones anteriores de Android®, lo que ralentizaba en gran medida la carga.

Para solventar este problema, se duplicó esta librería localmente y se agregó una opción para deshabilitar la comprobación de librerías externas. Este cambio se propuso al creador de la librería, y a día de hoy se está conversando sobre su inclusión. Para más información sobre el estado de esta conversación: *https://github.com/pardom/ActiveAndroid/pull/247*.

Dada la necesidad de reducir el tiempo de carga, la librería que se está utlizando actualmente en el proyecto es la versión modificada, disponible en *https://github.com/Angelmmiguel/ActiveAndroid*.

5.2.2 Restrung

Las peticiones y respuestas del gestor se realizan en formato JSON. Para trabajar con los resultados, es necesario convertir la cadena de texto de la respuesta a variables de Java®. Una manera de realizar esta conversion es mediante la clase `JSONObject`.

No obstante, trabajar con este tipo de objetos es tedioso y muy poco intuitivo. `Restrung` es una librería para gestionar peticiones a un servidor y convertir la respuesta en formato JSON a objetos de Java.

Para ello, creamos una clase que extiendiendo de `AbstractJSONResponse`, define la estructura del objeto JSON de la respuesta. Cuando realicemos peticiones al servidor, `Restrung` convertirá la respuesta a instancia de dicha clase.

Actualmente la librería ha cambiado su nombre a *Appsly Android REST*, pero al cambiar la estructura de las peticiones, se mantuvo la versión anterior. Esta librería es de código abierto y su documentación está disponible en [6].

5.2.3 Google Maps Android

Libería para la integración de mapas en la aplicación. Esta funcionalidad se utiliza en la vista de monitorización de nodos para la visualización de estos en el mapa, definido en la Subsección 4.8.2.

Esta librería requiere de un identificador del servicio, ya que require de la obtención de datos de la aplicación `Maps` de Google®. Podemos ver la inclusión de este identificador en el archivo *AndroidManifest.xml* incluído en el Apéndice B.

5.2.4 AnotherExpandableListView

Durante la implementación de la vista de lista de monitorización de nodos se agregaron varias modificaciones a las listas expandibles de Android®. Para asimilar el diseño a la lista de árbol del gestor, se agregaron flechas a la izquierda de los elementos que incluyen una animación al abrir y cerrar un elemento.

El resultado visual fue satisfactorio y muy sencillo de utilizar. Es por ello que se extrajo la funcionalidad agregada en una librería para que cualquier desarrollador de Android® pudiera utilizarla.

El código de esta librería está disponible en *https://github.com/Angelmmiguel/AnotherExpandableListView*.

5.2.5 Android Pull-to-refresh

Esta librería provee a la aplicación de un nuevo elemento gráfico, una lista que se puede arrastrar hacía abajo para lanzar un evento de actualización. Esta se utiliza en la mayoría de vistas de lista para actualizar los eventos.

Esta librería es de código abierto, y está disponible en *https://github.com/naver/android-pull-to-refresh*. Su utilización es muy sencilla y permite modificar varios detalles gráficos como el formato visual de la barra de progreso de carga.

Pulsación larga

El funcionamiento es similar al de una lista de Android® ya que incluye deslizamiento por la lista, pulsación de elementos... No obstante, al implementar la vista de Tops, no estaba disponible la opción de pulsación larga en un elemento de la lista. Este evento era necesario para el filtrado de eventos en la vista gráfica.

Se realizaron varios cambios sobre la librería y se propusieron al autor de esta. Este los aceptó y ahora esta funcionalidad está disponible para quienes la utilicen. Para más información sobre estos cambios, *https://github.com/naver/android-pull-to-refresh/pull/13*

Parada de la actualización de la lista

Otro problema con esta lista se presentó a la hora de trabajar con la vista gráfica. Cuando se realizaba una actualización mediante arrastre de la lista y se giraba el dispositivo antes de que esta terminara de actualizarse, la aplicación se cerraba debido a un error ya que la lista ya no estaba disponible.

Tras estudiar el caso, se agregó un método para parar la actualización de la lista cuando fuera necesario. Este cambio también fue aceptado por el autor de la librería como se puede ver en *https://github.com/naver/android-pull-to-refresh/pull/18*.

5.2.6 HoloGraph library

HoloGraph es una librería de gráficos para Android. No es la librería de este tipo que más funcionalidad tiene, pero si la más llamativa visualmente. Por esa característica fue la elegida entre varias, además de por ser de código libre. Su documentación está disponible en [5].

Esta librería provee de elementos visuales para generar gráficos de línea, barras y circulares en nuestra aplicación Android®.

Su uso se ha limitado a los gráficos de línea para las vistas de RAW y Tops, y gráficos circulares para la vista de detalles del estado de un nodo.

Gestión de eventos temporales

La manera de trabajar con gráficos de línea en el gestor de redBorder® es mediante series temporales. En estas, el eje de abcisas representa el tiempo y el de ordenadas el valor de una determinada columna de un evento.

Trabajar de esta manera con la librería se hacía muy complejo ya que esta no entiende de series temporales. Para solventar esta dificultad, se duplicó el proyecto de *HoloGraph* y se introdujeron mejoras para facilitar el trabajo con series temporales.

Estos cambios no se propusieron al creador de la librería ya que los requisitos de funcionalidad eran muy específicos para la aplicación de redBorder®. Es por ello que la librería original siguió avanzando por separado y que actualmente la librería utilizada es la duplicada, disponible en: *https://bitbucket.org/angelmm/holographlibrary*

5.3 Proyecto de redBorder Mobile

Definición 5.3.1 (Groovy) lenguaje de programación en el que se basa el sistema de automatización Gradle.

La aplicación está escrita íntegramente en Groovy, Java y XML, siguiendo la estructura básica de una aplicación Android que incluye:

- *Carpeta* `res`: incluye todos los recursos de la aplicación como imágenes, ficheros de vistas, de idiomas, de menús y de dimensiones.
- *Carpeta* `src`: contiene el código fuente de la aplicación.
- `AndroidManifest.xml`: fichero básico de definición de la aplicación. Necesario para Android.
- `build.gradle`: archivo de compilación de Gradle.

5.3.1 AndroidManifest.xml

Definición 5.3.2 (Actividad) en inglés *Activity*, es una clase que gestiona la interacción de un usuario en una parte de la aplicación. Una actividad incluye en una ventana la vista que se debe de cargar, así como uno o varios fragmentos. Cada actividad tiene su propio ciclo de vida. Más información en [1].

Definición 5.3.3 (Servicio) componente de la aplicación sin interfaz gráfica que se ejecuta en segundo plano aunque la aplicación principal se haya cerrado u ocultado.

Definición 5.3.4 (Receptor) actividad de la aplicación que puede recibir eventos externos a esta. Estos eventos pueden ser de otras aplicaciones o del propio sistema operativo.

Dentro de este fichero necesario para Android®, se deben de registrar:

- Todas las `actividades` que conforman la aplicación y pueden ser llamadas e inicidas.
- Los permisos, es decir, funcionalidades del dispositivo a las que puede acceder desde la aplicación.
- Nombre de la aplicación y una versión numérica, así como un alias de esta en formato de cadena de texto.
- `Servicios` que la aplicación utiliza, como la autenticación y el servicio de notificaciones.
- `Receptores` disponibles en la aplicación.

El contenido de este fichero se encuentra en el Apéndice B.

5.3.2 Permisos

Definición 5.3.5 (Notificación PUSH) mensajes entre un servidor y una aplicación en el que existe un nodo intermedio que almacena dichos mensajes durante un tiempo hasta que el receptor esté disponible.

La lista de permisos disponibles se encuentra en la documentación oficial de Android, concretamente en [4]. De todos ellos, los permisos solicitados por la aplicación son los siguientes:

- *INTERNET* y *ACCESS_NETWORK_STATE*: necesario para conectar con el gestor de redBorder.
- *AUTHENTICATE_ACCOUNTS*, *GET_ACCOUNTS*, *MANAGE_ACCOUNTS* y *USE_CREDENTIALS*: necesario para el almacenamiento y lectura de las credenciales del usuario para conectar a los distintos gestores.
- *ACCESS_COARSE_LOCATION*, *ACCESS_FINE_LOCATION*, *READ_EXTERNAL_STORAGE*, *WRITE_EXTERNAL_STORAGE* y *com.google.android.providers.gsf.permission.READ_GSERVICE*: necesario para el uso de mapas en la vista de elementos monitorizados.
- *com.google.android.c2dm.permission.RECEIVE*: permite a la aplicación recibir notificaciones PUSH.

5.4 Código fuente

Para falicitar el desarrollo, se ha organizado el proyecto en múltiples paquetes en función del propósito de las clases que contiene.

5.4.1 Actividades

Definición 5.4.1 (Intent) en Android®, se entienden los *intents* como una abstracción para solicitar al sistema operativo la realización de una operación. Esta puede ser desde un cambio de actividad, al envío de datos entre aplicaciones.

Las actividades de la aplicación se encuentran en el directorio raíz del código fuente. Cada vista con interfaz gráfica tiene como base una `actividad` que define las funciones básicas de esta e instancia la propia interfaz o los distintos `fragmentos` que la componen.

Para cambiar de actividad en Android® se utilizan los llamados `intents`. Estos nos permiten cambiar de actividad y enviar datos entre estas, como parámetros que el usuario haya introducido o cualquier variable de la actividad anterior.

En esta aplicación podemos dividir las actividades en dos tipos:

Actividades simples

Este tipo de actividades no tienen fragmentos asociados, es la propia actividad la que procesa la interfaz y gestiona todo los eventos de interacción del usuario. Este caso se da en las vistas de inicio de sesión y detalles.

Actividades compuestas

Esta estructura más compleja es la utilizada para las vistas de Tops y RAW. En estas vistas la `actividad` instancia dos `fragmentos`. Uno de ellos procesa la interfaz que corresponde en función de la posición del dispositivo (vista de lista para vertical y gráfico para horizontal). El otro fragmento no tiene interfaz y se encarga de obtener los datos del servidor y almacenarlos temporalmente.

5.4.2 Modelos

Los modelos representan la información con la que la aplicación trabaja, como eventos o usuarios. Este concepto se ha extraído del patrón de arquitectura de software Modelo-Vista-Controlador. Todos los modelos se encuentran en el paquete `net.redborder.redborder.models`.

La gran ventaja de la utilización de modelos es la integración con las librerías Restrung y ActiveAndroid, ya que las peticiones y las tuplas de la base de datos quedan instanciadas como objetos de estas clases. Generalmente contienen las variables y las funciones de obtención e inicialización de variables.

En el caso de los modelos que se inicializan con respuestas del gestor, las clases heredan de `AbstractJSONResponse`, definida en la librería de Restrung. También heredan de esta las clases que representan los eventos recibidos desde el gestor como `RAWEvent` y `RAWEventFlow`. A modo de ejemplo, se incluye la clase de `User` en el Apéndice C.

Las clase de alarmas, incluida en el Apéndice D y que utiliza ActiveAndroid para su gestión, hereda de la clase `Model` que establece distintas anotaciones para generar la tabla y las columnas en la base de datos.

Dentro de este paquete también se incluyen los modelos utilizados para el procesamiento de menús, concretamemte las clases `MenuItemCustom` y `MenuItemExpandableCustom`. Estas definen todas las propiedades necesarias, así como los constructores para inicializar cada elemento. Como ejemplo se incluye la clase `MenuItemCustom` en el Apéndice E.

5.4.3 Colecciones

Las colecciones representan conjuntos de modelos. Son utilizadas para la recepción de los eventos desde el gestor, por lo que como ocurre con algunos modelos, heredan de `AbstractJSONResponse`. Además de incluir una variable de tipo `List` donde se encuentra todos los eventos, también incluye otras necesarias en las peticiones al gestor.

Como ejemplo, las variables definidas para los eventos de IPS de la vista RAW son las mostradas en el Código 5.1. Para el caso de los eventos de Flow y la vista Tops solo cambia el tipo de objeto con el que se instacia la variable de tipo `List`.

Código 5.1 Variables definidas para los eventos de IPS de la vista RAW..

```
public class RAWEventCollection extends AbstractJSONResponse {

  // List of events
  private List<RAWEvent> events;
  // Status of the query (true or false)
  private String status;
  // Offset of elements to paginate
  private String offset;
  // End time of the query
  private String end_time;

  // ...

}
```

5.4.4 Adaptadores de listas

Cada elemento gráfico de tipo lista en Android® requiere de un *adaptador* que se encarga de generar las vistas de cada uno de los elementos de esta. Todos los adaptadores se encuentran en el paquete `package net.redborder.redborder.adapters`.

Cada adaptador hereda de la clase `BaseAdapter` de Android® que define una serie de métodos necesarios para la gestión de la lista:

- `getCount`: devuelve el número de elementos.
- `getItem`: devuelve el objeto asociado a un elemento.
- `getView`: procesa la vista de un elemento y la devuelve.

Cabe destacar que el método `getView` sigue el patrón `ViewHolder` recomendado por la comunidad de desarrollo de Android (más información en [3]).

Este patrón almacena las referencias de los componentes gráficos de un elemento de la lista. Esto permite el reciclado de las vistas de los elementos, ya que no necesitan buscar de nuevo las referencias, lo que aumenta el rendimiento de carga de las lista, evitando ralentizaciones al navegar por esta.

El código 5.2 ilustra el uso de este patrón en el adaptador de la lista de eventos de la vista RAW.

Código 5.2 Uso del patrón *ViewHolder* en la lista de eventos de la vista RAW..

```
/*
   * Private class view holder to initiate and reuse the views in the listView
   */
  private static class ViewHolder {
    public ImageView severity;
    public TextView signature;
    public TextView sensor;
    public TextView hour;
    public TextView events;
    public RelativeLayout wrap;
  }

  @Override
  public View getView(int position, View view, ViewGroup viewGroup) {

    View vi = view;
    ViewHolder holder = null;
    RAWEvent event = (RAWEvent) getItem(position);

    // If we can recycle the view, we don't create a new view
    if(vi == null){
        //The view is not a recycled one: we have to inflate
        vi = li.inflate(R.layout.event_item, viewGroup, false);
        holder = new ViewHolder();

        holder.wrap = (RelativeLayout)vi.findViewById(R.id.event_wrap);
        holder.severity = (ImageView)vi.findViewById(R.id.severity);
        holder.signature = (TextView)vi.findViewById(net.redborder.redborder.R.
            id.signature);
        holder.sensor = (TextView)vi.findViewById(net.redborder.redborder.R.id.
            sensor);
        holder.hour = (TextView) vi.findViewById(net.redborder.redborder.R.id.
            timestamp);
        holder.events = (TextView)vi.findViewById(net.redborder.redborder.R.id.
            events);
        vi.setTag(holder);
    } else {
        // View recycled !
        // no need to inflate
        // no need to findViews by id
        holder = (ViewHolder) vi.getTag();
```

```
    }

    // ...
    // Código no relevante para este ejemplo
    // ...

    // Return the view
    return vi;
  }
```

5.4.5 Colectores de datos

Como se ha comentado, tanto en RAW como en Tops tenemos dos tipos de vistas en función de la posición del dispositivo. El intercambio entre la vista de lista y gráfico debe de ser rápido para no perder la atención del usuario.

Concretamente para Tops, cuando filtramos en la lista esta asocia un color a cada elemento, por lo que al girar el dispositivo y ver el gráfico debemos de recordar la asociación de colores. Otra razón más para agilizar el intercambio entre estas.

El mayor tiempo de espera al cambiar la orientación es el de respuesta de las peticiones al servidor. Para solventar este problema de tiempos de carga, se ha definido el paquete `package net.redborder.redborder.dataCollector`.

Este paquete incluye dos clases[2] que heredan de la clase `Fragment`. Estos fragmentos se inicializan junto con una de las vistas y no son eliminados al cambiar entre estas. Se encargan de realizar las llamadas al servidor y almacenar las respuestas para evitar volver a solicitar los datos al cambiar la orientación del dispositivo.

Se incluye la clase `DataCollectorRAW` en el Apéndice F.

5.4.6 Gestor de gráficos

`HoloGraph` es la librería elegida para la generación de los gráficos de la aplicación. Sin embargo, dado su uso en varias secciones de la aplicación, se agrupó la gestión y representación de gráficos dentro del paquete `net.redborder.redborder.graph`.

El paquete define la morfología de la respuesta del servidor. Para ello, al igual que los modelos, utiliza la librería `Restrung` para inicializar los objetos a partir de la respuesta del gestor.

Cada serie de datos está representada por la clase `GraphCollection`. Para los gráficos de la vista de RAW solo tendremos una serie, pero en el caso de la vista Tops la clase `GraphMultipleCollection` contendrá la lista de series en una variable de tipo *List*.

A su vez, cada serie incluye multiples eventos representados por la clase `GraphSerie`. Cada evento incluye el valor de este y el tiempo en el que ha ocurrido.

Esta estructura tan compleja es necesaria para la librería `Restrung`. La Figura 5.4 muestra la estructura de la respuesta del servidor y su representación en la aplicación móvil.

Este paquete incluye la clase gestora `GraphManager`, que se encarga de inicializar los elementos de la interfaz y a partir de los datos y mediante la citada librería, procesar los gráficos. Los métodos más importantes son dos:

- `getRawGraph`: partiendo de un objeto de tipo `GraphCollection` instanciado por la respuesta del servidor, limpia la gráfica actual y procesa los nuevos puntos.
- `getTopGraph`: a partir de un objeto de tipo `MultipleGraphCollection`, itera sobre cada serie representándola con el color del filtro que se ha aplicado en la lista.

Estas dos funciones también se encargan de inicializar los valores mostrados en los ejes de coordenadas, que como se ha comentado, es una funcionalidad agregada a la librería para este proyecto.

`GraphManager` contiene una clase privada que gestiona los eventos táctiles de arrastre. Estos eventos permiten al usuario cambiar la ventana de tiempo para los gráficos de la vista RAW.

El contenido de `GraphManager` se incluye en el Apéndice G

[2] Una para las vistas de RAW y otra para las de Tops

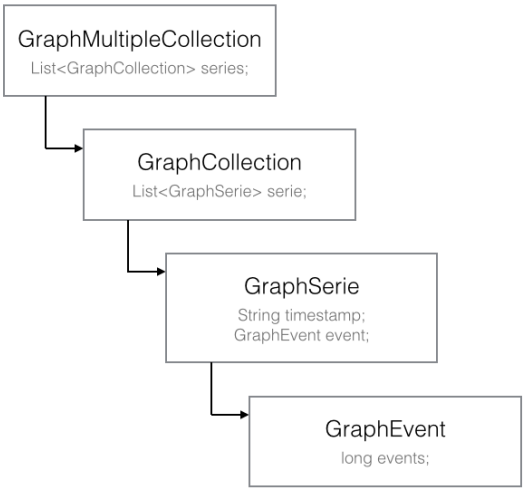

Figura 5.4 Estructura de clases de la respuesta del servidor para el procesamiento de gráficos.

5.4.7 Notificaciones PUSH

Para recibir las notificaciones de tipo PUSH en nuestro dispositivo necesitamos hacer uso del servicio de Google Cloud Messaging (GCM) [3] de Google®. Este servicio permite la comunicación mediante mensajes de este tipo entre en un servidor y los clientes. Este se asociada a una cuenta de desarrollador de Google®, pudiendo suponer un coste si se supera un número de mensajes al mes.

Al registrar el servicio con Google®, se nos proporciona un identificador de servicio que utilizará el servidor. A su vez, se asocia la aplicación móvil con la cuenta para permitir el envío de mensajes a esta.

Para poder enviar mensajes a un dispositivo concreto, este debe de quedar registrado mediante otro identificador, que será el que utilizará el servidor para comunicarse con el. Este registro se realiza en la vista de inicio de sesión siguiendo el diagrama mostrado en la Figura 5.5. Todas las clases relacionadas con este servicio se encuentran en el paquete `package net.redborder.redborder.gcm`.

Como muestra el diagrama, `GCMRegister` es la clase que se encarga de gestionar el registro del dispositivo en el servicio GCM y su código está incluido en el Apéndice H. También se encarga de almacenar el identificador, así como de leerlo y comprobar cuando es necesario solicitarlo. Las condiciones para solicitar un nuevo identificador son:

- No haber obtenido el identificador antes.
- La versión de la aplicación ha cambiado con respecto a cuando se solicitó el identificador.

Dentro de este paquete también tenemos la clase `GcmBroadcastReceiver`. Esta se encarga de recibir los mensajes que el servidor ha enviado. Para ello, inicia el servicio `GcmIntentService` que procesa el mensaje. En nuestro caso, estos mensajes contienen información sobre las alarmas, por lo que `GcmIntentService` almacena estas y muestra una notificación en el dispositivo cuando es necesario.

5.5 Implementación de las vistas

5.5.1 Inicio de sesión

Cuando un usuario inicia la aplicación de redBorder, esta sigue el diagrama de la Figura 5.6 para representar una vista u otra en función de la existencia de cuentas de redBorder en el dispositivo.

Si no existen cuentas, como muestra el Código 5.3, se lanza un `intent` para cambiar de actividad a la definida en la clase `AuthenticatorActivity`.

[3] Más información sobre el servicio en *https://developers.google.com/cloud-messaging/gcm*

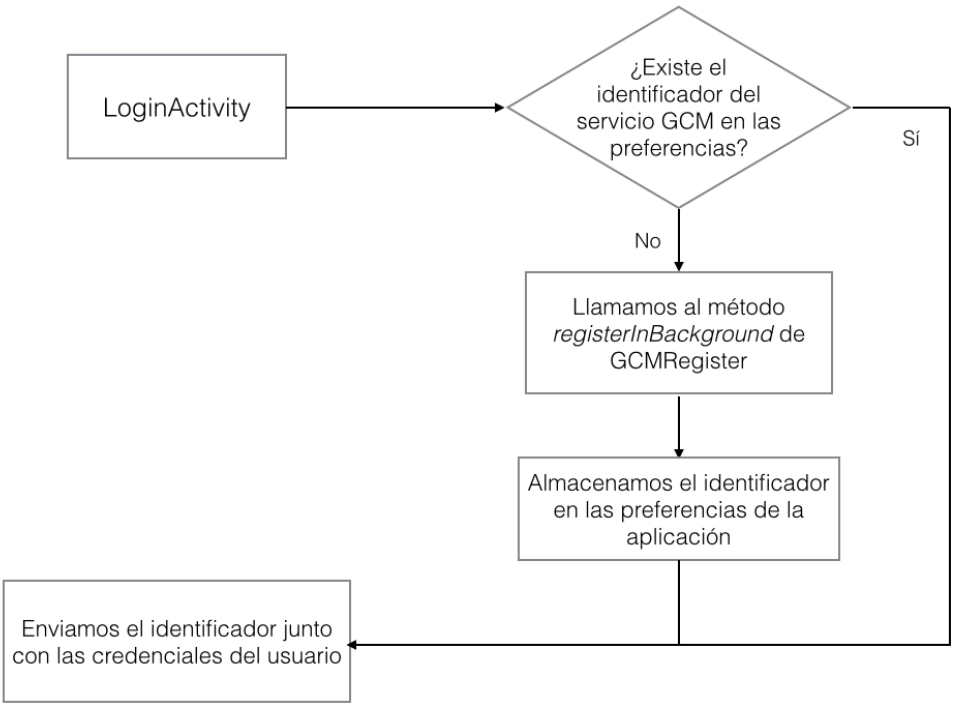

Figura 5.5 Digrama de flujo del registro de un dispositivo en el servicio GCM.

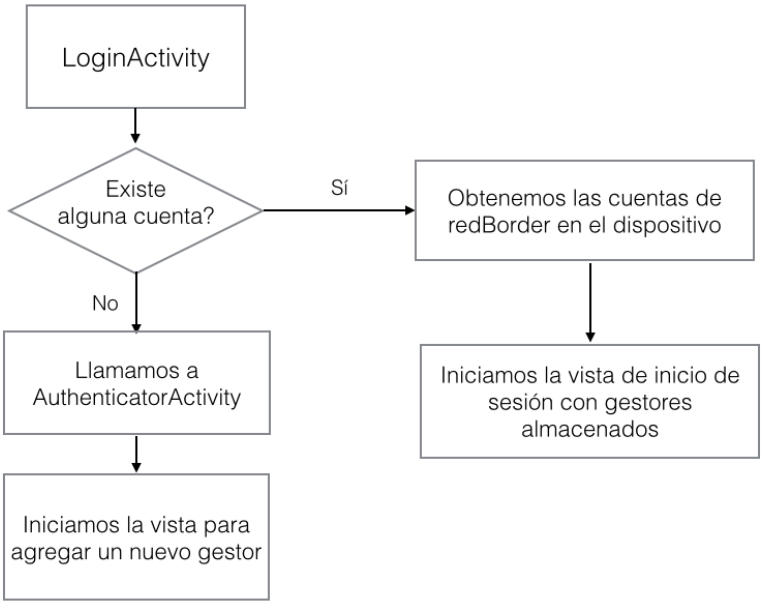

Figura 5.6 Digrama de flujo del inicio de sesión de redBorder mobile.

Código 5.3 Lanzamiento de la actividad de creación de nueva cuenta.

```
if (accounts.length == 0) {
  // ops! We don't have any, let's create one
  Intent auth = new Intent(this, AuthenticatorActivity.class);
  auth.putExtra(AuthenticatorActivity.ARG_IS_ADDING_NEW_ACCOUNT, true);
  startActivity(auth);
}
```

La clase `AuthenticatorActivity` se encarga de mostrar el formulario expuesto en la Subsección 4.4.1, y de crear la cuenta en el sistema si la autenticación es correcta. Una vez que el servidor retorna el asentimiento de autenticación y los valores del usuario, el Código 5.4 se encarga de crear la cuenta en el dispositivo.

Código 5.4 Método para la creación de una cuenta en Android.

```
private void finishLogin(Intent intent) {
  // Name of the accounts in the system
  String accountName = intent.getStringExtra(AccountManager.KEY_ACCOUNT_NAME);
  // Init parameters from server response
  String accountPassword = intent.getStringExtra(PARAM_USER_PASS);
  String server_key = intent.getStringExtra(ARG_SERVER_KEY);
  String server_modules = intent.getStringExtra(ARG_SERVER_MODULES);
  // Create the account
  final Account account = new Account(accountName, intent.getStringExtra(
      AccountManager.KEY_ACCOUNT_TYPE));

  // Set the token
  String authtoken = intent.getStringExtra(AccountManager.KEY_AUTHTOKEN);
  String authtokenType = "READ-ONLY";

  if (getIntent().getBooleanExtra(ARG_IS_ADDING_NEW_ACCOUNT, false)) {
    // Creating the account on the device and setting the auth token we got
    // (Not setting the auth token will cause another call to the server to
        authenticate the user)
    mAccountManager.addAccountExplicitly(account, accountPassword, null);
    mAccountManager.setUserData(account, ARG_EMAIL, email);
    mAccountManager.setUserData(account, ARG_DOMAIN, domain);
    mAccountManager.setUserData(account, ARG_SERVER_KEY, server_key);
    mAccountManager.setUserData(account, ARG_SERVER_MODULES, server_modules);
    mAccountManager.setAuthToken(account, authtokenType, authtoken);
  } else {
    // Updating credentials
    mAccountManager.setPassword(account, accountPassword);
    mAccountManager.setAuthToken(account, authtokenType, authtoken);
  }

  setAccountAuthenticatorResult(intent.getExtras());
  setResult(RESULT_OK, intent);

  // Return to login
  Intent login = new Intent(this, LoginActivity.class);
  startActivity(login);
  finish();
}
```

Todas las clases necesarias para la autenticación y gestión de cuentas se encuentra dentro del paquete `net.redborder.redborder.accounts`. El paquete `android.accounts` incluye las clases de las que se debe heredar para trabajar con cuentas en Android®.

La clase `AuthenticationServer` realiza las llamadas al gestor de redBorder®, instanciando la clase `User` con los datos devueltos por el servidor si la autenticación ha sido correcta. En el caso de que esta no sea correcta, se muestra un mensaje en la interfaz incluyendo la cadena de texto del error retornado por el servidor.

La clase principal para la gestión de las cuentas de redBorder una vez creadas es `Authenticator`. Esta clase hereda de `AbstractAccountAuthenticator` y define los siguientes métodos:

- `addAccount`: instancia la clase `AuthenticatorActivity` con el formulario y la lógica para la creación de la cuenta de usuario.
- `confirmCredentials`: también instancia la clase `AuthenticatorActivity`, pero esta vez con el objetivo de que el usuario vuelva a introducir las credenciales. Se utiliza para actualizar las credenciales cuando el gestor devuelve un error por usuario no autorizado.
- `getAuthToken`: retorna el token de autenticación almacenado en la cuenta seleccionada.

Esta clase a su vez, utiliza los métodos de `AuthenticationServer` para las llamadas al servidor. El código de `Authenticator` está incluído en el Apéndice I.

Por último, la clase `AuthenticationService` se encarga de recibir las creaciones de cuentas de tipo redBorder® desde los ajustes de nuestro dispositivo Android®.

5.5.2 Vistas de RAW

La implementación de estas vistas incluye tanto al fichero principal `RAWActivity` como a sus fragmentos, incluidos en el paquete `net.redborder.redborder.raw`.

La actividad `RAWActivity` se encarga de la representación de los menús laterales y de la gestión de fragmentos. Ampliando la información de la Subsección 5.4.5, el diagrama de iniciación de esta actividad es el mostrado en la Figura 5.7.

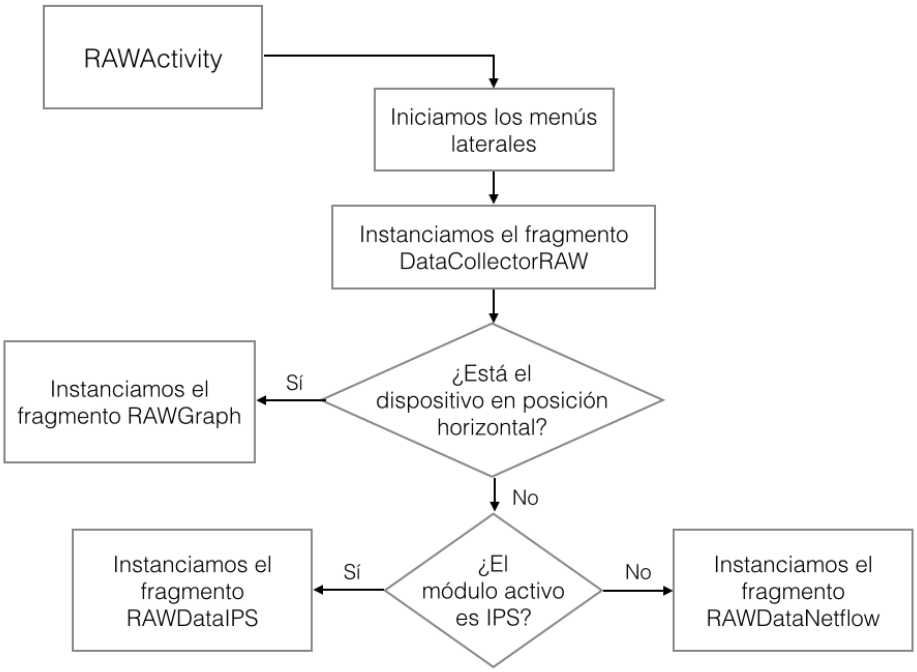

Figura 5.7 Digrama de flujo de RAWActivity.

Cuando instanciamos los fragmentos, iniciamos su ciclo de vida y los agregamos a un elemento de la interfaz mediante el gestor de fragmentos de la actividad.

Por ejemplo, para el caso de que el dispositivo se encuentre en posición vertical, el Código 5.5 del método `onCreate` de `RAWActivity` instancia uno de los dos fragmentos y posteriormente lo asocia al elemento gráfico `R.id.frag_raw`. El código 5.6 muestra una parte el contenido del fichero de la vista de RAW, donde se puede ver el que la etiqueta `FrameLayout` es en la que se incluye el fragmento anterior.

Al llamar al método `commit` el fragmento inicia su ciclo de vida, cargando su propia vista e incluyéndola en el elemnto gráfico al que ha sido asignado.

Código 5.5 Instanciación y agregación de un fragmento a la actividad.

```
// Check the product
if (Product.IPS == product) {
  frag_data = new RAWDataIPS();
} else {
  frag_data = new RAWDataNetflow();
}

// Attach the fragment
fm.beginTransaction().add(R.id.frag_raw, frag_data, FRAG_RAW_DATA_TAG).commit()
    ;
```

Código 5.6 Fragmento del fichero de vista activity_raw.xml.

```
<LinearLayout
  android:id="@+id/content_frame"
  android:layout_width="match_parent"
  android:orientation="vertical"
  android:background="@color/bg_event_list"
  android:layout_height="match_parent">

  <FrameLayout
    android:id="@+id/frag_raw"
    android:layout_weight="1"
    android:layout_width="match_parent"
    android:layout_height="0dp" />

</LinearLayout>
```

Los fragmentos deben de comunicarse con la actividad ya que estos solicitan datos al colector de datos. Para ello, `RAWData` (clase padre de *RAWDataIPS* y *RAWDataNetflow*) define una interfaz de Java® que la actividad debe de implementar. Los métodos de esta interfaz se corresponden con interacciones del usuario con la vista que requieren datos del colector. El código 5.7 muestra la definición de la interfaz.

Código 5.7 Interaz definida por `RAWData`.

```
public static interface RawDataCallbacks {

  /**
   * Method to catch the view event
   * */
  public void onDataViewLoad();

  /**
   * Function to get catch the ips item selected
```

```
   * */
  public void onEventItemSelected(RAWEvent event);

  /**
   * Function to get catch the netflow item selected
   * */
  public void onEventItemSelected(RAWEventFlow event);

  /**
   * Method to get the infinite scroll event
   * */
  public void onInfiniteScroll();
}
```

A su vez `RAWData` también define un método abstracto `setEvents`, que los hijos deben de implementar ya que será el utilizado por la actividad para enviar los datos al fragmento cuando se hayan recibido desde el colector.

Como resumen, el diagrama de flujo de la Figura 5.8 muestra el paso de mensajes desde que el usuario solicita datos mediante una acción hasta que la vista lo representa.

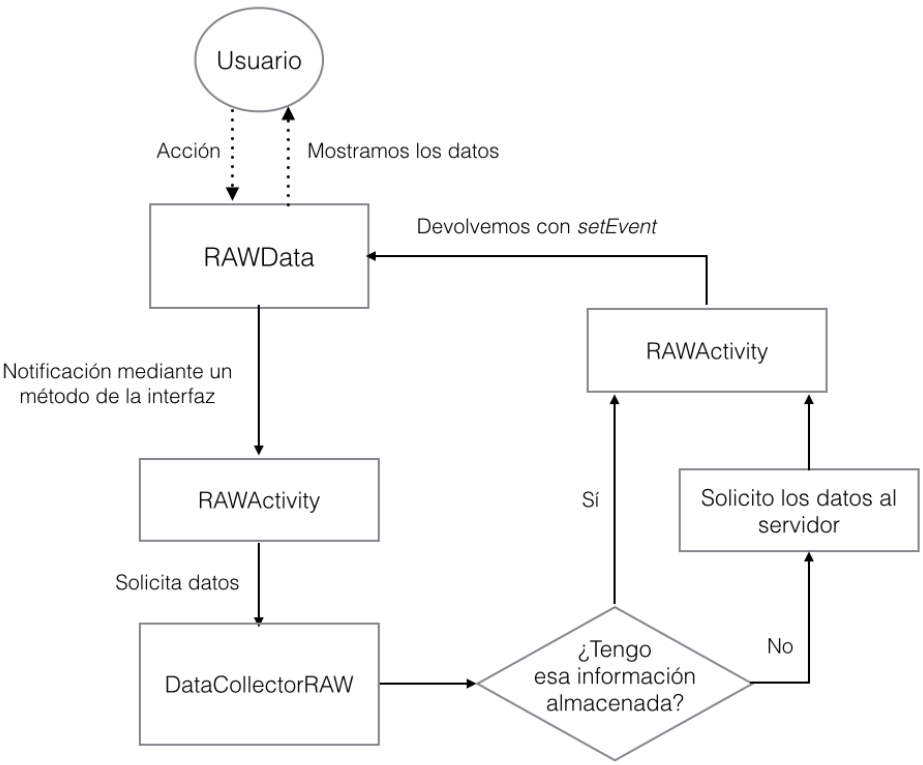

Figura 5.8 Digrama de paso de mensajes entre RAWActivity y sus fragmentos asociados.

El funcionamiento de la vista de gráfico es similar al de la vista lista, sustituyendo los fragmentos de que heredan de `RAWData` por `RAWGraph`. Este fragmento también incluye una interfaz para comunicarse con la actividad, como podemos ver en el Código 5.8. En este caso, el método análogo para `setEvents` es `drawGraph`. Este fragmento instancia la clase `GraphManager` para gestionar los gráficos, como se ha comentado en la Subsección 5.4.6.

Código 5.8 Interaz definida por `RAWGraph`.

```
/**
```

```
 * Interface to get the Load View event
 * */
public static interface onGraphViewLoadListener {
  /**
   * Method to catch the event
   * */
  public void onGraphViewLoad();

  /**
   * Method to request more data
   * */
  public void onRequestData(boolean data, long end_time);
}
```

5.5.3 Vistas de Tops

La lógica funcional de estas vistas es muy similar a la de las vistas RAW en cuanto a la estructura de fragmentos y el paso de mensajes, por lo que esa parte se omite de este capítulo para no duplicar la información.

Como podemos ver en la Subsección 4.6, el usuario dispone de una navegación horizontal para cambiar la ventana temporal de la información que se muestra. Para implementar esta funcionalidad se ha creado la clase `TopPagerAdapter` dentro del paquete `net.redborder.redborder.top`.

Esta clase hereda de `FragmentStatePagerAdapter` e inicializa un fragmento para la lista y otro para la gráfica por cada ventana temporal. Este adaptador funciona de manera similar a los comentados en la Subsección 5.4.4. A medida que el usuario navega cambiando la ventana temporal, se van cargando los fragmentos y mostrando la información.

La clase `FragmentStatePagerAdapter` se encarga de todo el proceso de gestión de los fragmentos. En la clase `TopPagerAdapter` se han sobrescrito los siguientes métodos:

- `getItem`: devuelve el fragmento inicializado en función de la posición del dispositivo. Si este es vertical, devolverá una instancia de la clase `TopData` y si su posición es horizontal, la clase del fragmento será `TopGraph`.
- `getCount`: número de elementos en el adaptador. Este se corresponde con el número de ventanas temporales disponibles siendo este valor igual a siete.
- `getPageTitle`: retorna los nombres de los paneles en función de la posición actual del usuario.

Se incluye el código de `TopPagerAdapter` en el Apéndice J.

5.5.4 Monitorización de nodos

Para la monitorización de nodos distingimos dos actividades: `MonitorActivity` y `MonitorActivityDetail`.

Vistas de lista y mapa

`MonitorActivity` muestra el árbol de nodos y la vista de mapa que incluye todo los nodos geoposicionados. Para ello utiliza el paquete de `net.redborder.redborder.monitor`, que incluye dos fragmentos.

`MonitorListFragment` es el primero de los fragmentos y utiliza la librería expuesta en Subsección 5.2.4 para listar el árbol de nodos. Este fragmento gestiona la visualización, pero los datos los solicita a la actividad. Para ello, al igual que en casos anteriores, define una interfaz que debe de implementar la actividad. El Código 5.9 contiene la definición de dicha interfaz.

Código 5.9 Interaz definida por `MonitorListFragment`.

```
/**
 * ------------------------------------------------------------
 * Callbacks to call the data_collector
 * ------------------------------------------------------------
 */
public interface MonitorListCallbacks {
```

```
    /**
     * Method to catch the created view event
     */
    public void onDataViewLoad();

    /**
     * Method to catch the click of an event
     */
    public void onClickMonitorDevice(String title);
}

// Callbacks
private MonitorListCallbacks mCallbacks;
```

El método `onDataViewLoad` de la interfaz notifica a la actividad que la interfaz del fragmento ya ha sido procesada y este está listo para recibir los datos. La ventaja de utilizar este paso de mensajes es que al cambiar a la vista de mapa, el fragmento cambia, pero la actividad no, por lo que una vez obtenidos los nodos, no hace falta volver a solicitarlos al volver a la vista de árbol. De este modo, los nodos quedan almacenados en la variable `devices`, como podemos ver en el Código 5.10.

Código 5.10 Función `onDataViewLoad` definida en `MonitorActivity`.

```
@Override
public void onDataViewLoad() {

    invalidateOptionsMenu();

    if(devices == null || devices.getDevices().isEmpty()) {
        getDevices();
    } else {
        monitorList.setDevices(devices);
    }
}
```

El segundo fragmento, `MonitorMapFragment`, representa la vista de mapas. Para ello utiliza la librería *Google Maps Android* expuesta en la Subsección 5.2.3. El funcionamiento de paso de mensajes es similar al utiliado con la vista de lista, ya que también se define una interfaz y los datos se obtienen desde la actividad y no en el fragmento.

Código 5.11 Interaz definida por `MonitorMapFragment`.

```
/**
 * ------------------------------------------------------------
 * Callbacks to call the data_collector
 * ------------------------------------------------------------
 */
public interface MonitorMapCallbacks {

    /**
     * Method to catch the view event
     */
    public void onMapViewLoad();

    /**
     * Method called then the user click on the marker to open a device
     * */
```

```
    public void onMarkerInfoClick(String title);
 }

// Callbacks
private MonitorMapCallbacks mCallbacks;
```

Este fragmento también incluye funciones para la gestión de los mapas. En nuestro caso hemos definido unas ventanas personalizadas de información al pulsar sobre los marcadores del mapa. Esta ventana incluye información como un icono que identifica al tipo de nodo, así como su nombre y su dirección IP. En el Código 5.12 podemos ver la definición de la clase que representa dichas ventanas.

Código 5.12 Clase para la representación de las ventanas de información en el mapa.

```
/**
 * Custom adapter to set the info view of the markers
 * */
private class customMarkerInfo implements GoogleMap.InfoWindowAdapter {

    @Override
    public View getInfoWindow(Marker marker) {
        return null;
    }

    @Override
    public View getInfoContents(Marker marker) {

        View v = getActivity().getLayoutInflater().inflate(R.layout.custom_info_
            marker, null, false);

        TextView title = (TextView)v.findViewById(R.id.title);
        TextView ip = (TextView)v.findViewById(R.id.ip);
        ImageView icon = (ImageView)v.findViewById(R.id.icon);

        String[] data = marker.getSnippet().split(":");

        title.setText(marker.getTitle());
        ip.setText(data[1]);

        if(data[0].contentEquals("2")){
            // IPS
            icon.setBackgroundResource(R.drawable.ic_ips);
        } else {
            icon.setBackgroundResource(R.drawable.ic_flow);
        }

        return v;
    }
}
```

Otra funcionalidad sobre el mapa es el centrado de este en función de los nodos, para que el usuario pueda visualizar todos los nodos sin tener que reducir el aumento del mapa. El método que se encarga de ello es el mostrado en el Código 5.13.

Código 5.13 Método para centrar el mapa.

```
/**
 * Center the map to view all the markers
```

```
 * */
private void centerInMarkers(){
    if(markers.size() > 1) {
        LatLngBounds.Builder builder = new LatLngBounds.Builder();
        for (Marker m : markers) {
            builder.include(m.getPosition());
        }

        LatLngBounds bounds = builder.build();

        int padding = 180; // offset from edges of the map in pixels

        getMap().moveCamera(CameraUpdateFactory.newLatLngBounds(
                bounds,
                getActivity().getResources().getDisplayMetrics().widthPixels,
                getActivity().getResources().getDisplayMetrics().heightPixels,
                padding));
    } else {
        // We have only 1 marker
        getMap().moveCamera(CameraUpdateFactory.newLatLngZoom(
                markers.get(0).getPosition(),
                15
        ));
    }
}
```

Vistas de detalles

Esta vista es directamente gestionada por la actividad `MonitorDetailActivity`, sin agregar ningún fragmento. La actividad se encarga de realizar las llamadas al servidor y representar los datos en la interfaz.

Para obtener el efecto del gráfico circular que no realiza una vuelta completa, como podemos ver en la Subsección 4.8.3, se agregó la funcionalidad a la librería `HoloGraph` para poder definir el ángulo máximo de este tipo de gráfico.

Con este cambio, conseguir dicho efecto es tan sencillo como modificar la propiedad `maxAngle` de la instancia de la clase `PieGraph`, como podemos ver en el Código 5.14

Código 5.14 Modificación del ángulo máximo de un gráfico circular.

```
graphCpu.setAnimate(0.175f);
graphCpu.setMaxAngle(225);
graphCpu.setBaseCircle(getResources().getColor(R.color.bg_monitor_graph));
graphCpu.setCustomTotal(100);
```

Todo los cambios de código realizados sobre la librería para introducir está funcionalidad lo tenemos disponible en *https://bitbucket.org/angelmm/holographlibrary/commits/ef8d89f7b6392b11e3323883316535cb73fc372c*.

5.5.5 Alarmas

Las alarmas no poseen una actividad, sino que se adhieren al ciclo de `TopActivity` o `RAWActivity`. Toda la información sobre alarmas viene gestionada por el paquete `net.redborder.redborder.notifications`, más concretamente, por la clase `NotificationFragment`.

Para acceder a esta vista, se utiliza el icono de la campana situado en la barra superior. Tomando como ejemplo la actividad `RAWActivity`, el Código 5.15 muestra el método que es ejecutado al pulsar sobre el icono de alarmas. Este se encarga de iniciar el ciclo de vida del fragmento.

Código 5.15 Método para mostrar la lista de alarmas en la actividad `RAWACtivity`.

```
/**
 * hide or show the notifications
 * */
private void toggleNotifications(){

    if(notifications){
        // Back pressed
        notifications = false;
        onBackPressed();
    } else {
        notifications = true;

        Bundle args = new Bundle();
        args.putSerializable(FragmentArguments.ARG_USER, user);

        // Initiate the frag
        frag_notification = new NotificationFragment();
        frag_notification.setArguments(args);

        getSupportFragmentManager().beginTransaction().replace(R.id.frag_raw,
            frag_notification, FRAG_NOTIFICATIONS).addToBackStack(FRAG_
            NOTIFICATIONS).commit();
    }

}
```

Si la vista de alarmas ya está visible y se pulsa en el icono, lo que ocurre es que se llama al método `onBackPressed()` que provocará que la aplicación vuelva al estado anterior, terminando el ciclo de vida del fragmento. Esto es posible gracias a que cuando se agregó el fragmento a la actividad, se agregó a la pila de estados mediante el método `addToBackStack(FRAG_NOTIFICATIONS)`.

Un problema que nos encontramos al trabajar con varias cuentas en un mismo dispositivo era discernir a que gestor correspondía cada alarma. Para solventar este problema, el gestor define una cadena aleatoria que lo identifica unívocamente llamada *server_key*. Este valor es retornado al iniciar sesión y cuando se recibe una notificación. El modelo de usuario también incluye dicha cadena, como podemos ver en el código del modelo user incluído en el Apéndice C

La clase `NotificationFragment` es la que se encarga de procesar la lista de alarmas, así como de eliminar las notificaciones de la barra de notificaciones y estado. Para la obtención de las alarmas, se utiliza otra clase: `Alarms`. Esta define varios métodos útiles para la gestión de alarmas como obtener las alarmas no vistas para un determinado gestor. El código de esta clase se incluye en el Apéndice K

6 Presupuesto

A continuación se detalla un presupuesto sobre el coste de desarrollo de este proyecto:

Tabla 6.1 Presupuesto del proyecto redBorder Mobile.

Descripción	Precio/Hora	Número de horas	Total (Euros)
Estudio del proyecto, investigación y definición de la estructura	12	20	240
Conceptualización y diseño de la interfaz	12	35	420
Implementación del proyecto	12	200	2400
Mejora continua y soporte	12	60	720
Total		**315**	**3780**

En total en este proyecto se han invertido trecientas quince horas de trabajo.

7 Conclusiones

7.1 Conclusiones

Tras un tiempo con la intención de aprender sobre tecnologías móviles, este proyecto ha sido todo un reto. Desarrollar para Android es relativamente sencillo, pero trabajar en mejoras de rendimiento es más complejo e interesante al mismo tiempo.

Partiendo de una interfaz ya existente, se ha adaptado esta y su lógica de interacción a una aplicación móvil capaz de ofrecer al usuario una vista rápida del estado de su red, así como notificaciones en tiempo real de cualquier problema que pudiera estar ocurriendo.

El tener una interfaz sobre la que basarse es una ventaja y un inconveniente. La conceptualización de ideas y el diseño de la interfaz móvil se ha visto acelerado gracias a esto, pero a la vez, las personas que han utilizado el gestor web ya tienen interiorizado una manera de trabajar. Es por ello que la interacción móvil debía de corresponderse con la del gestor, ya que es lo que esperaban los clientes.

Trabajar en este entorno de trabajo ha sido una gran experiencia. Desde el primer momento me he sentido integrado y escuchado por el equipo, por lo que el proceso desarrollo de esta aplicación ha sido muy satisfactorio a nivel personal y profesional.

La impresión del equipo ha sido muy buena, ya que el los clientes se han visto interesados en la aplicación y han estado consultando el estado actual de esta.

En definitiva, la aplicación ha cumplido su objetivo, que era disponer de una herramienta móvil para la gestión de la plataforma y en consecuencia, agregar valor al producto.

7.2 Líneas de mejora

Como línas de mejora se proponen varias:

- Inclusión de más tipos de gráficos, como gráficos de barras, de área...
- Agregar más módulos de redBorder®, que se agregaron tras el comienzo del proyecto.
- Mejorar la interfaz en tabletas, ya que se podría aprovechar el espacio para mostrar los gráficos y las listas de eventos al mismo tiempo.
- Agregar algunos parámetros de configuración básicos del gestor.
- Modificar el código y desarrollar una aplicación multiplataforma para el funcionamiento en el sistema operativo iOS®, ya que varios clientes preguntaron la existencia de la aplicación en dicha plataforma.

Apéndice A
Fichero de configuración de Gradle

Código A.1 Contenido del fichero build.gradle.

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.3.0'
    }
}

apply plugin: 'android'

repositories {
    mavenCentral()
    maven {
        url "http://clinker.47deg.com/nexus/content/repositories/snapshots"
    }
}

// Ligrerías de la aplicación
dependencies {
    compile 'com.android.support:support-v4:19.1.+'
    compile fileTree(dir: 'libs', include: '*.jar')
    compile 'com.google.android.gms:play-services:4.3.23'
    compile project(':ActiveAndroid')
    compile project(':holographlibrary:HoloGraphLibrary')
    compile project(':android-pull-to-refresh:library')
    compile project(':AnotherExpandableListView')
}

android {
    compileSdkVersion 19
    buildToolsVersion "19.1.0"

    lintOptions {
        checkReleaseBuilds false
        abortOnError false
    }
```

```
    signingConfigs {

        debug {
            // Define la ubicación, nombre y contraseña del fichero de la
            // clave para firmar la aplicación en el entorno de desarrollo.
            // Se omite el contenido de esta por motivos de seguridad.
        }

        signedAPK {
            // Define la ubicación, nombre y contraseña del fichero de la
            // clave para firmar la aplicación en el entorno de producción.
            // Se omite el contenido de esta por motivos de seguridad.
        }
    }

    sourceSets {
        main {
            manifest.srcFile 'AndroidManifest.xml'
            java.srcDirs = ['src']
            resources.srcDirs = ['src']
            aidl.srcDirs = ['src']
            renderscript.srcDirs = ['src']
            res.srcDirs = ['res']
            assets.srcDirs = ['assets']
        }
    }

    buildTypes {

        debug {
            debuggable true
            signingConfig signingConfigs.debug
            proguardFile 'proguard.cfg'
        }

        release {
            debuggable false
            signingConfig signingConfigs.signedAPK
            proguardFile 'proguard.cfg'
        }
    }
}
```

Apéndice B
AndroidManifest.xml

Código B.1 Contenido del fichero AndroidManifest.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.redborder.redborder"
    android:versionCode="250"
    android:versionName="2.5 Stable" >

    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.MANAGE_ACCOUNTS"/>
    <uses-permission android:name="android.permission.USE_CREDENTIALS"/>
    <uses-permission android:name="com.google.android.providers.gsf.permission.
        READ_GSERVICES"/>
    <!-- The following two permissions are not required to use
         Google Maps Android API v2, but are recommended. -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"
        />
    <permission
        android:name="com.example.gcm.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
    <uses-permission android:name="com.example.gcm.permission.C2D_MESSAGE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:name=".Application">
        <uses-library android:name="com.google.android.maps" />
```

```
        <activity
            android:name="net.redborder.redborder.LoginActivity"
            android:label="@string/app_name"
            android:launchMode="singleTop"
            android:theme="@style/Theme.NoTitle" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="net.redborder.redborder.RAWActivity"
            android:configChanges="orientation|screenSize"
            android:label="@string/title_activity_ips" >
        </activity>
        <activity
            android:name="net.redborder.redborder.DetailActivity"
            android:label="@string/title_activity_detail"
            android:launchMode="singleTop"
            android:parentActivityName="net.redborder.redborder.RAWActivity"
            android:screenOrientation="portrait" >

            <!-- The meta-data element is needed for versions lower than 4.1 -->
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="net.redborder.redborder.RAWActivity" />
        </activity>
        <activity
            android:name="net.redborder.redborder.TopActivity"
            android:configChanges="orientation|screenSize"
            android:label="@string/title_activity_top" >
        </activity>
        <activity
            android:name="net.redborder.redborder.StreamActivity"
            android:label="@string/title_activity_stream" >
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />

                <data
                    android:host="redborder.net"
                    android:pathPrefix="/event/"
                    android:scheme="https" />
            </intent-filter>
        </activity>

        <!-- Monitor -->
        <activity
            android:name="net.redborder.redborder.MonitorActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/title_activity_monitor" >
        </activity>
        <activity
            android:name="net.redborder.redborder.MonitorDetailActivity"
```

```
            android:label="@string/title_activity_monitor_detail"
            android:launchMode="singleTop"
            android:parentActivityName="net.redborder.redborder.MonitorActivity"
                 >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="net.redborder.redborder.MonitorActivity" />
        </activity>

        <!-- Authentication service -->
        <activity
            android:name="net.redborder.redborder.accounts.AuthenticatorActivity
                "
            android:label="@string/app_name"
            android:theme="@style/Theme.NoTitle" >
        </activity>

        <service android:name="net.redborder.redborder.accounts.
            AuthenticationService">
            <intent-filter>
                <action android:name="android.accounts.AccountAuthenticator" />
            </intent-filter>
            <meta-data android:name="android.accounts.AccountAuthenticator"
                android:resource="@xml/authenticator" />
        </service>

        <!-- GCM -->
        <receiver
            android:name=".gcm.GcmBroadcastReceiver"
            android:permission="com.google.android.c2dm.permission.SEND" >
            <intent-filter>
                <action android:name="com.google.android.c2dm.intent.RECEIVE" />

                <category android:name="com.example.gcm" />
            </intent-filter>
        </receiver>

        <service android:name=".gcm.GcmIntentService" />

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

        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="SE OMITE POR SEGURIDAD"/>

        <meta-data android:name="AA_DB_NAME" android:value="alarms.db" />
        <meta-data android:name="AA_DB_VERSION" android:value="11" />
    </application>

</manifest>
```

Apéndice C
Modelo de usuarios

Código C.1 Contenido del fichero models/User.java.

```
package net.redborder.redborder.models;

import it.restrung.rest.marshalling.response.AbstractJSONResponse;

public class User extends AbstractJSONResponse {

    // API Key of the user
    private String auth_token;
    // Status of the request
    private Boolean status;
    // Message error
    private String message;
    // Email of the user
    private String email;
    // Login
    private String login;
    // Name of the user
    private String name;
    // Domain of the current server
    private String domain;
    // Server key
    private String server_key;
    // Modules available on the server
    private String server_modules;

    // GET functions ------------------------------

    public String getName() {
        return name;
    }
    public String getEmail(){
        return email;
    }
    public String getDomain() {
        return domain;
    }
    public String getAuth_token() {
        return auth_token;
```

```
    }
    public Boolean getStatus() {
        return status;
    }
    public String getLogin() {return login; }
    public String getMessage() {
        return message;
    }
    public String getServer_key() {
        return server_key;
    }
    public String getServer_modules() {
        return server_modules;
    }
    // SET functions ---------------------------------

    public void setStatus(Boolean status) {
        this.status = status;
    }
    public void setAuth_token(String auth_token) {
        this.auth_token = auth_token;
    }
    public void setEmail(String email){
        this.email = email;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setDomain(String domain) {
        this.domain = domain;
    }
    public void setLogin(String login) { this.login = login; }
    public void setMessage(String message) {
        this.message = message;
    }
    public void setServer_key(String server_key) {
        this.server_key = server_key;
    }
    public void setServer_modules(String server_modules) {
        this.server_modules = server_modules;
    }
}
```

Apéndice D
Modelo de alarmas

Código D.1 Contenido del fichero models/alarm.java.

```
package net.redborder.redborder.models;

import com.activeandroid.Model;
import com.activeandroid.annotation.Column;
import com.activeandroid.annotation.Table;

/**
 * Alarm model of the application
 */
@Table(name = "Alarms")
public class Alarm extends Model {

    @Column(name = "Name")
    public String name;

    @Column(name = "Upper_limit")
    public float upper_limit;

    @Column(name = "Bottom_limit")
    public float bottom_limit;

    @Column(name = "Upper_units")
    public String upper_units;

    @Column(name = "Bottom_units")
    public String bottom_units;

    @Column(name = "Value")
    public float value;

    @Column(name = "Start_time")
    public long start_time;

    @Column(name = "End_time")
    public long end_time;

    @Column(name = "Server_key")
    public String server_key;
```

```
    @Column(name = "Severity_colour")
    public String severity_colour;

    /**
     * "upper" or "bottom"
     * */
    @Column(name = "Type")
    public String type;

    /**
     * 0 - not view
     * 1 - view
     * */
    @Column(name = "View")
    public int view;


}
```

Apéndice E
Modelo para elementos del menú

Código E.1 Contenido del fichero models/MenuItemCustom.java.

```
package net.redborder.redborder.models;

import android.graphics.Bitmap;

public class MenuItemCustom {

    // Identifier of res/string.xml source
    public int title;
    // No icon = -1
    public int icon;
    // String of the username
    public String user;
    // Bitmap received from gravatar
    public Bitmap avatar;
    // True -> header, false -> item
    public boolean isHeader;
    // True -> Avatar
    public boolean isAvatar;
    // True if is selected
    public boolean selected = false;

    // General constructor
    public MenuItemCustom(int a_title, String a_user, int a_icon, Bitmap a_
        avatar,boolean a_isHeader, boolean a_isSelected, boolean a_isAvatar){
        this.title = a_title;
        this.user = a_user;
        this.icon = a_icon;
        this.avatar = a_avatar;
        this.isHeader = a_isHeader;
        this.selected = a_isSelected;
        this.isAvatar = a_isAvatar;
    }

    // Constructor to build an item with an icon
    public MenuItemCustom(int a_title, int a_icon){
        this(a_title, null,a_icon, null, false, false, false);
    }
```

```
    // Constructor to build an item
    public MenuItemCustom(int a_title){
        this(a_title, null,-1, null, false, false, false);
    }

    // Constructor to build an avatar
    public MenuItemCustom(String user, Bitmap avatar){
        this(-1, user, -1, avatar, false, false, true);
    }
}
```

Apéndice F
Colector de datos para la vistas de RAW

Código F.1 Contenido del fichero dataCollector/DataCollectorRAW.java.

```
package net.redborder.redborder.dataCollector;

import android.app.Activity;
import android.content.Context;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.util.Log;

import net.redborder.redborder.Product;
import net.redborder.redborder.collections.RAWEventCollection;
import net.redborder.redborder.collections.RAWEventFlowCollection;
import net.redborder.redborder.graph.GraphCollection;
import net.redborder.redborder.graph.GraphSerie;
import net.redborder.redborder.models.User;
import net.redborder.redborder.rest.RAWStrings;
import net.redborder.redborder.rest.RequestParams;
import net.redborder.redborder.utils.FragmentArguments;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

import it.restrung.rest.async.asynctasks.APIGetAsyncTask;
import it.restrung.rest.client.APICredentialsDelegate;
import it.restrung.rest.client.ContextAwareAPIDelegate;
import it.restrung.rest.client.RestClient;
import it.restrung.rest.client.RestClientFactory;
import it.restrung.rest.exceptions.InvalidCredentialsException;

/**
 * Class to retain and get the data for other fragments.
 * Author: Angel M
 * Version: 1.0
 */
public class DataCollectorRAW extends Fragment {

    /**
```

```
     * Type events to configure the callback
     */
    private static int TYPE_RAW_EVENTS = 1;
    public static int TYPE_RAW_GRAPH = 2;

    // Data from users
    private User user;
    private int granularity;
    private int product;
    private int serie;

    // List of asynctask
    private List<AsyncTask> tasks;

    /** ----------------------------------------------------
     * Requests variables and interfaces
     * ---------------------------------------------------- */

    /**
     * Interface to connect the callbacks to the
     * main activity.
     */
    public static interface requestCallbacks {

        /**
         * The same methods in asynctask.
         */
        void onPreExecute();

        void onProgressUpdate(int percent);

        void onCancelled();

        void onCredentialsError();

        void onPostExecute(int granularity, int product, int type, boolean clear
            , boolean finish);

    }

    /**
     * Interface to connect the graph request to the main activity
     */
    public static interface graphCallbacks {

        /**
         * Return the data
         */
        void onGraphData(GraphCollection data);

    }

    private graphCallbacks mgraphCallbacks;
    private requestCallbacks mCallbacks;

    /** ----------------------------------------------------
     * Fragment live cycle
```

```
     * ------------------------------------------------*/

    /**
     * Detect the onAttach event to the main activity,
     * we must detect if the activity are implementing
     * the callbacks
     */
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        Log.d("DataCollectorRAW", "onAttach()");

        // Check the callbacks
        if (!(activity instanceof requestCallbacks) || !(activity instanceof
            graphCallbacks)) {
            // Throws a exception
            throw new IllegalStateException("Activity must implement the
                callbacks interface.");
        } else {
            // Set the callbacks
            this.mCallbacks = (requestCallbacks) activity;
            this.mgraphCallbacks = (graphCallbacks) activity;
        }
    }

    /**
     * This method is called only once when the Fragment is first created.
     * We retain this fragment on configChanges
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Get the data from the Bundle
        Bundle args = getArguments();

        user = (User)args.getSerializable(FragmentArguments.ARG_USER);
        granularity = args.getInt(FragmentArguments.ARG_GRANULARITY);
        product = args.getInt(FragmentArguments.ARG_PRODUCT);
        serie = args.getInt(FragmentArguments.ARG_SERIE);

        tasks = new ArrayList<AsyncTask>();

        // Retain the fragment
        setRetainInstance(true);
    }

    /**
     * This method called when the the fragment is destroyed, not retained
     */
    @Override
    public void onDestroy() {
        super.onDestroy();

        // Cancel actual tasks
        cancel();
```

```
    }

    /**
     * ------------------------------------------------------
     * Data methods
     * ------------------------------------------------------
     */

    // Granularity
    public void setGranularity(int granularity) {
        this.granularity = granularity;
        // Clean the offset
        this.offset = 0;
    }

    // User
    public void setUser(User user) {
        this.user = user;
    }

    // Product
    public void setProduct(int product) {
        this.product = product;
    }

    /** ------------------------------------------------------------------
     * Task fragments API
     * ------------------------------------------------------------------- */

    /**
     * Stop the download taks if are someone active
     */
    public void cancel() {

        // Stop all non-finished tasks
        for(AsyncTask task : tasks){

            // Stop all the task needed
            if(task.getStatus() != AsyncTask.Status.FINISHED && !task.
                isCancelled()) {
                task.cancel(true);
            }
        }

        // Clean the list
        tasks.clear();
    }

    /**
     * ------------------------------------------------------------------
     * Request methods based on Restrung
     * ------------------------------------------------------------------
     */

    // Data IPS and Flow
    public RAWEventCollection events;
    public RAWEventFlowCollection events_flow;
```

```
private int offset = 0;
private long end_time;

/**
 * Get the RAW events from a before request. If clean is true, offset and
     end_time are cleaned
 *
 * @param clean If true, clean the offset, the end_time and the list.
 */
public void getRawEvents(final boolean clean) {

    // Request the parameters
    RequestParams params = new RequestParams(user.getDomain() + "/api/" +
        Product.getProduct(product));

    // Insert the auth_token of the user
    params.addNewParam("auth_token", user.getAuth_token());
    params.addNewParam("offset", String.valueOf(offset));
    params.addNewParam("granularity", RAWStrings.getRAWGranularity(
        granularity));

    // We don't need to send the end_time at first request
    if (!clean) {
        params.addNewParam("end_time", String.valueOf(end_time));
    }

    if (product == Product.IPS) {

        // Initiate the callback
        ContextAwareAPIDelegate callback = new ContextAwareAPIDelegate<
            RAWEventCollection>(getActivity(), RAWEventCollection.class) {

            @Override
            public void onResults(RAWEventCollection response) {
                //handle results here in the main thread

                // Receive the events
                events = response;
                offset = Integer.parseInt(response.getOffset());
                end_time = Long.parseLong(response.getEnd_time());

                if (mCallbacks != null) {
                    mCallbacks.onPostExecute(granularity, product, TYPE_RAW_
                        EVENTS, clean, false);
                } else {
                    Log.d("Callback", "null");
                }

            }

            @Override
            public void onError(Throwable e) {
                //handle errors here in the main thread

                // The app can't access to the server
                // Show an error
                // FIXME delete the trace on final APP
```

```
                e.printStackTrace();
            }

        };

        // defatlt error on credentials
        // FIXME Add a method to control this on raw to stop the loading
            item
        APICredentialsDelegate credentialsDelegate = new
            APICredentialsDelegate() {
            @Override
            public void onInvalidCredentials(InvalidCredentialsException e) {

                mCallbacks.onCredentialsError();

            }
        };

        // Get the request
        executeAsync(params.getUrl(), callback, credentialsDelegate, new
            Object());

    } else {
        // Initiate the callback
        ContextAwareAPIDelegate callback = new ContextAwareAPIDelegate<
            RAWEventFlowCollection>(getActivity(), RAWEventFlowCollection.
            class) {

            @Override
            public void onResults(RAWEventFlowCollection response) {
                //handle results here in the main thread

                if (response.getEvents() != null && !response.getEvents().
                    isEmpty()) {
                    // Receive the events
                    events_flow = response;
                    offset = Integer.parseInt(response.getOffset());
                    end_time = Long.parseLong(response.getEnd_time());

                    if (mCallbacks != null) {
                        mCallbacks.onPostExecute(granularity, product, TYPE_RAW
                            _EVENTS, clean, false);
                    } else {
                        Log.d("Callback", "null");
                    }

                } else {
                    // We don't receive any data
                    // FIXME send an error or a exception
                }
            }

            @Override
            public void onError(Throwable e) {
                //handle errors here in the main thread

                // The app can't access to the server
```

```
                // Show an error


                // FIXME delete the trace on final APP
                e.printStackTrace();
            }
        };

        APICredentialsDelegate credentialsDelegate = new
            APICredentialsDelegate() {
            @Override
            public void onInvalidCredentials(InvalidCredentialsException e) {
                mCallbacks.onCredentialsError();
            }
        };

        // Get the request
        executeAsync(params.getUrl(), callback, credentialsDelegate, new
            Object());
    }
}

/**
 * Refresh the RAW events on the list. Load based a new granularity.
 */
public void getRefreshRawEvents() {
    getRawEvents(true);
}

/**
 * Get the raw graph events from the server
 */
void getRawGraph(long end_time) {

    // Request the parameters
    RequestParams params = new RequestParams(user.getDomain() + "/api/" +
        Product.getProduct(product) +
            "/" + Product.getSerie(serie));

    // Insert the auth_token of the user
    params.addNewParam("auth_token", user.getAuth_token());
    params.addNewParam("granularity", RAWStrings.getRAWGranularity(this.
        granularity));

    if (end_time > 0) {
        params.addNewParam("end_time", String.valueOf(end_time));
    }

    Log.i("Request", params.getUrl());

    // Initiate the callback
    ContextAwareAPIDelegate callback = new ContextAwareAPIDelegate<
        GraphCollection>(getActivity(), GraphCollection.class) {

        @Override
        public void onResults(GraphCollection response) {
            //handle results here in the main thread
```

```
            if (mgraphCallbacks != null) {

                if (product == Product.NETFLOW) {

                    // Get the seconds of the granularity
                    int granInSeconds = RAWStrings.getSecondsFromGranularity(
                        granularity);

                    // Divide the elements
                    for (GraphSerie item : response.getSerie()) {

                        long last = item.getEvent().getEvents() * 8;
                        double newValue = (double) last / granInSeconds;

                        //item.getEvent().setEvents(Math.round(newValue));
                        item.getPoint().value = Math.round(newValue);

                    }
                }
                mgraphCallbacks.onGraphData(response);
            } else {
                Log.d("Callback", "null");
            }
        }

        @Override
        public void onError(Throwable e) {
            //handle errors here in the main thread

            // The app can't access to the server
            // Show an error

            // FIXME delete the trace on final APP
            e.printStackTrace();
        }
    };

    APICredentialsDelegate credentialsDelegate = new APICredentialsDelegate
        () {
        @Override
        public void onInvalidCredentials(InvalidCredentialsException e) {
            mCallbacks.onCredentialsError();
        }
    };
    // Get the request
    executeAsync(params.getUrl(), callback, credentialsDelegate, new Object
        ());
}

/**
 * This method receive the params to initiate a asynctask using the appsly
     library.
 * It adds the task to the list to cancel if is needed.
 * */
private void executeAsync(String url, ContextAwareAPIDelegate callback,
    APICredentialsDelegate credentials, Object params){
```

```
        // Get the request
        APIGetAsyncTask getRequest = new APIGetAsyncTask(url, callback,
            credentials, params);

        // Add the request to the list
        tasks.add(getRequest);

        // Execute!
        getRequest.execute();
    }

    public void getRawGraph() {
        getRawGraph(0);
    }

    public void getRawGraphOld(long end_time) {
        getRawGraph(end_time);
    }
}
```

Apéndice G
Gestor para la renderización de gráficos

Código G.1 Contenido del fichero graph/GraphManager.java.

```
package net.redborder.redborder.graph;


import android.content.Context;
import android.support.v4.app.Fragment;
import android.view.MotionEvent;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.Toast;

import com.echo.holographlibrary.Line;
import com.echo.holographlibrary.LineGraph;
import com.echo.holographlibrary.representation.DatePoints;
import com.echo.holographlibrary.scale.Scale;

import net.redborder.redborder.R;
import net.redborder.redborder.dataCollector.DataCollectorTop;
import net.redborder.redborder.rest.RAWStrings;
import net.redborder.redborder.rest.TopStrings;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collections;
import java.util.List;
import java.util.TimeZone;

/**
 * Class created to manage the graph in the app.
 * <p/>
 * This class receive the context of the app and the graph ID to show.
 */
public class GraphManager {

    // Invalid id for touch event
    private static final int INVALID_POINTER_ID = -1;
    private static final int DISTANCE_TO_SLIDE_LEFT = 200;
```

```
// Context and reference
private Context context;
private LineGraph graph;

// Offset and points to load more events
private List<DatePoints.DatePoint> points;
private int offset = 1;

// Granularity in RAW
private int granularity;

// Range in top
private int range;

// Loader wrap
private LinearLayout loader;
private LinearLayout advice;

// Gesture callbacks
private onGestureListener gestureListener;

private int mNumberPointers;
private float mTotalDistance = 0;
private float mMovement = 0;

private float mLastTouchX;
private float mLastTouchY;
private int mActivePointerId;

// Representation
private DatePoints represent;

/**
 * Initiate the class with the context and the reference of the graph
 *
 * @param cont Context of the application
 * @param frag fragment to receive the callbacks
 * @param graphm LineGraph reference
 */
public GraphManager(Context cont, Fragment frag, LineGraph graphm) {

    // Save the context and reference
    this.context = cont;
    this.graph = graphm;

    // Check the callbacks
    if (frag instanceof onGestureListener) {
        // Throws a exception
        this.gestureListener = (onGestureListener) frag;
        this.graph.setOnTouchEventListener(new onTouchEventListener());
    }

    // Set the formatter
    SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss 'UTC
        '");
    format.setTimeZone(TimeZone.getTimeZone("UTC"));
```

```
        // Initiate the representation
        represent = new DatePoints(
                format,
                graph);

        // Hold the Y axis when the windows change
        represent.setHoldY(true);
    }

    /**
     * Get the data of RAW and show it
     *
     * @param granularity Granularity in RAW
     */
    public boolean getRawGraph(int granularity, GraphCollection data) {

        // Save granularity
        this.granularity = granularity;

        // Clean the graph
        represent.clearLines();

        // Set the events
        if (data.getSerie() != null && !data.getSerie().isEmpty()) {
            // Set the items in the graph

            // Only one line
            Line line = new Line();

            // Revert the order
            Collections.reverse(data.getSerie());

            // Load the first list
            points = new ArrayList<DatePoints.DatePoint>();

            for (GraphSerie serie : data.getSerie()) {
                points.add(serie.getPoint());
            }

            // default value of X values
            int typeXValues;

            // String for start time
            String start_timestamp;

            // Set the formatter adapted to the timezone
            SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:
                ss 'UTC'");
            formatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            // start time
            Calendar calendar = Calendar.getInstance(TimeZone.getTimeZone("UTC")
                );

            // end time
            Calendar end = Calendar.getInstance(TimeZone.getTimeZone("UTC"));
```

```
/**
 * When the user select more graph we must get the data to the
     correct
 * range, so we must initiate the time to current and use offset to
 * subtract the time: add(..., ... * offset)
 * */

// Set the start time in the graph based on the Range
switch (granularity) {
    case RAWStrings.GRANULARITY_1_MINUTE:

        typeXValues = LineGraph.HOURS_AND_MINUTES;

        // Set the times
        end.add(Calendar.HOUR, -3 * (offset -1));
        calendar.add(Calendar.HOUR, -3 * offset);
        start_timestamp = formatter.format(calendar.getTime());

        break;
    case RAWStrings.GRANULARITY_5_MINUTE:

        typeXValues = LineGraph.HOURS_AND_MINUTES;

        end.add(Calendar.DAY_OF_YEAR, -1 * (offset - 1));
        calendar.add(Calendar.DAY_OF_YEAR, -1 * offset);
        start_timestamp = formatter.format(calendar.getTime());

        break;
    case RAWStrings.GRANULARITY_15_MINUTE:

        typeXValues = LineGraph.DAYS_AND_HOURS;

        end.add(Calendar.DAY_OF_YEAR, -3 * (offset - 1));
        calendar.add(Calendar.DAY_OF_YEAR, -3 * offset);
        start_timestamp = formatter.format(calendar.getTime());

        break;
    case RAWStrings.GRANULARITY_30_MINUTE:

        typeXValues = LineGraph.DAYS_AND_HOURS;

        end.add(Calendar.DAY_OF_YEAR, -5 * (offset - 1));
        calendar.add(Calendar.DAY_OF_YEAR, -5 * offset);
        start_timestamp = formatter.format(calendar.getTime());

        break;
    case RAWStrings.GRANULARITY_1_HOUR:

        typeXValues = LineGraph.DAYS_AND_HOURS;

        end.add(Calendar.DAY_OF_YEAR, -7 * (offset - 1));
        calendar.add(Calendar.DAY_OF_YEAR, -7 * offset);
        start_timestamp = formatter.format(calendar.getTime());

        break;
    case RAWStrings.GRANULARITY_8_HOUR:
```

```
                typeXValues = LineGraph.DAYS_AND_HOURS;

                end.add(Calendar.DAY_OF_YEAR, -120 * (offset - 1));
                calendar.add(Calendar.DAY_OF_YEAR, -120 * offset);
                start_timestamp = formatter.format(calendar.getTime());

                break;
            default:

                typeXValues = LineGraph.DAYS_AND_HOURS;

                end.add(Calendar.YEAR, -1 * (offset - 1));
                calendar.add(Calendar.YEAR, -1 * offset);
                start_timestamp = formatter.format(calendar.getTime());
                break;
        }

        // Add the last and the first point to set the range
        DatePoints.DatePoint first = new DatePoints.DatePoint();
        first.timestamp = start_timestamp;
        first.value = 0;

        DatePoints.DatePoint last = new DatePoints.DatePoint();
        last.timestamp = formatter.format(end.getTime());
        last.value = 0;

        points.add(0, first);
        points.add(last);

        // Add the points
        represent.setPoints(points);

        // Represent
        represent.setRangeTime(calendar, end);
        represent.setTypeXValues(typeXValues);

        // Render the graph
        represent.render();

        // Hide loader
        loader.setVisibility(View.GONE);

    }

    return true;
}

/**
 * Get the data of RAW and show it
 *
 * @param collections Data from the response
 * @param range Range in Top
 * @param filters Filters in this fragment. The array can be empty
 */
public boolean getTopGraph(GraphMultipleCollection collections, int range,
    List<DataCollectorTop.filterTop> filters) {
```

```
        // Save granularity
        this.range = range;

        // Number of series to represent
        int numSeries = 0;

        // boolean no data
        boolean noData = false;

        // List of series
        DatePoints.MultipleDatePoint series = new DatePoints.MultipleDatePoint()
            ;

        // Clean the graph
        graph.removeAllLines();
        graph.invalidate();

        if(collections.getSeries() != null) {

           for (GraphCollection data : collections.getSeries()) {

              // Increment the number of series
              numSeries++;

              if (data.getSerie() != null && !data.getSerie().isEmpty() && !
                  noData) {
                 // Set the items in the graph
                 // Revert the order
                 Collections.reverse(data.getSerie());

                 // Create a new item
                 DatePoints.ListDatePoint listPoints = new DatePoints.
                     ListDatePoint();

                 // Load the first list
                 points = new ArrayList<DatePoints.DatePoint>();

                 for (GraphSerie serie : data.getSerie()) {
                    points.add(serie.getPoint());
                 }

                 // Save the points
                 listPoints.points = points;

                 // Set the color based on filters
                 if (filters.size() > 0) {

                    for (DataCollectorTop.filterTop filter : filters) {

                       if (filter.getValues().get(0).equals(data.getName())) {
                          listPoints.color = filter.getColor();
                       }
                    }

                 }

                 // Add the serie
```

```
                series.series.add(listPoints);

            } else {

                noData = true;

            }

        }
    } else {

        noData = true;

    }

    // Check if we have some data
    if(noData){

        // We don't have data
        // Hide loader
        loader.setVisibility(View.GONE);

        // show an error
        advice.setVisibility(View.VISIBLE);

    } else {

        // we have data, set the scale and go to represent!!

        // default value of X values
        int typeXValues;

        // String for start time
        String start_timestamp;

        // Set the formatter adapted to the timezone
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:
            ss 'UTC'");
        formatter.setTimeZone(TimeZone.getTimeZone("UTC"));

        // start time
        Calendar calendar = Calendar.getInstance(TimeZone.getTimeZone("UTC")
            );

        // end time
        Calendar end = Calendar.getInstance(TimeZone.getTimeZone("UTC"));

        /**
         * When the user select more graph we must get the data to the
             correct
         * range, so we must initiate the time to current and use offset to
         * subtract the time: add(..., ... * offset)
         * */

        // Set the start time in the graph based on the Range
        switch (range) {
            case TopStrings.LAST1:
```

```
                typeXValues = LineGraph.HOURS_AND_MINUTES;

                calendar.add(Calendar.HOUR, -1);
                start_timestamp = formatter.format(calendar.getTime());

                break;
            case TopStrings.LAST3:

                typeXValues = LineGraph.HOURS_AND_MINUTES;

                calendar.add(Calendar.HOUR, -3);
                start_timestamp = formatter.format(calendar.getTime());

                break;
            case TopStrings.LAST24:

                typeXValues = LineGraph.HOURS_AND_MINUTES;

                calendar.add(Calendar.DAY_OF_YEAR, -1);
                start_timestamp = formatter.format(calendar.getTime());

                break;
            case TopStrings.LAST_WEEK:

                typeXValues = LineGraph.DAYS_AND_HOURS;

                calendar.add(Calendar.DAY_OF_YEAR, -7);
                start_timestamp = formatter.format(calendar.getTime());

                break;
            case TopStrings.LAST_MONTH:

                typeXValues = LineGraph.DAYS;

                calendar.add(Calendar.DAY_OF_YEAR, -30);
                start_timestamp = formatter.format(calendar.getTime());

                break;
            case TopStrings.LAST_QUARTER:

                typeXValues = LineGraph.DAYS;

                calendar.add(Calendar.DAY_OF_YEAR, -120);
                start_timestamp = formatter.format(calendar.getTime());

                break;
            default:

                typeXValues = LineGraph.DAYS;

                calendar.add(Calendar.YEAR, -1);
                start_timestamp = formatter.format(calendar.getTime());
                break;
        }

        // Add the last and the first point to set the range
```

```
            DatePoints.DatePoint first = new DatePoints.DatePoint();
            first.timestamp = start_timestamp;
            first.value = 0;

            DatePoints.DatePoint last = new DatePoints.DatePoint();
            last.timestamp = formatter.format(end.getTime());
            last.value = 0;

            for(DatePoints.ListDatePoint listPoints : series.series){
               listPoints.points.add(0, first);
               listPoints.points.add(last);
            }

            // Add the points
            represent.setMultipleSeries(series);

            // Represent
            represent.setRangeTime(calendar, end);
            represent.setTypeXValues(typeXValues);

            // Set multiple series
            represent.setMultiple(true);

            // Render the graph
            represent.render();

            // Hide loader
            loader.setVisibility(View.GONE);


        }

        return true;
    }

    /**
     * Initiate the graph
     */
    public void setGraph(LineGraph graph) {

        // Set the graph
        this.graph = graph;

        // set the touch event listener
        this.graph.setOnTouchEventListener(new onTouchEventListener());

    }

    /**
     * Set the loader to hide
     */
    public void setLoader(LinearLayout bar) {
        loader = bar;
    }

    /**
     * Set the advice
```

```
 * */
public void setAdvice(LinearLayout ad) {
    this.advice = ad;
}

/**
 * Set the scale in the representation
 *
 * */
public void setScale(Scale scale){

    represent.setScaleFactor(scale);

}

/**-------------------------------------------------------------------------

 * Gesture Events
 *
     ------------------------------------------------------------------------
      */

/**
 * Instance of interface to handle the touch event
 */
private class onTouchEventListener implements LineGraph.
    OnTouchEventListener {

    @Override
    public void onTouchEvent(MotionEvent event) {

        // Let the ScaleGestureDetector inspect all events.
        //mScaleDetector.onTouchEvent(event);

        mNumberPointers = event.getPointerCount();

        // Get the action mask
        final int action = event.getActionMasked();

        // Switch the action
        switch (action) {

            // The first point enter in the screen
            case MotionEvent.ACTION_DOWN: {
                final float x = event.getX();
                final float y = event.getY();

                mLastTouchX = x;
                mLastTouchY = y;
                mActivePointerId = event.getPointerId(0);
                break;
            }

            case MotionEvent.ACTION_MOVE: {
                final int pointerIndex = event.findPointerIndex(
                    mActivePointerId);
                final float x = event.getX(pointerIndex);
```

```
                final float y = event.getY(pointerIndex);

                if (mNumberPointers == 1) {

                    mTotalDistance += Math.abs(x - mLastTouchX);
                    mMovement += x - mLastTouchX;

                    //Log.i("Values", String.valueOf(mTotalDistance) + ":" +
                        String.valueOf(mMovement));

                    graph.setX(mMovement);

                    mLastTouchX = x;
                    mLastTouchY = y;

                }

                break;
            }

            case MotionEvent.ACTION_UP: {
                mActivePointerId = INVALID_POINTER_ID;

                if (mMovement >= DISTANCE_TO_SLIDE_LEFT) {

                    // Animate the graph
                    graph.animate()
                            .translationX(graph.getWidth())
                            .setDuration(300)
                            .start();

                    // We must reload the data with more values
                    //Log.i("Reloading", String.valueOf(mTotalDistance) + ","
                        + String.valueOf(mLastTouchX) + "," + String.valueOf(
                        pointInScale(mLastTouchX)));

                    // Remove all lines and
                    if (gestureListener != null) {
                        gestureListener.onLoading();

                        // Increment the offset
                        offset++;

                        // The end_time is the minPoint
                        gestureListener.onRequestData(false, represent.
                            getMinPoint());
                    }

                } else if(mMovement <= -DISTANCE_TO_SLIDE_LEFT && offset > 1){


                    // Cargamos eventos futuros si el offset es mayor que 1
                    if(gestureListener != null){

                        // Animate the graph
                        graph.animate()
                                .translationX(-graph.getWidth())
```

```
                .setDuration(300)
                .start();

        gestureListener.onLoading();

        // Decrement the offset
        offset--;

        // The end_time is the maxPoint + range based on
            granularity
        Calendar end_req = Calendar.getInstance(TimeZone.
            getTimeZone("UTC"));

        // The point was in seconds
        end_req.setTimeInMillis(represent.getMaxPoint()*1000);

        switch(granularity){
            case RAWStrings.GRANULARITY_1_MINUTE:

                // Set the times
                end_req.add(Calendar.HOUR, 3);
                break;
            case RAWStrings.GRANULARITY_5_MINUTE:

                end_req.add(Calendar.DAY_OF_YEAR, 1);
                break;
            case RAWStrings.GRANULARITY_15_MINUTE:

                end_req.add(Calendar.DAY_OF_YEAR, 3);

                break;
            case RAWStrings.GRANULARITY_30_MINUTE:

                end_req.add(Calendar.DAY_OF_YEAR, 5);
                break;
            case RAWStrings.GRANULARITY_1_HOUR:

                end_req.add(Calendar.DAY_OF_YEAR, 7);

                break;
            case RAWStrings.GRANULARITY_8_HOUR:

                end_req.add(Calendar.DAY_OF_YEAR, 120);

                break;
            default:

                end_req.add(Calendar.YEAR, 1);
                break;

        }

        // Request the data
        // SEND IN SECONDS!!!
        gestureListener.onRequestData(false, end_req.
            getTimeInMillis()/1000);
```

```
                    }

                } else if(mMovement <= -DISTANCE_TO_SLIDE_LEFT && offset == 1)
                    {

                    // We try to get more events on the init of the graph

                    // Not enough movement
                    graph.animate()
                           .translationX(0)
                           .setDuration(100)
                           .start();

                    // Show an error
                    Toast.makeText(context,
                           context.getResources().getString(R.string.graph_no_
                               more_data),
                           Toast.LENGTH_LONG).show();
                } else {

                    // Not enough movement
                    graph.animate()
                           .translationX(0)
                           .setDuration(100)
                           .start();
                }

                // Clean distances
                mTotalDistance = 0;
                mMovement = 0;
                mLastTouchX = 0;

                break;
            }

            case MotionEvent.ACTION_CANCEL: {
                mActivePointerId = INVALID_POINTER_ID;
                break;
            }

            case MotionEvent.ACTION_POINTER_UP: {
                final int pointerIndex = (event.getAction() & MotionEvent.
                    ACTION_POINTER_INDEX_MASK)
                       >> MotionEvent.ACTION_POINTER_INDEX_SHIFT;
                final int pointerId = event.getPointerId(pointerIndex);
                if (pointerId == mActivePointerId) {
                    // This was our active pointer going up. Choose a new
                    // active pointer and adjust accordingly.
                    final int newPointerIndex = pointerIndex == 0 ? 1 : 0;
                    mLastTouchX = event.getX(newPointerIndex);
                    mLastTouchY = event.getY(newPointerIndex);
                    mActivePointerId = event.getPointerId(newPointerIndex);
                }
                break;
            }
        }
    }
```

```
    }

    /**
     * Convert a point to the Scale
     */
    public float pointInScale(float f) {
        return (DatePoints.INTERVAL * f) / graph.getWidth();
    }

    /**
     * ------------------------------------------------------
     * Callbacks
     * ------------------------------------------------------
     */
    public static interface onGestureListener {
        // The view is loading
        public void onLoading();
        // The user is requesting new data
        public void onRequestData(boolean before, long end_time);
    }
}
```

Apéndice H
Gestor de registro del servicio GCM

Código H.1 Contenido del fichero gcm/GCMRegister.java.

```
package net.redborder.redborder.gcm;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.os.AsyncTask;
import android.util.Log;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GooglePlayServicesUtil;
import com.google.android.gms.gcm.GoogleCloudMessaging;

import net.redborder.redborder.LoginActivity;

import java.io.IOException;

/**
 * This class get the registration ID to use GCM services
 */
public class GcmRegister {

    // globals vars for GCM
    public static final String EXTRA_MESSAGE = "message";
    public static final String PROPERTY_REG_ID = "registration_id";
    private static final String PROPERTY_APP_VERSION = "appVersion";
    private static final int PLAY_SERVICES_RESOLUTION_REQUEST = 9000;

    // Variables globales para guardar los datos
    public static final String SHARED_SERVIDORES = "servidores";

    /**
     * Tag of log messages
     */
    static final String TAG = "GCM Demo";

    /**
```

```
 * ID to get the registration ID of this user
 */
String SENDER_ID = "22355507711";

// This is needed to send the notifications
GoogleCloudMessaging gcm;

// The registration ID
String regid;

// Context of the app
Context context;
Activity activity;

/**
 * Initiate the GCM register
 *
 * @param context context of the application
 * */
public GcmRegister(Context context, Activity activity){

    this.context = context;
    this.activity = activity;

}

/**
 * Gets the current registration ID for application on GCM service.
 * <p>
 * If result is empty, the app needs to register.
 *
 * @return registration ID, or empty string if there is no existing
 *       registration ID.
 */
public String getRegistrationId() {

    final SharedPreferences prefs = getGCMPreferences(context);
    String registrationId = prefs.getString(PROPERTY_REG_ID, "");
    if (registrationId.isEmpty()) {
        Log.i(TAG, "Registration not found.");
        return "";
    }
    // Check if app was updated; if so, it must clear the registration ID
    // since the existing regID is not guaranteed to work with the new
    // app version.
    int registeredVersion = prefs.getInt(PROPERTY_APP_VERSION, Integer.MIN_
        VALUE);
    int currentVersion = getAppVersion(context);
    if (registeredVersion != currentVersion) {
        Log.i(TAG, "App version changed.");
        return "";
    }

    return registrationId;
}

/**
```

```
     * @return Application's {@code SharedPreferences}.
     */
    private SharedPreferences getGCMPreferences(Context context) {
        // This sample app persists the registration ID in shared preferences,
            but
        // how you store the regID in your app is up to you.
        return context.getSharedPreferences(LoginActivity.class.getSimpleName(),
                Context.MODE_PRIVATE);
    }

    /**
     * @return Application's version code from the {@code PackageManager}.
     */
    private static int getAppVersion(Context context) {
        try {
            PackageInfo packageInfo = context.getPackageManager()
                    .getPackageInfo(context.getPackageName(), 0);
            return packageInfo.versionCode;
        } catch (PackageManager.NameNotFoundException e) {
            // should never happen
            throw new RuntimeException("Could not get package name: " + e);
        }
    }

    /**
     * Stores the registration ID and app versionCode in the application's
     * {@code SharedPreferences}.
     *
     * @param context application's context.
     * @param regId registration ID
     */
    private void storeRegistrationId(Context context, String regId) {
        final SharedPreferences prefs = getGCMPreferences(context);
        int appVersion = getAppVersion(context);
        Log.i(TAG, "Saving regId on app version " + appVersion);
        SharedPreferences.Editor editor = prefs.edit();
        editor.putString(PROPERTY_REG_ID, regId);
        editor.putInt(PROPERTY_APP_VERSION, appVersion);
        editor.commit();
    }

    /**
     * Check the device to make sure it has the Google Play Services APK. If
     * it doesn't, display a dialog that allows users to download the APK from
     * the Google Play Store or enable it in the device's system settings.
     */
    public boolean checkPlayServices() {
        int resultCode = GooglePlayServicesUtil.isGooglePlayServicesAvailable(
            context);
        if (resultCode != ConnectionResult.SUCCESS) {
            if (GooglePlayServicesUtil.isUserRecoverableError(resultCode)) {
                GooglePlayServicesUtil.getErrorDialog(resultCode, activity,
                        PLAY_SERVICES_RESOLUTION_REQUEST).show();
            } else {
                Log.i(TAG, "This device is not supported.");
            }
            return false;
```

```
        }
        return true;
    }

    /**
     * Delete the registration ID
     * */
    public void deleteRegistrationId(){
        final SharedPreferences prefs = getGCMPreferences(context);
        int appVersion = getAppVersion(context);
        Log.i(TAG, "Deleted regId on app version " + appVersion);
        SharedPreferences.Editor editor = prefs.edit();
        editor.remove(PROPERTY_REG_ID);
        editor.putInt(PROPERTY_APP_VERSION, appVersion);
        editor.apply();
    }

    /**
     * Registers the application with GCM servers asynchronously.
     * <p>
     * Stores the registration ID and app versionCode in the application's
     * shared preferences.
     */
    public void registerInBackground() {

        new AsyncTask<Void, Void, String>() {

            @Override
            public String doInBackground(Void... params) {
                String msg = "";
                try {
                    if (gcm == null) {
                        gcm = GoogleCloudMessaging.getInstance(context);
                    }
                    regid = gcm.register(SENDER_ID);
                    msg = "Device registered, registration ID=" + regid;

                    // Persist the regID - no need to register again.
                    storeRegistrationId(context, regid);
                } catch (IOException ex) {
                    msg = "Error :" + ex.getMessage();
                    // If there is an error, don't just keep trying to register.
                    // Require the user to click a button again, or perform
                    // exponential back-off.
                }
                return msg;
            }

            @Override
            protected void onPostExecute(String msg) {

            }

        }.execute(null, null, null);
    }
}
```

Apéndice I
Clase Authenticator

Código I.1 Contenido del fichero accounts/authenticator.java.

```
package net.redborder.redborder.accounts;

import android.accounts.AbstractAccountAuthenticator;
import android.accounts.Account;
import android.accounts.AccountAuthenticatorResponse;
import android.accounts.AccountManager;
import android.accounts.AccountManagerFuture;
import android.accounts.NetworkErrorException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;

import net.redborder.redborder.models.User;

/**
 * Class to manage the redBorder accounts in mobile
 */
class Authenticator extends AbstractAccountAuthenticator {

    private Context mContext;
    private AuthenticationServer mServerAuthenticate;

    /**
     * Basic constructor
     *
     * @param context context of the application
     * */
    public Authenticator(Context context){
        super(context);
        this.mContext = context;

        mServerAuthenticate = new AuthenticationServer(context);
    }

    @Override
```

```
public Bundle editProperties(AccountAuthenticatorResponse
    accountAuthenticatorResponse, String s) {
   return null;
}

/**
 * Initiate the add-account request. This method return a bundle to launch
    de AuthenticatorActivity.
 * */
@Override
public Bundle addAccount(AccountAuthenticatorResponse response,
                     String accountType,
                     String authTokenType,
                     String[] requiredFeatures,
                     Bundle options) throws NetworkErrorException {

   // Initiate the intent
   final Intent intent = new Intent(mContext, AuthenticatorActivity.class);
   // Add the data
   intent.putExtra(AuthenticatorActivity.ARG_ACCOUNT_TYPE, accountType);
   intent.putExtra(AuthenticatorActivity.ARG_AUTH_TYPE, authTokenType);
   intent.putExtra(AuthenticatorActivity.ARG_IS_ADDING_NEW_ACCOUNT, true);
   intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE,
       response);
   // Initiate the bundle and add the intent
   final Bundle bundle = new Bundle();
   bundle.putParcelable(AccountManager.KEY_INTENT, intent);
   // Return the bundle
   return bundle;
}

/*
* Init the form to confirm credentials
***/
@Override
public Bundle confirmCredentials(AccountAuthenticatorResponse
    accountAuthenticatorResponse, Account account, Bundle options) throws
    NetworkErrorException {

   AccountManager am = AccountManager.get(mContext);

   // Initiate the intent
   final Intent intent = new Intent(mContext, AuthenticatorActivity.class);
   // Add the data
   intent.putExtra(AuthenticatorActivity.ARG_NAME, account.name.split("@")
       [0]);
   intent.putExtra(AuthenticatorActivity.ARG_DOMAIN, am.getUserData(account
       , AuthenticatorActivity.ARG_DOMAIN));
   intent.putExtra(AuthenticatorActivity.ARG_IS_CONFIRM_ACCOUNT, true);
   // Initiate the bundle and add the intent
   final Bundle bundle = new Bundle();
   bundle.putParcelable(AccountManager.KEY_INTENT, intent);
   // Return the bundle
   return bundle;
}

/**
```

```
     * Get the authToken
     * */
    @Override
    public Bundle getAuthToken(AccountAuthenticatorResponse response, Account
        account, String authTokenType, Bundle options) throws
        NetworkErrorException {

        // Extract the username and password from the Account Manager, and ask
        // the server for an appropriate AuthToken.
        final AccountManager am = AccountManager.get(mContext);

        // Try to get the authToken from cache
        AccountManagerFuture<Bundle> manager = am.getAuthToken(account,
            authTokenType, new Bundle(), true, null, null);
        String authToken = "";

        try {
            authToken = manager.getResult().getString(AccountManager.KEY_
                AUTHTOKEN);
        }catch (Exception e){
            e.printStackTrace();
        }

        User user;

        // Lets give another try to authenticate the user
        if (TextUtils.isEmpty(authToken)) {
            final String password = am.getPassword(account);
            if (password != null) {

                try {
                    String email = am.getUserData(account, AuthenticatorActivity.
                        ARG_EMAIL);
                    String domain = am.getUserData(account, AuthenticatorActivity.
                        ARG_DOMAIN);

                    // Fix an error with installed apps
                    if(email == null){
                        user = mServerAuthenticate.userSignIn(account.name,
                            password, authTokenType, domain);
                    } else {
                        // Initiate the autentication
                        user = mServerAuthenticate.userSignIn(email, password,
                            authTokenType, domain);
                    }

                    authToken = user.getAuth_token();
                } catch (Exception e){
                    e.printStackTrace();
                }
            }
        }

        // If we get an authToken - we return it
        if (!TextUtils.isEmpty(authToken)) {
            final Bundle result = new Bundle();
            result.putString(AccountManager.KEY_ACCOUNT_NAME, account.name);
```

```
            result.putString(AccountManager.KEY_ACCOUNT_TYPE, account.type);
            result.putString(AccountManager.KEY_AUTHTOKEN, authToken);
            return result;
        }

        // If we get here, then we couldn't access the user's password - so we
        // need to re-prompt them for their credentials. We do that by creating
        // an intent to display our AuthenticatorActivity.
        final Intent intent = new Intent(mContext, AuthenticatorActivity.class);
        intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE,
            response);
        intent.putExtra(AuthenticatorActivity.ARG_ACCOUNT_TYPE, account.type);
        intent.putExtra(AuthenticatorActivity.ARG_AUTH_TYPE, authTokenType);
        final Bundle bundle = new Bundle();
        bundle.putParcelable(AccountManager.KEY_INTENT, intent);
        return bundle;

    }

    // Now, we don't need to define these methods.

    @Override
    public String getAuthTokenLabel(String s) {
        return null;
    }

    @Override
    public Bundle updateCredentials(AccountAuthenticatorResponse
        accountAuthenticatorResponse, Account account, String s, Bundle bundle)
         throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle hasFeatures(AccountAuthenticatorResponse
        accountAuthenticatorResponse, Account account, String[] strings) throws
         NetworkErrorException {
        return null;
    }
}
```

Apéndice J
Adaptador de la vista de Tops

Código J.1 Contenido del fichero top/TopPagerAdapter.java.

```
package net.redborder.redborder.top;

import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

import net.redborder.redborder.Product;
import net.redborder.redborder.R;
import net.redborder.redborder.dataCollector.DataCollectorTop;
import net.redborder.redborder.graph.GraphMultipleCollection;
import net.redborder.redborder.models.TopEvent;
import net.redborder.redborder.rest.RAWStrings;
import net.redborder.redborder.rest.TopStrings;
import net.redborder.redborder.utils.FragmentArguments;

import android.content.Context;
import android.content.res.Configuration;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentStatePagerAdapter;

import com.echo.holographlibrary.scale.ScaleBits;
import com.echo.holographlibrary.scale.ScaleBytes;

public class TopPagerAdapter extends FragmentStatePagerAdapter {

    private static final int NUM_ITEMS = 7;

    private final FragmentManager mFragmentManager;

    private Context context = null;
    private int product;
    private int source;
    private List<TopData> fragment_list;
    private List<TopGraph> fragment_list_graph;

    // Orientation
```

```
private int orientation;

/**
 * Constructor for control the pages adapter
 *
 * @param fm FragmentManager to control the pager
 */
public TopPagerAdapter(FragmentManager fm, Context c, int orientation, int
    product, int source) {

    super(fm);

    this.mFragmentManager = fm;
    this.context = c;
    this.orientation = orientation;
    this.source = source;
    this.product = product;

    // Initiate the list
    this.fragment_list = new ArrayList<TopData>();
    this.fragment_list_graph = new ArrayList<TopGraph>();
}

@Override
public Fragment getItem(int position) {

    if (this.orientation == Configuration.ORIENTATION_LANDSCAPE) {

        return getTopGraphFragment(position);

    } else {

        return getTopDataFragment(position);

    }
}

/**
 * Return the TopData fragment of the current position
 * If the fragment doesn't exist, it will be created.
 *
 * @param position Position of the fragment we must return
 */
Fragment getTopDataFragment(int position) {

    // Boolean to set the fragment at current position
    boolean atPosition = false;

    if (fragment_list.size() > position) {

        // If the item isn't null
        if (fragment_list.get(position) != null) {

            fragment_list.get(position).setSource(source);

            return fragment_list.get(position);
        }
```

```
            // The item is null, so we must instantiate it in the correct
                position
            atPosition = true;

        } else if (!(fragment_list.size() == position)) {

            // Check it size + 1 == to position
            int i = 0;

            // Fill the array with null
            while (i < position) {
                fragment_list.add(null);
                i++;
            }

        }

        // Initiate the fragment
        TopData frag = new TopData();

        Bundle options = new Bundle();
        options.putInt(FragmentArguments.ARG_PRODUCT, product);
        options.putInt(FragmentArguments.ARG_SOURCE, source);
        options.putInt(FragmentArguments.ARG_POSITION, position);

        frag.setArguments(options);

        // If we must to set it in the current position
        if (atPosition) {
            fragment_list.remove(position);
            fragment_list.add(position, frag);
        } else {
            // Add at the final of the array
            fragment_list.add(frag);
        }

        return frag;

    }

    /**
     * Stop the current pullToRefresh
     * */
    public void stopPullToRefresh(){

        // If the orientation is PORTRAIT
        if (this.orientation == Configuration.ORIENTATION_PORTRAIT){

            for(TopData item : fragment_list){
                // Check if the item is initiated
                if(item != null && item.isVisible()) {
                    item.cancelPullToRefresh();
                }
            }

        }
```

```
}

/**
 * Return the TopGraph fragment of the current position
 * If the fragment doesn't exist, it will be created.
 *
 * @param position Position of the fragment we must return
 */
Fragment getTopGraphFragment(int position) {

    // Boolean to set the fragment at current position
    boolean atPosition = false;

    // Check if we have items in the array
    if (fragment_list_graph.size() > position) {

        // If the item isn't null
        if (fragment_list_graph.get(position) != null) {
            return fragment_list_graph.get(position);
        }

        // The item is null, so we must instantiate it in the correct
            position
        atPosition = true;

    } else if (!(fragment_list_graph.size() == position)) {

        // Check it size + 1 == to position
        int i = 0;

        // Fill the array with null
        while (i < position) {
            fragment_list_graph.add(null);
            i++;
        }

    }

    // Create a new Frag
    TopGraph frag = new TopGraph();

    Bundle options = new Bundle();
    options.putInt(FragmentArguments.ARG_POSITION, position);

    frag.setArguments(options);

    if(product == Product.NETFLOW){
        frag.setScale(new ScaleBits());
    }

    // If we must to set it in the current position
    if (atPosition) {
        fragment_list_graph.remove(position);
        fragment_list_graph.add(position, frag);
    } else {
        // Add at the final of the array
```

```
            fragment_list_graph.add(frag);
        }

        return frag;

    }

    /**
     * When the user swipe to other window time,
     * we must refresh the current, next and last window
     *
     * @param context Context of the application FIXME is needed?
     * @param pos position of the current fragment
     * @param source new Source
     */
    public void updatePages(Context context, int pos, int source) {

        this.source = source;

        // FIXME only +-1

        // Update the fragments
        if (this.orientation == Configuration.ORIENTATION_LANDSCAPE) {
            for(TopGraph item : fragment_list_graph){
                // Check if the item is initiated
                if(item != null) {
                    item.onSourceChange(context, source);
                }
            }
        } else {
            for(TopData item : fragment_list){
                // Check if the item is initiated
                if(item != null) {
                    item.onSourceChange(context, source);
                }
            }
        }


    }

    /**
     * Change the current orientation and delete the references of the
         fragments
     * */
    public void setOrientation(int orientation, int position) {

        this.orientation = orientation;

        // If new configuration is Landscape, we must destroy the reference of
            TopDataFragment
        if (this.orientation == Configuration.ORIENTATION_LANDSCAPE) {

            // Remove the fragments
            mFragmentManager.beginTransaction().remove(getTopDataFragment(
                position)).commit();
```

```
            if (position == 0) {
                mFragmentManager.beginTransaction().remove(getTopDataFragment(
                    position + 1)).commit();
            } else if (position == 6) {
                mFragmentManager.beginTransaction().remove(getTopDataFragment(
                    position - 1)).commit();
            } else {
                mFragmentManager.beginTransaction().remove(getTopDataFragment(
                    position + 1)).commit();
                mFragmentManager.beginTransaction().remove(getTopDataFragment(
                    position - 1)).commit();
            }

        } else {

            // We must destroy the reference of TopGraphFragment
            // Remove the fragments
            mFragmentManager.beginTransaction().remove(getTopGraphFragment(
                position)).commit();

            if (position == 0) {
                mFragmentManager.beginTransaction().remove(getTopGraphFragment(
                    position + 1)).commit();
            } else if (position == 6) {
                mFragmentManager.beginTransaction().remove(getTopGraphFragment(
                    position - 1)).commit();
            } else {
                mFragmentManager.beginTransaction().remove(getTopGraphFragment(
                    position + 1)).commit();
                mFragmentManager.beginTransaction().remove(getTopGraphFragment(
                    position - 1)).commit();
            }

        }

        notifyDataSetChanged();

    }

    @Override
    public int getItemPosition(Object object) {
        return POSITION_NONE;
    }

    /**
     * Set the events in the current TopData
     */
    public void setEvents(List<TopEvent> events, boolean update, boolean
        position, int itemPosition) {

        ((TopData) getItem(itemPosition)).setEvents(events, update, position);

    }

    /**
     * Set the graph items in the fragment
     *
```

```
     * @param series Response from the server with the server
     * @param position Position of the fragment
     * @param filters Filters in this position. They array can be empty
     * */
    public void setGraphItems(GraphMultipleCollection series, int position,
        List<DataCollectorTop.filterTop> filters) {

        ((TopGraph) getItem(position)).drawGraph(series, position, filters);
    }

    /**
     * Return the number of pages
     */
    @Override
    public int getCount() {
        // Show 7 total pages.
        return NUM_ITEMS;
    }

    /**
     * Return the title of the page
     *
     * @param position Actual page position
     * @return The title of the page
     */
    @Override
    public String getPageTitle(int position) {
        Locale l = Locale.getDefault();
        switch (position) {
            case 0:
                return context.getString(R.string.title_section1).toUpperCase(l);
            case 1:
                return context.getString(R.string.title_section2).toUpperCase(l);
            case 2:
                return context.getString(R.string.title_section3).toUpperCase(l);
            case 3:
                return context.getString(R.string.title_section4).toUpperCase(l);
            case 4:
                return context.getString(R.string.title_section5).toUpperCase(l);
            case 5:
                return context.getString(R.string.title_section6).toUpperCase(l);
            case 6:
                return context.getString(R.string.title_section7).toUpperCase(l);
        }
        return null;
    }
}
```

Apéndice K
Clase para la gestión de alarmas

Código K.1 Contenido del fichero notifications/Alarms.java.

```
package net.redborder.redborder.notifications;

import com.activeandroid.query.Select;

import net.redborder.redborder.models.Alarm;

import java.util.List;

/**
 * Class to work with Alarms
 */
public class Alarms {

    // List of alarms
    List<Alarm> alarms;

    public Alarms(){
        // Init the alarms
        alarms = new Select().from(Alarm.class).orderBy("id DESC").execute();
    }

    /**
     * Get not view alarms
     * */
    public List<Alarm> getNotViewAlarms(){
        return new Select().from(Alarm.class).where("View = ?", 0).execute();
    }

    /**
     * Get the size of number of not view alarms
     * */
    public int getNumberNotViewAlarms(){
        return getNotViewAlarms().size();
    }

    /**
     * Get the size of number of not view alarms of a server
     * */
```

```
public int getNumberNotViewServerAlarms(String server_key){
    return getNotViewServerAlarms(server_key).size();
}

/**
 * Set all alarms on view
 * */
public void setViewAlarms(){

    for(Alarm alarm : getNotViewAlarms()){
        alarm.view = 1;
        alarm.save();
    }
}

/**
 * Get all alarms
 * */
public List<Alarm> getAllAlarms(){
    return alarms;
}

public static void setViewAllAlarms(){

    List<Alarm> alarms = new Select().from(Alarm.class).where("View = ?", 0)
        .execute();

    for(Alarm alarm : alarms){
        alarm.view = 1;
        alarm.save();
    }
}

/**
 * Get the alarms of the current server
 *
 * @param server_key The server key
 * @return the list of alarms
 * */
public List<Alarm> getServerAlarms(String server_key){

    if(server_key == null || server_key.isEmpty()){
        return new Select().from(Alarm.class).orderBy("id DESC").where("
            Server_key = ?", "").execute();
    } else {
        return new Select().from(Alarm.class).orderBy("id DESC").where("
            Server_key = ?", server_key).execute();
    }
}

/**
 * Get the not view alarms of the server
 *
 * @param server_key The server key
 * @return the list of alarms
 * */
public List<Alarm> getNotViewServerAlarms(String server_key){
```

```
        if(server_key == null || server_key.isEmpty()) {
            return new Select().from(Alarm.class).orderBy("id DESC").where("
                Server_key = ?", "").where("View = ?", 0).execute();
        } else {
            return new Select().from(Alarm.class).orderBy("id DESC").where("
                Server_key = ?", server_key).where("View = ?", 0).execute();
        }
    }

    /**
     * Set viewed the alarms of the server
     *
     * @param server_key The server key
     * */
    public void setViewServerAlarms(String server_key){
        // Each alarm...
        for(Alarm alarm : getServerAlarms(server_key)){
            alarm.view = 1;
            alarm.save();
        }
    }
}
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Varios autores, *Activities in android developer reference*, *http://developer.android.com/reference/android/app/Activity.html*, Septiembre 2008.

[2] _____, *Android developer reference*, *http://developer.android.com/develop/index.html*, Septiembre 2008.

[3] _____, *Making listview scrolling smooth*, *http://developer.android.com/training/improving-layouts/smooth-scrolling.html*, Septiembre 2008.

[4] _____, *Manifest permissions in android developer reference*, *http://developer.android.com/reference/android/Manifest.permission.html*, Septiembre 2008.

[5] Daniel Nadeau, *Holograph library documentation*, *https://bitbucket.org/danielnadeau/holographlibrary/wiki/Home*.

[6] Javier Pacheco, *Restrung documentation*, *https://github.com/47deg/appsly-android-rest/tree/restrung-1.1*, 2013.

[7] Michael Pardo, *Activeandroid documentation*, *http://www.activeandroid.com*, 2013.

Índice alfabético

Glosario

EDI Entorno de desarrollo integrado. 29

FCAPS Fault, Configuration, Accounting, Performance and Security. 1

GCM Google Cloud Messaging. 38

HTTP Hypertext Transfer Protocol. 7

OLAP On-Line Analytical Processing. 4

ORM Object-Relational mapping. 32

REST Representational State Transfer. 7

SDK Software Development Kit. 29