MEMORIA DE ARQUITECTURA

LOGROLLING

Alberto Almagro
Rubén Gómez
Juan Carlos Llamas
Jaime Martínez
Santiago Mourenza
Pedro Palacios
Adrián Sanjuán
Pablo Torre

Índice

1. Introducción y estructura de la vista

(PARA LA COMPRENSIÓN DE ESTE DOCUMENTO ES NECESARIO HABER LEIDO LA MEMORIA Y ARQUITECTURA DEL SERVIDOR)

Pasamos ahora a analizar la estructura del cliente, que es una aplicación Android. Para poder comprender dicha estructura, es necesario tener un conocimiento básico sobre la API de Android.

Cada *pantalla* o *ventana* de una aplicación Android es una clase diferente. Dichas clases se llaman *Activities* y heredan de la clase **AppCompatActivity** definida por el sistema operativo Android. Así mismo, cada uno de los componentes gráficos (botones, texto, imágenes) tienen su propia clase, que hereda de la clase general **View**.

Otro concepto muy importante en Android es el de **permiso**. Las aplicaciones no pueden acceder a todos los recursos que quieran, tienen que pedirle en **tiempo de ejecución** permiso al usuario, que puede darlo o denegarlo. Además, existen más operaciones **asíncronas**, como las peticiones HTTP. Por tanto, no podemos trabajar con código secuencial, sino que tenemos que trabajar con **callbacks**. Los callbacks son funciones que pasamos como argumento a una función, y que ésta ejecutará cuando termine su tarea asíncrona. Normalmente, las funciones toman dos callbacks. Uno que llamará en caso de error, y otro que llamará en caso de que todo haya salido correctamente.

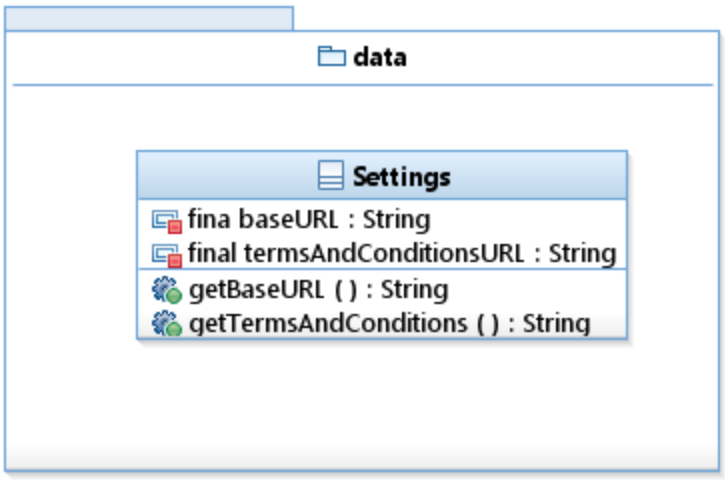
Además, igual que no implementamos desde 0 en el servidor la plataforma de pagos, también utilizaremos en Android ciertas APIs para poder tener acceso a ciertas características, como peticiones HTTP, localización, anuncios o pagos.

Con estos conocimientos, podemos pasar a analizar la arquitectura del cliente:

```
com.logrolling.client
├── data
│   └── Settings
├── exceptions
├── web
├── transfer
├── delegates
├── controllers
├── services
├── view
└── adapters
```

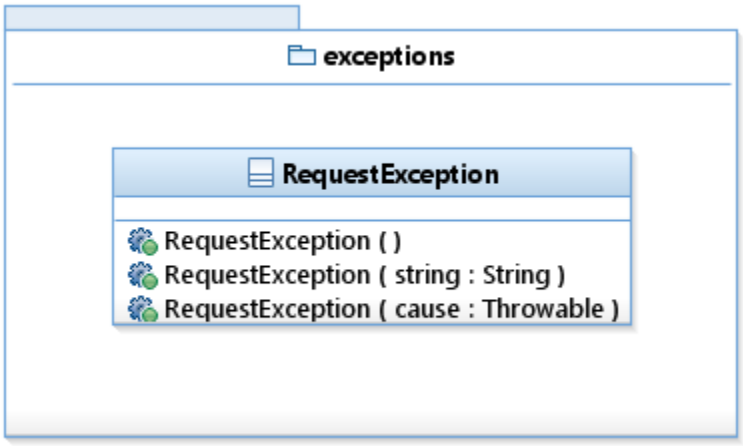
2. Paquete data

Solo contiene la clase Settings. La única función de esta clase es almacenar datos estáticos, como la URL del servidor a la que hacer peticiones HTTP.



3. Paquete exceptions

Este paquete contiene las excepciones del cliente. De momento solo contiene la excepción RequestException que se lanza cuando hay algún error de red.



4. Paquete web

Este paquete se encarga de definir abstracciones a bajo nivel para peticiones HTTP. Es decir, ofrece formas de mandar peticiones GET, POST, PUT o DELETE a cierta URL dada.

Para poder enviar peticiones HTTP, utilizamos la librería de Google *Volley*, que permite acceso de bajo nivel a las comunicaciones de red. Para comprender el funcionamiento de este paquete, debemos comprender primero las básicas de Volley.

Volley consiste principalmente en dos componentes:

- **Peticiones web:** Volley permite definir nuestras propias peticiones HTTP, que luego enviará a la URL que contenga la petición. Sin embargo, es posible que haya varias peticiones. Como Volley no puede enviar varias peticiones a la vez, hay que crear una *cola de peticiones*. Para enviar una petición, basta con añadirla a dicha cola, y Volley se encargará de enviarla cuando pueda.
- **Carga de imágenes:** Como veremos más adelante en la vista, Volley soporta la carga de imágenes. Además, para evitar volver a cargar imágenes y gastar muchos datos, se puede configurar una caché para las imágenes. Esa caché también tenemos que configurarla nosotros.

Con estos conocimientos básicos, podemos analizar la estructura del paquete:

```
web
├── WebRequestQueue
├── WebServiceClient
├── ErrorListener
├── ResponseListener
└── SuccessListener
```

4.1 WebRequestQueue

La clase WebRequestQueue es la encargada de crear una cola de peticiones, a la que más adelante podemos añadir peticiones para enviar, así como de configurar la caché que Volley utilizará para Volley. Aquí observamos un patrón que se repetirá mucho a lo largo de la arquitectura de Android. Es una especie de Singleton, salvo que existe un método aparte, *createInstance* que recibe más parámetros que *getInstance*. Esto es porque en muchos casos, es necesario inicializar estas clases desde una *Activity*, para pasarle el contexto. Sin embargo, en muchas otras ocasiones, no disponemos de este contexto y queremos utilizar igualmente la clase, por lo que existe esa asimetría entre la creación y el uso, pero sin embargo, seguimos queriendo tener los beneficios de un Singleton.

4.2 WebServiceClient

La clase WebServiceClient es la encargada de realizar una petición a cierta URL (que concatena a la URL base definida en la clase Settings), método y datos de entrada. Acepta dos callbacks, uno que llamará en caso de éxito con la información devuelta en forma de String, y otro que llamará en caso de error.

4.3 ErrorListener

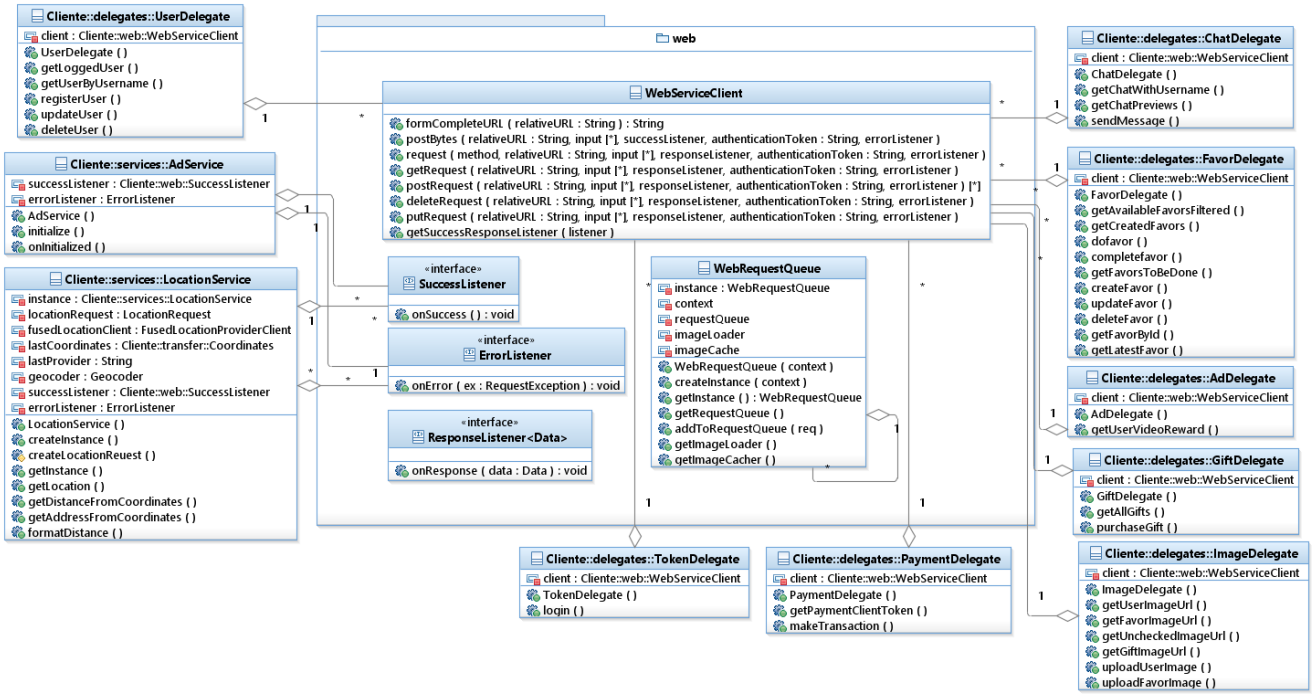
La interfaz ErrorListener define un callback que será llamado en caso de error. Para utilizarla, puede crearse una clase anónima que herede de ErrorListener, o, la opción preferida, utilizar una función lambda.

4.4 ResponseListener

La interfaz ResponseListener<T> define un callback que será llamado con un parámetro de tipo T en caso de éxito.

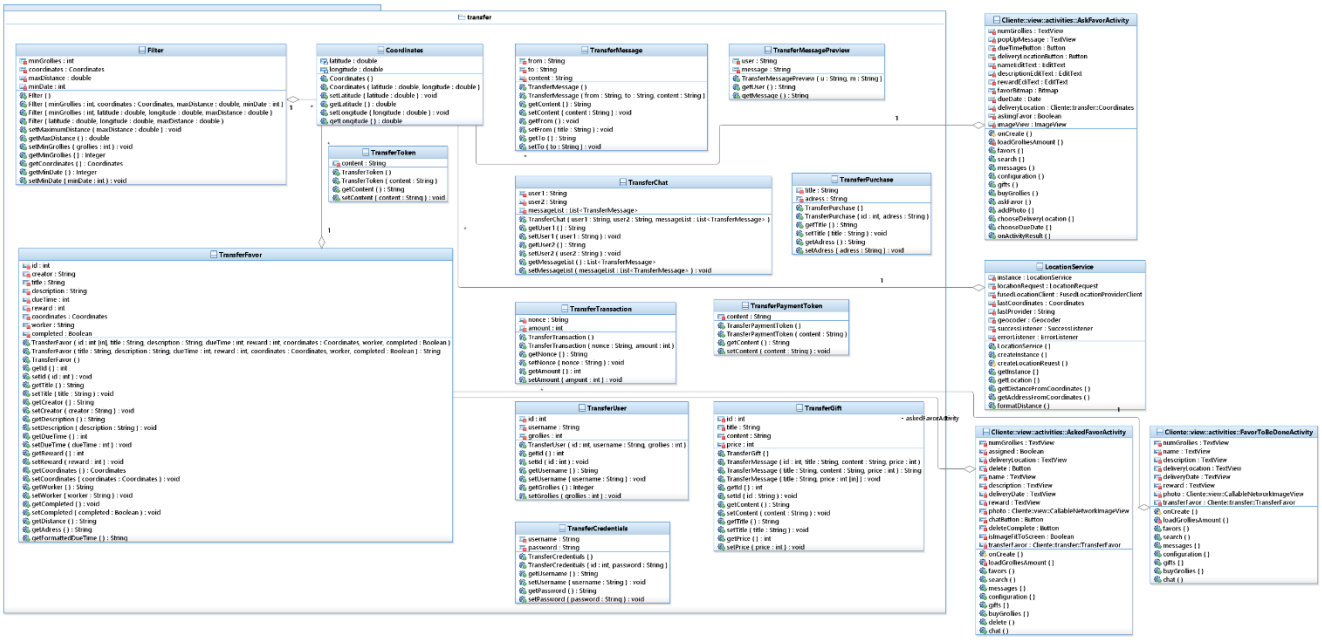
4.5 SuccessListener

La interfaz SuccessListener define un callback sin parámetros que será llamado en caso de éxito.



5. Paquete transfer

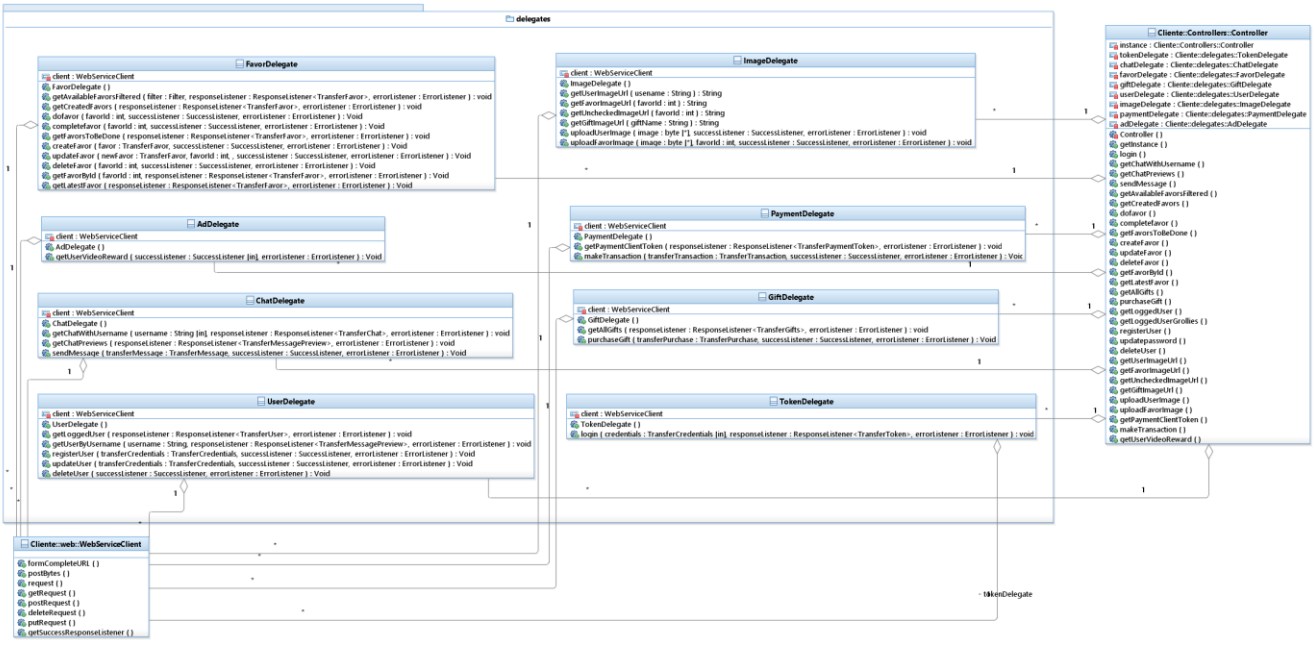
Hemos hablado de enviar datos de entrada y salida a través de peticiones HTTP. Estos datos son codificados en java como objetos transfer. Estos objetos transfer consisten en solo *getters* y *setters*, y tienen los mismos atributos que sus homólogos en el servidor. Estos objetos transfer serán serializados y deserializados hacia y desde JSON, como veremos más adelante.



6. Paquete delegates

Hemos comentado antes como el paquete **web** constituía una abstracción a bajo nivel de las peticiones HTTP. El paquete **delegates** constituye una abstracción a alto nivel de las peticiones HTTP al servidor. Existe un delegado por cada uno de los servicios que tiene el servidor. Podría verse como una contrapartida a cada *WebServiceBroker*.

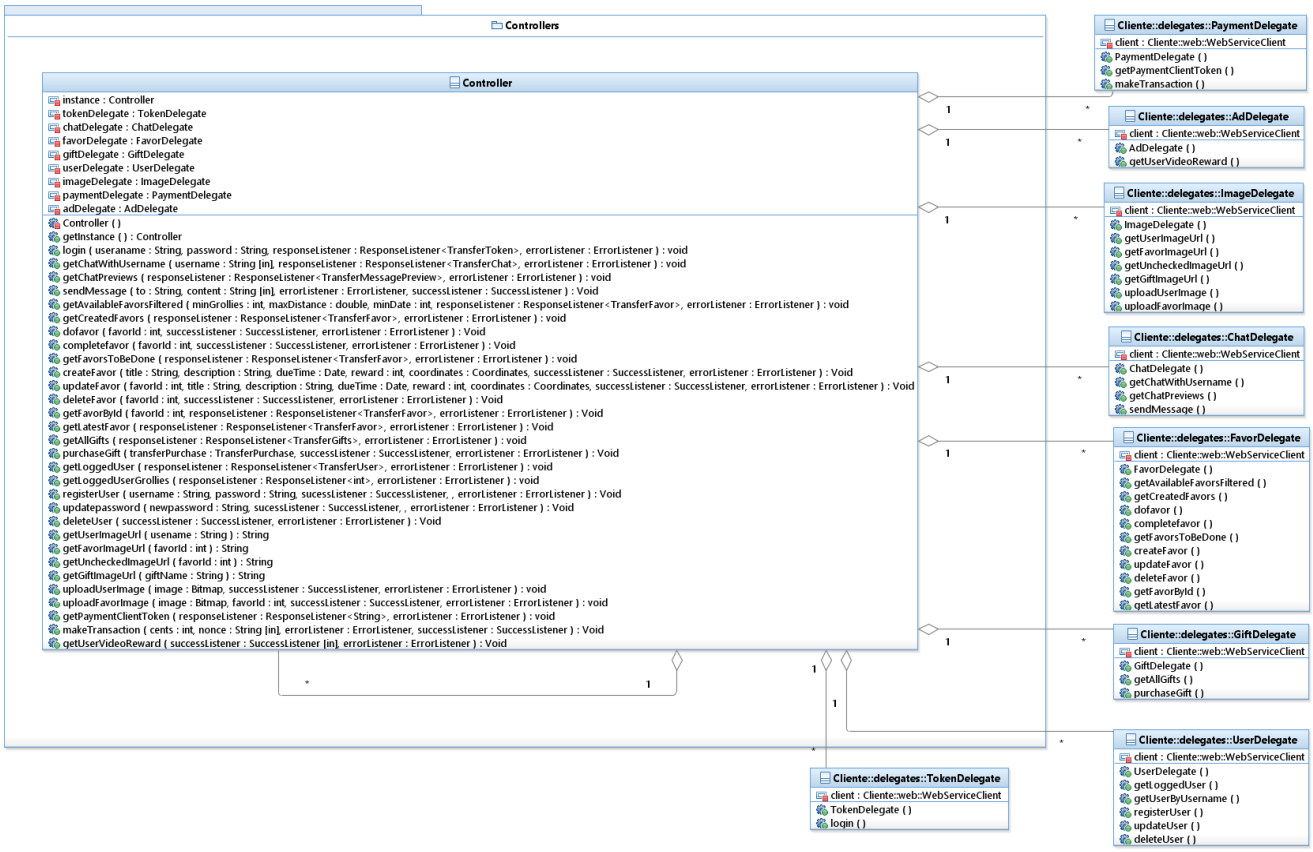
Un ejemplo es la clase **UserDelegate**, que tiene métodos como *registerUser*, que toma un **TransferCredentials**, un delegado de éxito y un delegado de error. Por tanto, es una abstracción a alto nivel.



7. Paquete controllers

Tiene una única clase, **Controller**, que es una abstracción de más alto nivel que los delegates. Esta clase es la que usa la vista para comunicarse con el servidor. Hemos creado otra capa de indirección en el caso de que en un futuro queramos tener distintos *delegates* mediante distintos métodos de comunicación (por ejemplo, WebSockets para los mensajes). De esta manera, solo es necesario añadir un nuevo delegado, sin tener que modificar todo el código.

Otra abstracción que añade **Controller** es que oculta el uso de los transfer, que convierte en parámetros de entrada a la vista. Un ejemplo sería el método *registerUser,* que además de los delegados, en vez de tomar un **TransferCredentials**, toma dos **Strings**.



8. Paquete services

La aplicación Android tiene distintos *servicios* independientes, como por ejemplo, localización, almacenamiento persistente… Cada uno de estos servicios tiene su propia clase, algunas son singletons y otras, las que necesitan contexto para su creación, siguen el mismo modelo que el ya citado en WebRequestQueue.

Los servicios por tanto, son en su mayor parte independientes entre sí, y llevan a cabo una acción muy concreta.

La estructura del paquete es la siguiente:

services
- AdService
- SerializationService
- LocationService
- PersistentStorageService
- AuthenticationService

Como los servicios son mayormente independientes entre sí, los analizaremos uno a uno.

8.1 Anuncios

Para mostrar anuncios, utilizamos el SDK para Android de Facebook Ads. Por tanto, AdService se encarga de inicializar el SDK de los anuncios de Facebook, tomando un callback que llama cuando ha sido inicializado correctamente el SDK.

El usuario puede elegir ver un vídeo de 30 segundos a cambio de una recompensa en *grollies*.

8.2 Serialización

Al contrario que en JAX-RS, Volley no convierte automáticamente entre objetos y JSON. Para ello utilizamos otra API de Google, *Gson*. SerializationService se encarga de deserealizar las respuestas recibidas en String de Volley a el objeto que se le pasa. Como el resto de servicios, toma un callback de error que llama en caso de algún fallo. Esto ayuda a su integración con clases del paquete **web**.

8.3 Localización

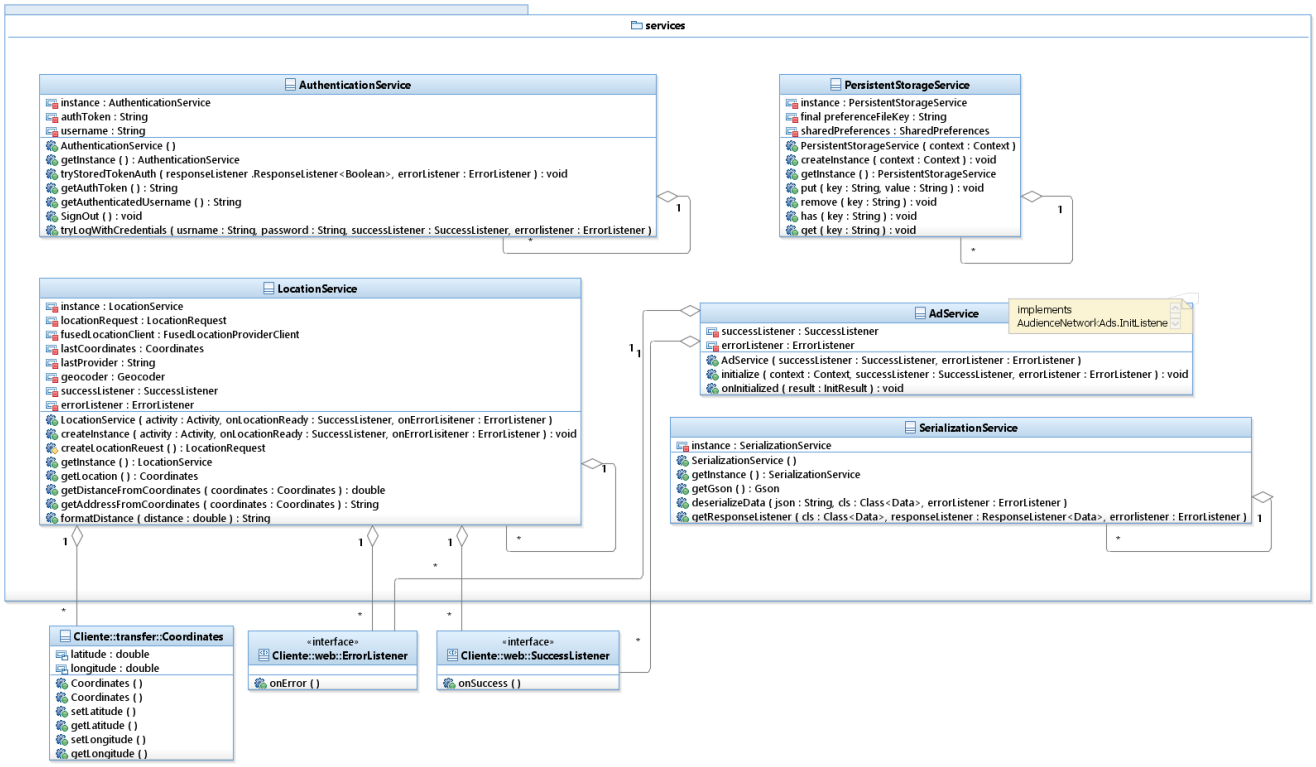
Para obtener la localización del usuario, utilizamos el servicio Google Play Maps. LocationService se encarga de manejar este servicio. Además, aparte de para obtener la localización, utilizamos este servicio para mostrar un mapa al usuario para elegir la localización del favor, y para presentar de manera bonita las coordenadas como una dirección, utilizando el servicio de google para obtener dirección en texto a partir de coordenadas.

8.4 Persistencia

Ninguna de la información de las clases de Android sobrevive entre distintas instancias de la aplicación. Para poder almacenar información entre distintas ejecuciones, necesitamos un servicio de persistencia. PersistentStorageService se encarga de almacenar persistentemente un mapa que tiene como claves Strings y como valores Strings.

8.5 Autenticación

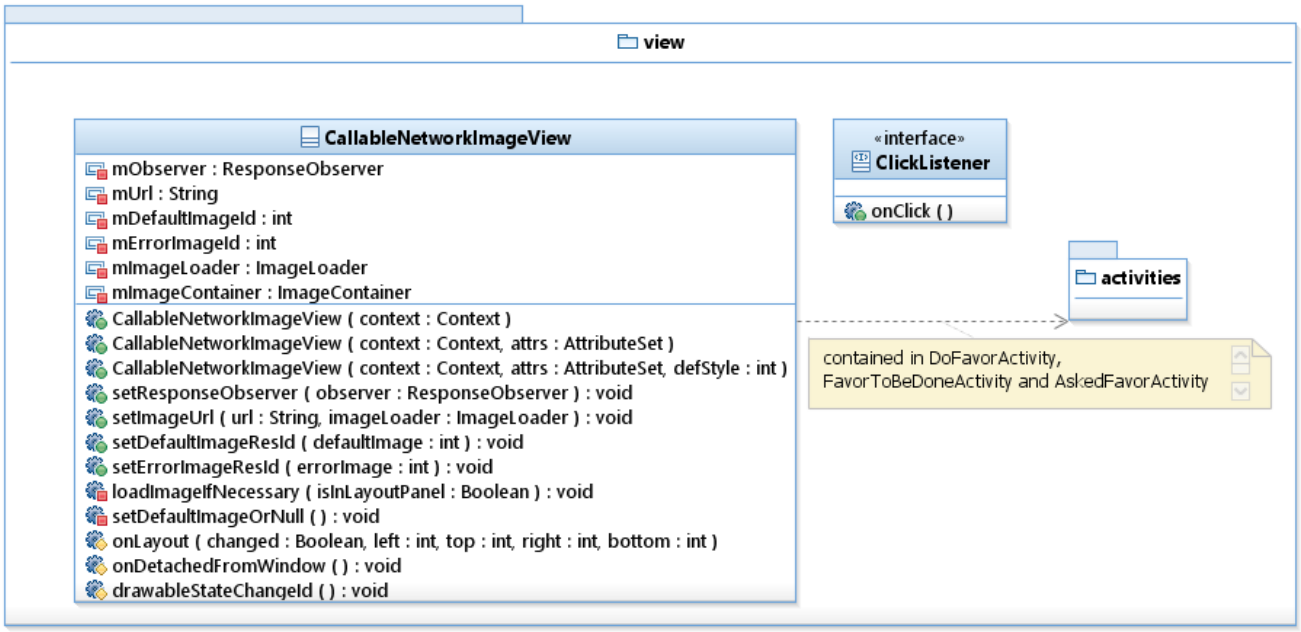
AuthenticationService se encarga de lo relacionado con la autenticación. Utiliza PersistentStorageService para almacenar tokens de autenticación persistentemente, y de esta manera poder *recordar* la sesión del usuario.



9. Paquete view

En este paquete está todo el código encargado de mostrar al usuario todo lo relacionado con lo visual. En el patrón MVC, este paquete sería la vista. La estructura de este paquete es:

services
- ClickListener
- CallableNetworkImageView
- activities



9.1 ClickListener

La clase ClickListener define un callback que se ejecutará cuando se haga click en alguna View. Lo usamos para añadir mayor interactividad con el usuario.

9.2 CallableNetworkImageView

Como indicamos antes, Volley tiene soporte de serie para cargar imágenes. Sin embargo, la clase que trae por defecto, NetworkImageView, no tiene soporte para callbacks en caso de error, y solo permite mostrar una imagen por defecto en caso de error. Como esto no era suficiente para nuestras necesidades, hemos modificado el código de NetworkImageView, creando CallableNetworkImageView, que llama a un callback en caso de error al cargar la imagen.

9.3 Activities

- MainActivity:

  Es la primera actividad que se lanza al ejecutar la aplicación. No tiene funcionalidad a alto nivel. Se encarga de inicializar todos los servicios y de pasar a la pantalla de Inicio de sesión o a la pantalla principal si ya se había iniciado sesión anteriormente.

- **Pantalla de Inicio de sesión (SignInActivity):**

  El usuario puede intentar iniciar sesión o registrarse. Si intenta iniciar sesión, debe estar ya registrado e introducir su nombre de usuario y contraseña. En caso de éxito, se le dirigirá a la pantalla *Explorar*. Si esta acción fracasa, se le mantendrá en la pantalla de Inicio de sesión y se le mostrará una notificación con el error. Si pulsa en el botón *REGÍSTRATE*, se le dirigirá a la pantalla de Registro.

- **Pantalla de Registro (RegistrationActivity):**

  En esta pantalla el usuario puede o bien registrarse, o bien volver a la pantalla de Inicio de sesión pulsando en el botón Iniciar sesión. Si decide registrarse como un nuevo usuario, deberá introducir los datos pertinentes en los campos *Nombre de usuario*, *Contraseña* y *Repetir contraseña*. Además, deberá aceptar los Términos y Condiciones y la Política de Privacidad. Si pulsa en Términos y Condiciones y Política de Privacidad se le redireccionará a una página web donde se encuentran ambos documentos. Si el usuario pulsa en el botón *REGÍSTRATE* una vez haya cumplimentado esta información, se procederá al registro. Si la operación tiene éxito, se le notificará al usuario mediante un popUp con la posibilidad de ir a la pantalla de inicio de sesión, y si el registro no ha tenido éxito, se le notificará el motivo.

- **Pantalla Explorar (SearchActivity):**

  Esta es una de las pantallas principales de la aplicación. Al acceder a ella, el usuario puede ver una lista de favores pedidos por otros usuarios. Las principales funcionalidades de esta pantalla son el acceso a un favor y el filtrado de favores. Pulsando en un favor el usuario es dirigido a la pantalla *Realizar favor*, que se cargará con la información relativa al favor seleccionado. Por otro lado, al pulsar el botón de la esquina superior derecha, se despliega un menú de filtrado de favores que permitirá al usuario filtrar los favores que aparecen en la lista según tres criterios. Estos son: el tiempo que queda para que caduque un favor (diferencia entre fecha límite y fecha actual), la recompensa mínima de *grollies* por realizar el favor y la distancia máxima a la que se encuentra el lugar de entrega. Si el usuario pulsa en el botón *BUSCAR*, se le ofrecerá la lista de favores según el criterio que ha escogido o se le informará de que no hay ningún favor que verifique las condiciones impuestas. Además, esta pantalla tiene un menú inferior de navegación y un panel en la esquina superior izquierda de acceso a la pantalla de la Tienda. Con el menú de navegación inferior se puede acceder a las pantallas *Explorar, Favores, Mensajes, Regalos o Ajustes*. Está es una característica común de todas las pantallas que siguen a esta. El panel en

la esquina superior izquierda muestra los *grollies* que tiene el usuario y si se pulsa en el interior de dicho panel el usuario accede a la pantalla *Tienda*. Salvo que se indique lo contrario, de esta pantalla en adelante se presupondrá que todas las pantallas tienen dicho panel de grollies con esta misma funcionalidad.

- **Pantalla Favores (MyFavorActivity):**

  En esta pantalla el usuario puede visualizar la pestaña *Favores pedidos* o de *Favores por realizar*. Cada una mostrará la lista de los favores pedidos por el usuario, o bien la lista de favores a realizar por el usuario. Por defecto se le mostrará primero la lista de favores pedidos, pero podrá cambiar de una a otra pulsando en los botones de la parte superior. Si se está mostrando la pestaña con la lista de favores pedidos, al pulsar en un cualquiera, se le conducirá a la pantalla *Favor pedido*, que se cargará con la información relativa al favor pedido seleccionado. Si se está mostrando la pestaña con la lista de favores a realizar, al pulsar un cualquiera de los favores, se le conducirá a la pantalla Favor a realizar, que se cargará con la información relativa al favor a realizar seleccionado. Desde cualquiera de las dos pestañas, el usuario puede pulsar en el botón *PEDIR NUEVO FAVOR* que le conducirá a la pantalla *Pedir nuevo favor*.

- **Pantalla Mensajes (MessageActivity):**

  En esta pantalla al usuario se le mostrarán los chats que tiene abiertos con otros usuarios de la aplicación. Si pulsa en cualquiera de los chats, se le dirigirá a la pantalla *Chat*, que se cargará en función de la conversación que el usuario haya seleccionado. Para cada chat de la lista, el usuario puede ver el último mensaje que se envió en esa conversación y la foto de perfil del otro usuario, así como su nombre de usuario.

- **Pantalla Regalos (GiftsActivity):**

  En esta pantalla el usuario puede elegir canjear una cantidad de *grollies* por uno de los regalos que se muestran. De cada regalo se puede ver una foto, el nombre del regalo y el precio en *grollies* que cuesta el regalo. Cuando el usuario selecciona un regalo, si no tiene *grollies* suficientes, se le muestra un mensaje de error y se cancela la operación. Si tiene *grollies* suficientes, se le pide a usuario la dirección postal a la que enviar el regalo y, cuando da la confirmación, se le substraen los *grollies* correspondientes. El usuario también puede cancelar la operación.

- **Pantalla de Ajustes (ConfigurationActivity):**

  El usuario tiene la opción de acceder a su perfil o cerrar sesión. Si decide cerrar sesión se le pedirá confirmación (con la opción de cancelar la operación) y, si se ratifica, se cerrará su sesión y se le llevará a la pantalla de *Inicio de Sesión*. Si opta por acceder a su perfil se le mostrará la pantalla de *Mi Perfil*. Para futuras versiones, teníamos pensado incluir la posibilidad de cambiar de idioma, y esta opción estaría disponible en la pantalla de ajustes. Gracias al diseño que hemos realizado tenemos una gran internacionalización dado que Android Studio nos proporcionas útiles herramientas para tal propósito. Dentro de los archivos de la aplicación, contamos con un documento llamado strings.xml dentro de la carpeta de values en resources. En este documento aparecen lo que podríamos identificar como nombres de variables constantes a las que se le asocia una cadena de texto que no se modifica en ningún momento. El hecho de trabajar con esta especie de variables constantes globales permite que si queremos cambiar el idioma de la aplicación, solo tendremos que tener otro documento donde el valor de estos identificadores esté en otro idioma. Automáticamente, la interfaz se podrá mostrar en distintos idiomas según se elija el documento de strings correspondiente.

- **Pantalla de la Tienda (ShopActivity):**

  En esta pantalla el usuario puede adquirir un número determinado de *grollies* realizando una compra o de manera gratuita viendo un video. Si pulsa en el botón *GRATIS* se le mostrará un anuncio y, cuando este termine, se le sumarán 10 *grollies* a su total. Si pulsa en cualquiera de los botones de precio, se le pedirá una confirmación para realizar dicha compra y se le redireccionará al sistema de pagos de Braintree. Al finalizar la operación se le sumará el número de *grollies* que haya adquirido a su total. En cualquier momento el usuario puede cancelar la operación. Si hay algún error se le notificará mediante un popUp con la información del error.

- **Pantalla Mi Perfil (MyProfileActivity):**

  En esta pantalla el usuario puede ver su foto de perfil, así como su nombre de usuario. Si el usuario pulsa en la foto, se le da la opción de cambiar la foto de perfil por una que haga con la cámara del móvil u otra que ya tenga en la Galería. Cuando se produce el cambio, el usuario es notificado con un mensaje de éxito. En esta pantalla también se puede cambiar la contraseña. Para ello se debe introducir la nueva contraseña dos veces y pulsar sobre el botón *CAMBIAR CONTRASEÑA*. Si la operación tiene éxito, se le notifica al usuario

que su perfil ha sido actualizado y se le dirige a la pantalla de *Ajustes*. En caso contrario se le muestra el error. En esta pantalla no se puede cambiar el nombre de usuario.

- **Pantalla de Chat (UserChatActivity):**

  Esta pantalla representa los chats abiertos con otros usuarios. Los chats se crean si un usuario accede desde la pantalla de Favores → pestaña Favores pedidos → pantalla de Favor Pedido → Botón *CHATEAR CON REALIZADOR*, o bien desde la pantalla de Favores → Pestaña Favores a realizar → pantalla de Favor a realizar → Botón *CHATEAR*. Una vez se ha creado un chat, se puede acceder a la pantalla de dicho chat por el método anterior o dirigiéndose directamente a la pantalla de *Mensajes* y pulsando en la conversación pertinente. En la pantalla de *Chat* se muestra la conversación entre los dos usuarios. Estos pueden escribir mensajes en el cuadro inferior y enviarlos pulsando en el botón de la parte inferior derecha. También, se puede acceder a la pantalla de *Mensajes* pulsando en la flecha superior izquierda. Observación: esta pantalla no muestra el panel con el número de *grollies* que permite acceder a la tienda.

- **Pantalla Pedir nuevo favor (AskFavorActivity):**

  En esta pantalla aparecen una serie de campos que el usuario debe rellenar para poder pedir un favor. Estos campos son *Nombre del favor*, *Descripción del favor*, *Lugar de entrega*, *Fecha límite* y *Recompensa ofrecida*. Todos los campos tienen que estar completados y se tiene que cumplir que la fecha de entrega es dentro de al menos una hora y la recompensa en *grollies* tiene que ser mayor o igual que 10 *grollies*. Si no se satisfacen estas condiciones se le muestra un mensaje de error al usuario. Adicionalmente, el usuario puede añadir de manera opcional una foto del favor pulsando en el icono de la imagen. Cuando el usuario ha completado todos los campos con datos válidos, se le pide confirmación para publicar el favor. Si se da la confirmación, el favor queda publicado y se le substraen los *grollies* correspondientes al usuario de su total y se le dirige a la pantalla de Favores pedidos, donde ya aparecerá el nuevo favor dentro de la lista de *Favores pedidos*.

- **Pantalla Favor Pedido (AskedFavorActivity):**

  Se le muestra al usuario la información de un favor que ha pedido: *Nombre del favor*, *Descripción del favor*, *Foto del favor* (opcional), *Lugar de entrega*, *Fecha límite* y *Recompensa ofrecida*. Existen dos posibilidades: que este favor tenga

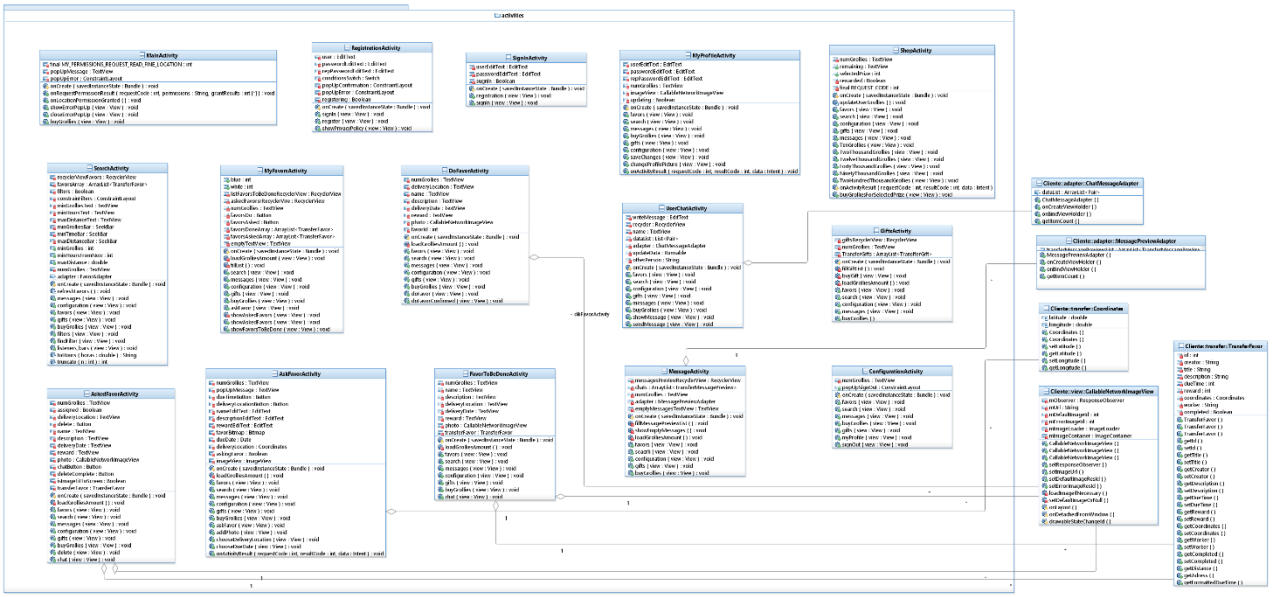
asignado un realizador, o bien que no lo tenga. Si no tiene un realizador asignado, se le ofrece al usuario la posibilidad de borrar el mensaje. Si este favor ya ha sido asignado a alguien, aparece en el campo de realizador la persona que se ha adjudicado dicho favor y dos botones que son *COMPLETAR* y *CHAT CON REALIZADOR*. Si alguien se ha adjudicado un favor, no se le da al usuario la posibilidad de borrarlo. El botón de *COMPLETAR* elimina el favor de la lista de Favores pedidos del usuario y de Favores a realizar del realizador y le transfiere al realizador la recompensa en grollies del favor. Se debe completar un favor cuando ya ha sido realizado. El botón de *CHAT CON REALIZADOR* sirve para acceder o crear un chat con el usuario que realiza el favor.

- **Pantalla Favor a realizar (FavorToBeDoneActivity):**

  Muestra al usuario que se ha adjudicado un favor la información de dicho favor. También permite acceder o crear un chat con el usuario que demanda el favor.

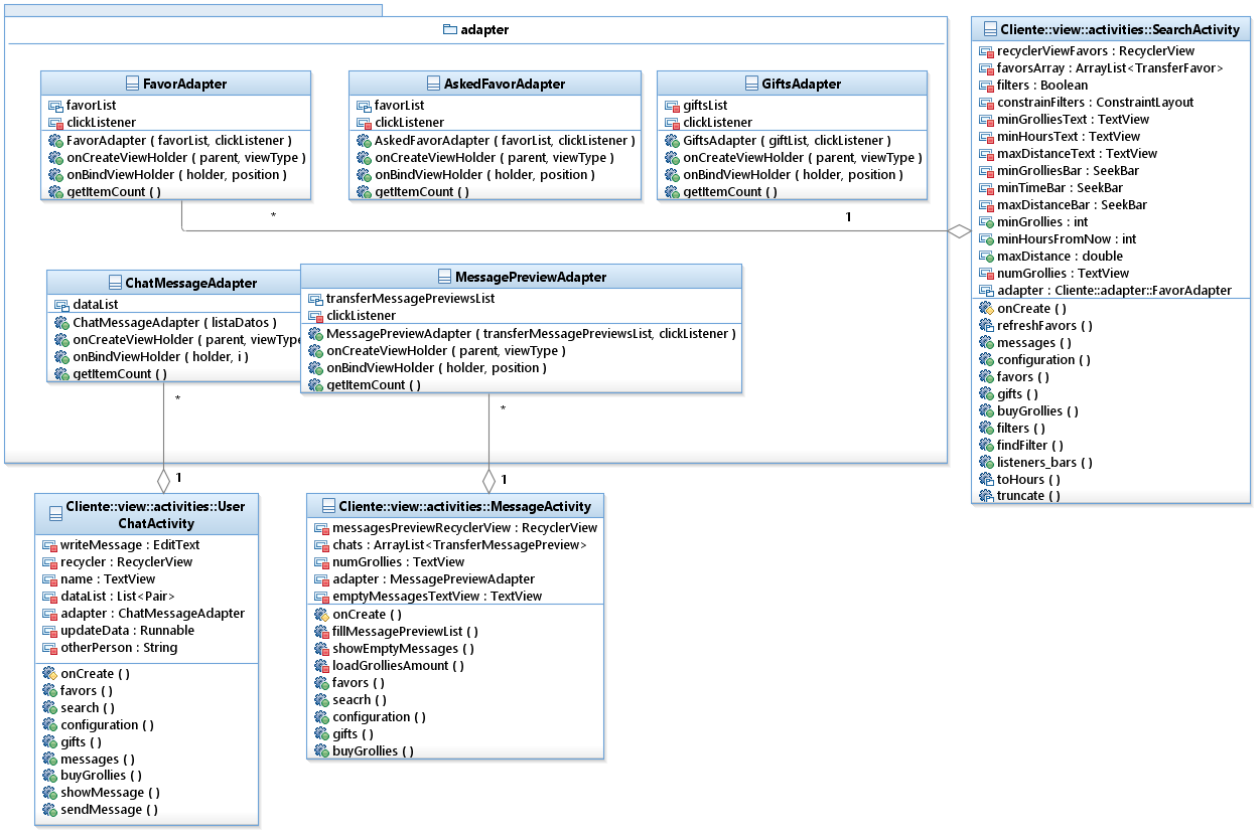
- **Pantalla Realizar favor (DoFavorActivity):**

  Muestra al usuario la información de un favor que otro usuario ha solicitado y nadie ha decidido realizar todavía y le permite adjudicarse el favor pulsando en el botón *REALIZAR FAVOR*. Si el usuario decide realizar el favor, se le pedirá confirmación y, si continúa, se añade el favor a la lista de *Favores a realizar* y se cambia el estado del favor (en la lista de favores pedidos del usuario que demanda el favor) asignado.



10. Paquete adapter

Para poder mostrar una cantidad arbitraria de elementos (como la lista de mensajes, o lista de favores), utilizamos la clase que trae Android llamada RecyclerView. Sin embargo, esta clase no tiene por defecto soporte para las clases creadas por el usuario (los transfer). Para ello, necesitamos crear un *Adapter* para cada uno de los datos que queramos mostrar en un RecyclerView. Estos *Adapter* le indican a Android como debe dibujarse cada uno de los elementos, y están todos en el paquete adapter.



11. Patrones utilizados

En el cliente de Logrolling hemos usado varios patrones de diseño de software estándar, pero en cada caso modificándolos ligeramente para adecuarlos a las necesidades concretas de nuestro proyecto.

11.1 Observer

El patrón Observer es ampliamente utilizado en la implementación de varias características de Android que utilizamos. Todos los *Listeners*, que nos permiten reaccionar a distintos eventos que ocurren en la aplicación (pulsación de botones, escritura en campos de texto, …), son registrados en una lista de observadores por Android. Cada vez que el usuario realiza alguna de esas acciones, el subsistema de Android llama a los observadores.

11.2 Singleton

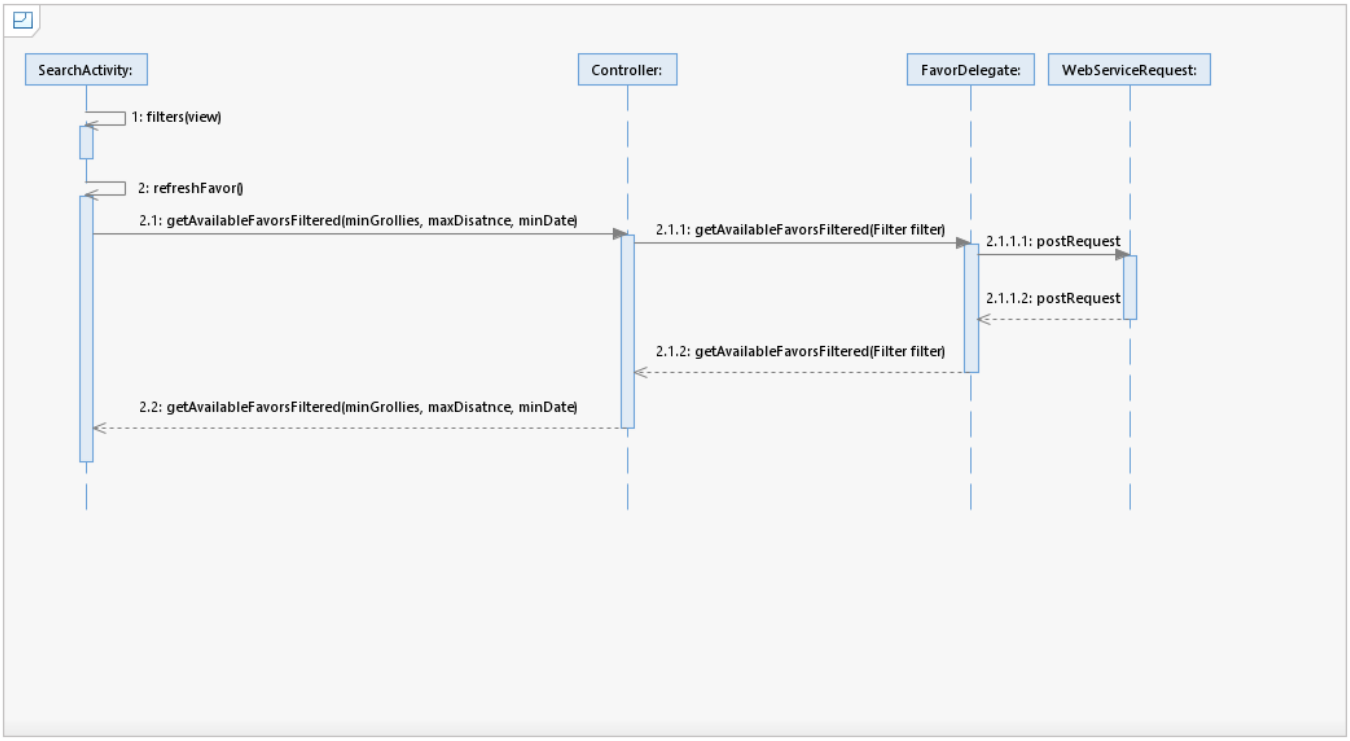
En Logrolling utilizamos el patrón Singleton en el caso de que queramos una única instancia y queramos mantener los beneficios del polimorfismo asociado a tener objetos dinámicos y no estáticos. Esta ventaja es clara en los distintos *Services* de la vista, ya que tal vez en un futuro añadamos más proveedores distintos de la información que solicita (distintos proveedores de anuncios o de servicios de localización) y querramos poder utilizar polimorfismo unido a una factoría abstracta, por lo que sería sencillo de modificar el código sin tener que modificar todo el código que utiliza las clases de *Services*. Sin embargo, en otros casos, como en ciertas partes del servidor, más beneficioso que el Singleton son las clases con métodos estáticos por motivos de concurrencia, distribución y rendimiento, así como la falta de necesidad de polimorfismo y vinculación dinámica en esos casos.

11.3 Adapter

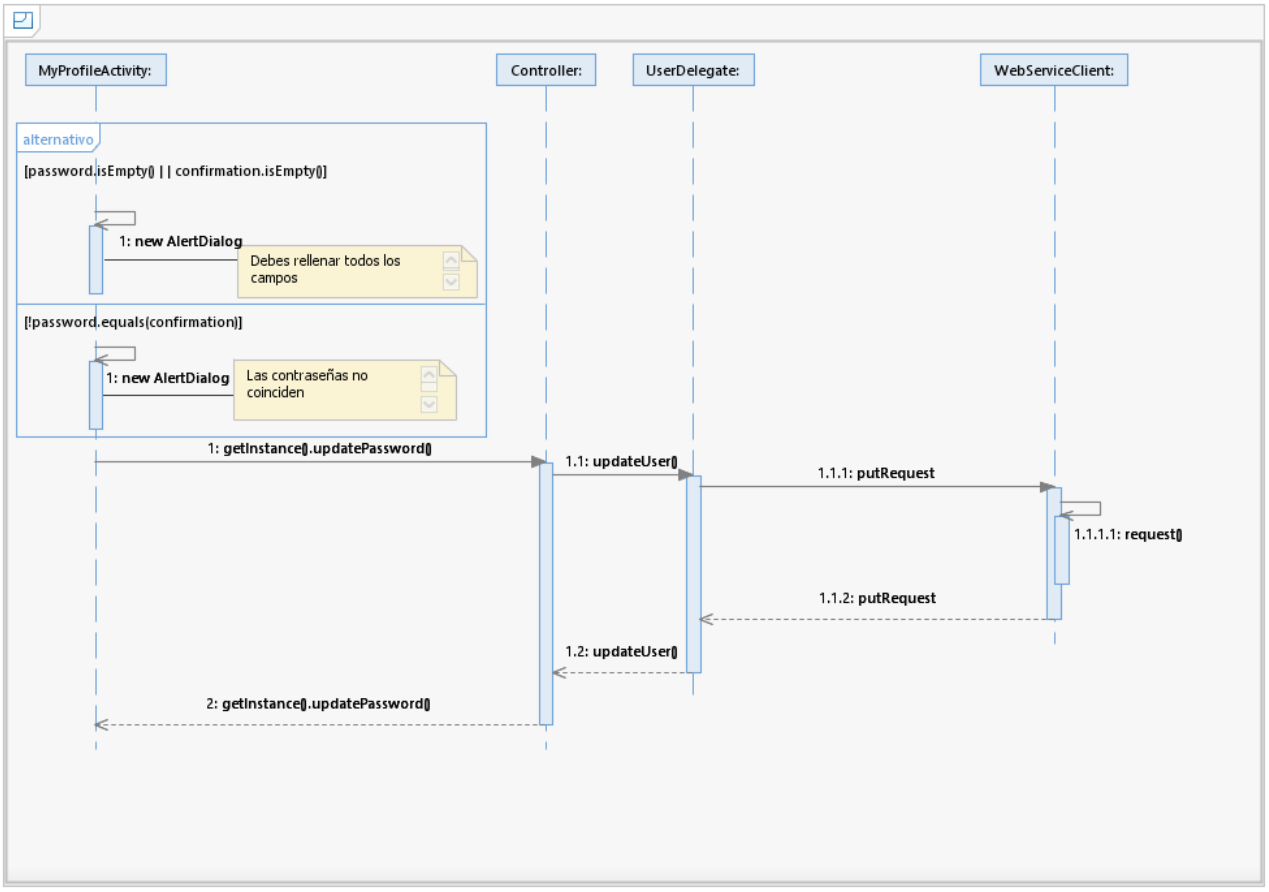
Utilizamos el patrón Adapter varias veces en la capa de la vista de Android. Para mostrar eficientemente una lista de muchos elementos a través de la que el usuario se pueda desplazar, utilizamos el componente de Android RecyclerView. Dicho RecyclerView, necesita un Adapter para cada lista de objetos definidos por el usuario, que definan cómo se van a renderizar dichos elementos. Por tanto, para cada RecyclerView que utilizamos, tenemos su respectivo Adapter.

12. Diagramas de secuencia

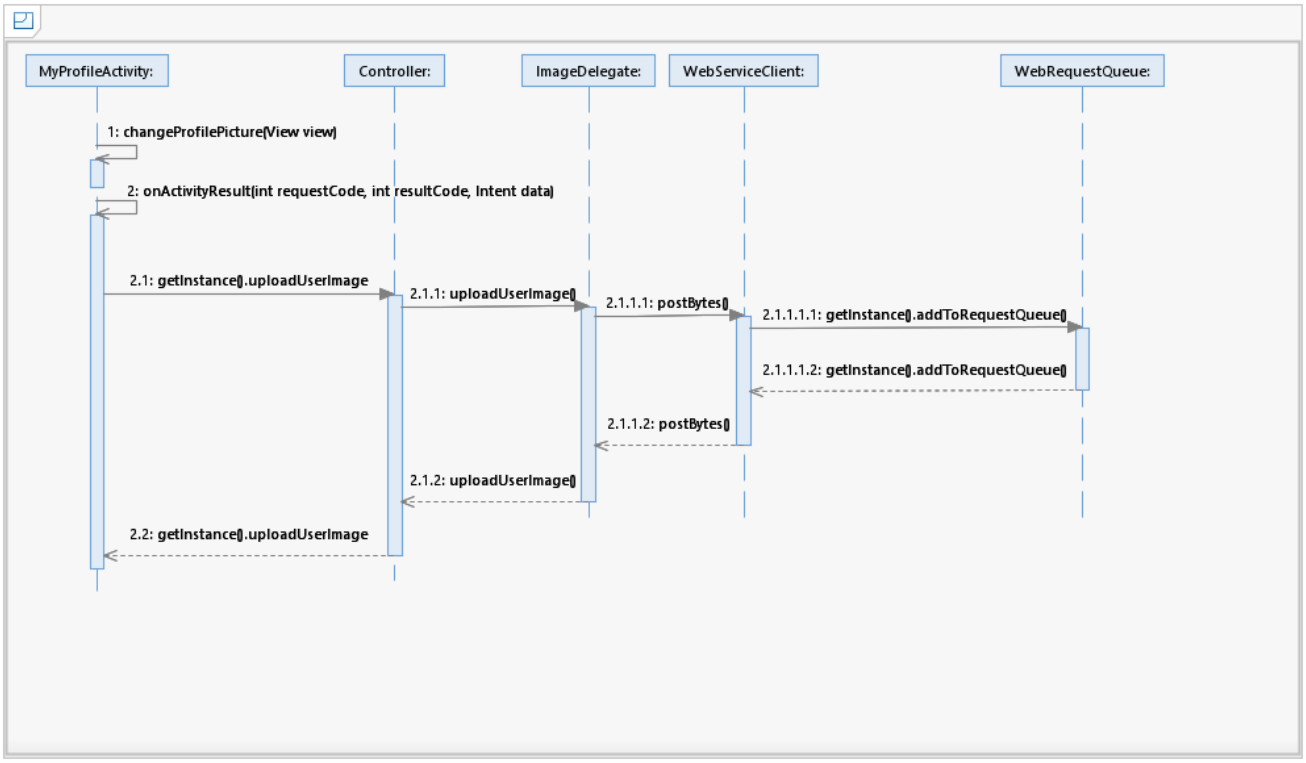
12.1 Buscar por filtro



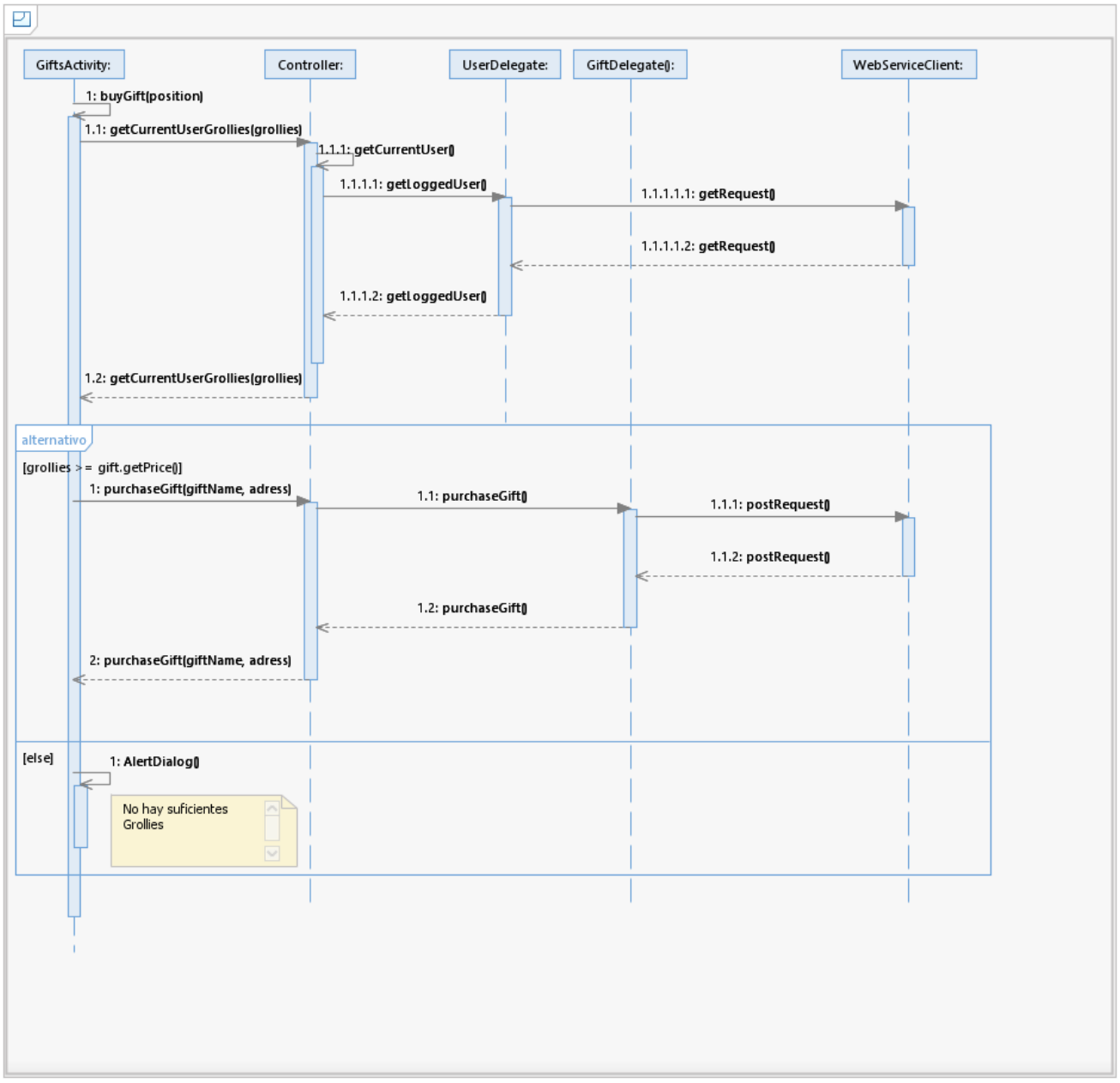
12.2 Cambiar contraseña



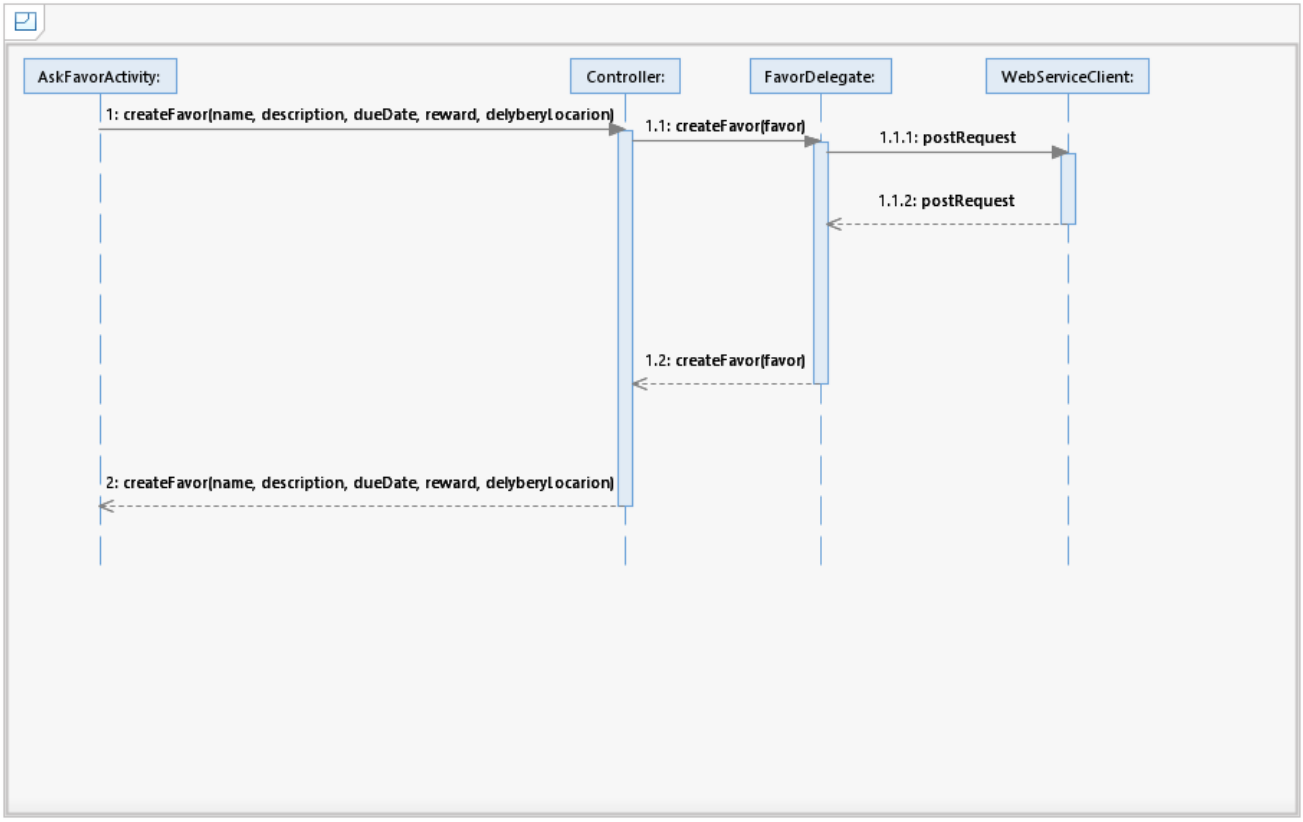
12.3 Cambiar foto de perfil



12.4 Comprar regalo



12.5 Crear favor



12.6 Iniciar sesión