



Cliente Android

05.06.2017

Ricardo Alexis Remache Sánchez

Bajo la supervisión de:

Luis Izquierdo López

IES GOYA

MADRID

Introducción	2
Objetivos	4
Especificaciones	5
Librerías:	6
Operaciones en Segundo plano	8
Creación de la base de datos local	12
Creación de las llamadas al API	14
Creación de la clase ApplicationConfig	15
Creación de las pantallas de la aplicación.	15
Creación de una interface que gestiona la respuestas de servidor.	18
Traducción de la aplicación a diferentes lenguajes.	22
Pruebas realizadas	22
Futuras Mejoras	23
Biografía	23

Introducción

Según avanza el tiempo el uso de smartphones es cada vez más extendido en la población, siendo muchas veces imprescindibles en la vida diaria, ya sea para el uso social u orientado al trabajo.

Siendo más específicos, las cifras de penetración en España son sorprendentes, un 96% de la población tiene un dispositivo móvil de los cuales más del 80% son smartphones.

Ya en 2016 los móviles superaban al ordenador tradicional en acceso a internet, no es de extrañar como el mercado de las aplicaciones móviles es un nicho en constante expansión.

Por ello y dado que toda mi experiencia en las prácticas de empresa estuvieron relacionadas con este entorno, he decidido crear una aplicación para la plataforma Android.

Plataformas de streaming como Netflix, Amazon, HBO o Movistar, han abierto una gran oferta de entretenimiento a precios competitivos y con ello multitud de usuarios consumiendo series y películas a su propio ritmo.

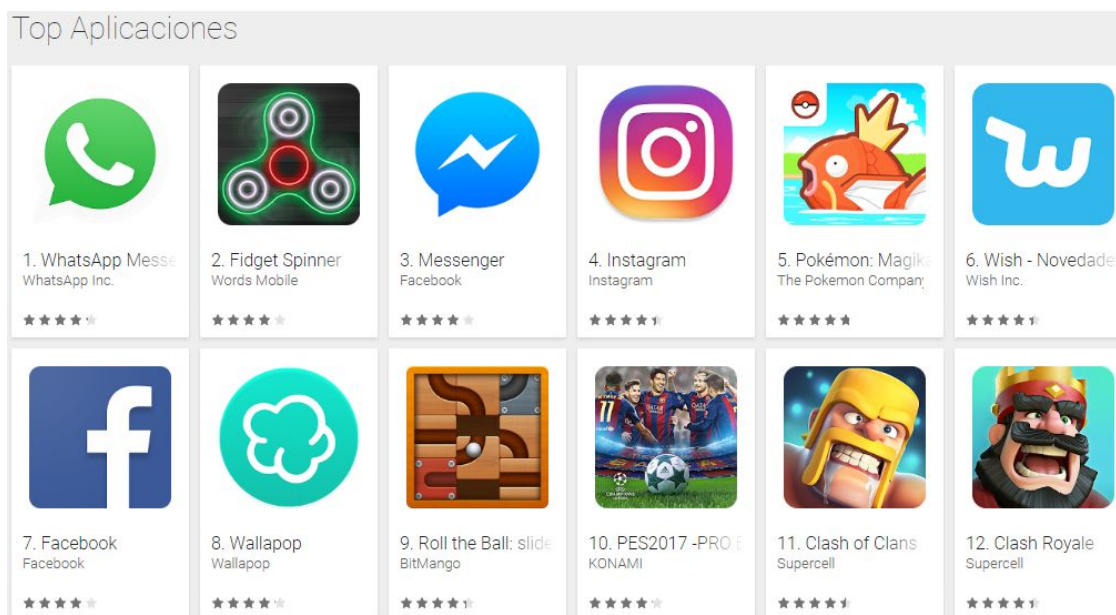
Sitios como themoviedb.org y trakt.tv han sido creados para gestionar todo el contenido que sus usuarios consumen, llevando un registro de todo los recursos audiovisuales que han visto, permitiéndoles calificar, opinar, además de proveer de información técnica, como actores o directores, entre otras cosas.

Cada una de estas plataformas están dotadas de un API.

¿Qué es un API?

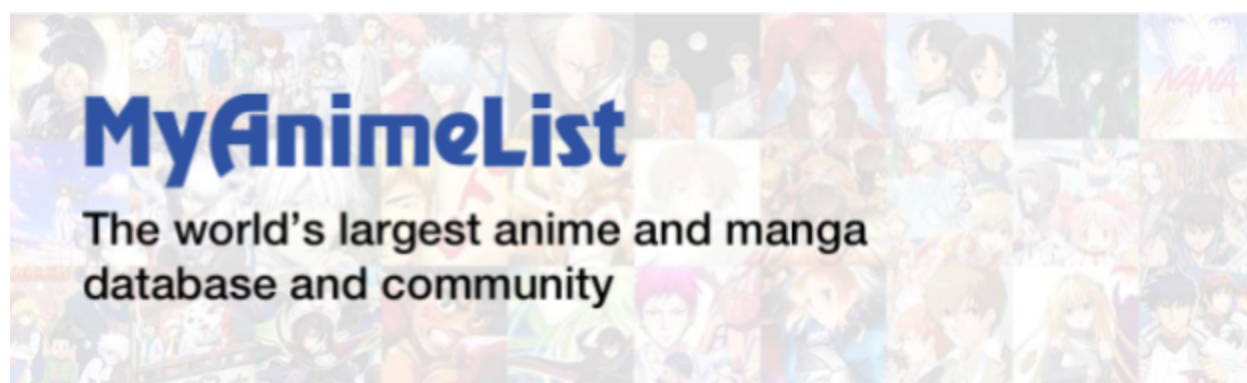
En los términos que nos conciernen, las API nos proveen de información, constantemente en actualización, estas nos permiten a los programadores consumir información a través de llamadas a los endpoints que estas disponen. Dependiendo del sitio consultado estas requieren de ciertos parámetros para su uso, como una api key generada por el proveedor, aunque también existen las de libre uso.

Estas por regla general suelen responder a las peticiones de los clientes con ficheros XML o JSON principalmente, aunque este último va prevaleciendo en las nuevas plataformas, por su fácil tratamiento, en contraposición del XML.



Actualmente en el mercado de las aplicaciones todas las APPS más descargadas se basan en la interacción con un servicio en línea por lo que conocer el funcionamiento y el comportamiento que un dispositivo móvil con un servidor es fundamental profesionalmente.

Conocidos estos datos el presente proyecto desarrolla una aplicación para Android, que servirá de cliente para la plataforma MyAnimeList.net un servicio similar a los anteriormente mencionados, donde la principal diferencia radica en estar focalizada en series de Animación Japonesa.



¿Por qué MyAnimeList.net ?

- Cada usuario registrado tiene una lista personal con todas las series agregadas por el usuario, estas muestran el estado en el que se encuentran, cuantos episodios han

sido vistos, la nota que este le ha dado, etc., siendo fácil la manipulación de datos por parte del programador.

- El [API](#) del que dispone este sitio es realmente sencilla, dispone de un limitado número de métodos para realizar peticiones HTTP, buscar, añadir, actualizar, borrar y verificar credenciales, cada uno de estos métodos, se realizan por GET O POST, requieren de las credenciales del usuario y la respuestas del servidor son XML , además de permitir un uso libre de sus recursos, aunque también tiene sus inconvenientes que veremos más adelante, ligados principalmente a las limitaciones de la propia API.
- Personalmente me gusta la cultura japonesa y su animación también está incluida.

Por esto es necesario la creación de una cuenta en el servicio sobre la cual llevar todas pruebas que se realizarán en el desarrollo del proyecto, esta tendrá las siguientes credenciales:

Nombre de usuario: ricardoAlexis

Contraseña: proydam**

Objetivos

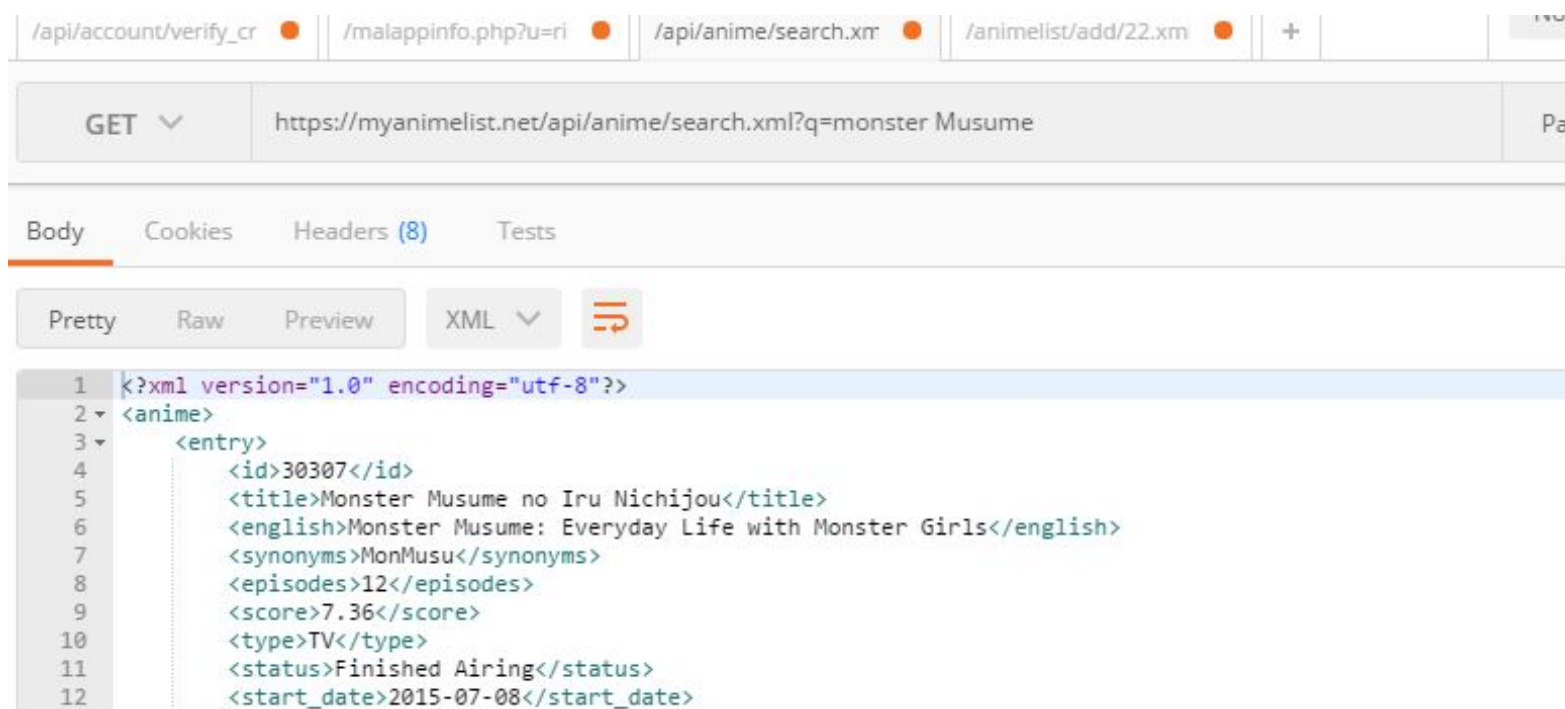
1. Crear un aplicación funcional y estable para dispositivos android.
2. Explotar los beneficios de utilizar APIs web, que nos proveen de la información concreta que estamos buscando.
3. Conectar un servicio web con una aplicación a través de peticiones HTTP REST.
4. Gestionar eficazmente en la base de datos local del dispositivo, los contenidos descargados de la llamadas al API, para limitar el número de peticiones al servidor, lo que conlleva a un menor consumo de recursos tanto por parte del servidor como por el cliente, el cual ve reducido su consumo de datos.
5. Sincronización de contenidos, la información reflejada en la aplicación debe estar al día con lo reflejado en el perfil de usuario.
6. Control del uso de la aplicación en modo offline, cuando el usuario se encuentre desconectado de internet y este realiza operaciones en nuestra aplicación, esta tendrá que gestionar todas esas peticiones pendientes para cuando volvamos a tener una conexión estable, actualizando en el servicio lo realizado anteriormente.

7. Aprender el uso de librerías concretas, para un mejor uso de los recursos del dispositivo.
8. Utilizar los conocimientos aprendidos en las prácticas de empresa.

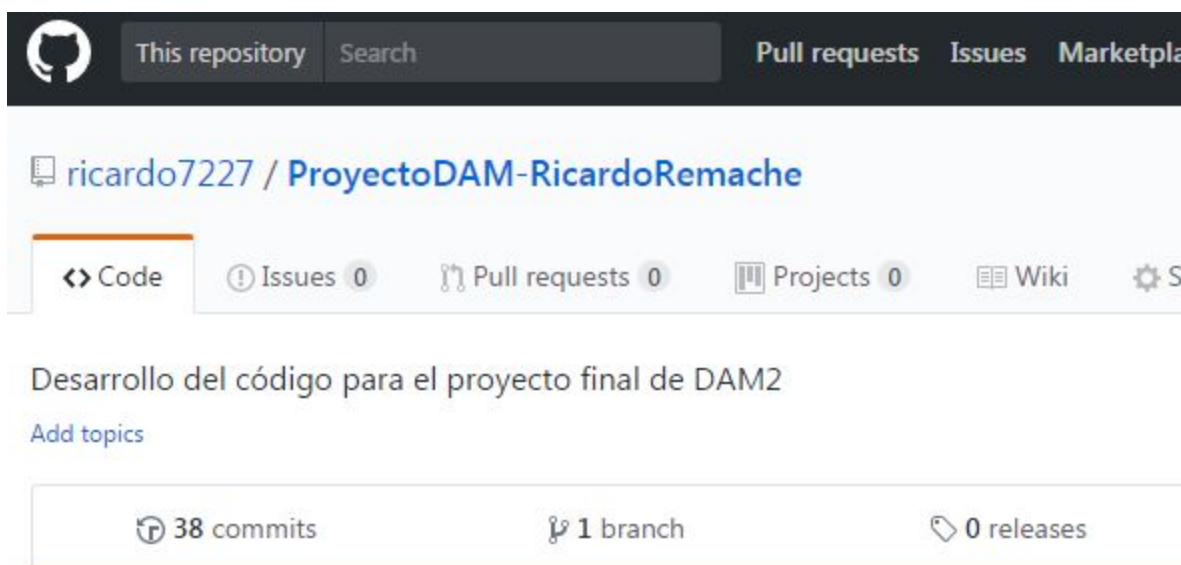
Especificaciones

Las herramientas utilizadas para la llevar a cabo este proyecto son las siguientes:

- Java es el lenguaje en el que se encuentra desarrollado el proyecto.
- Android Studio es el IDE seleccionado para escribir todo el código, principalmente por ser la plataforma oficial para desarrollar aplicaciones Android.



- [Postman](#) es una extensión para el navegador Google Chrome el cual nos permite hacer peticiones al API para realizar la pruebas pertinentes de su funcionamiento.



- Github es una plataforma online que permite almacenar código de manera gratuita en la cual puedes llevar un control de todos los cambios realizados en el proyecto, en este [enlace](#) se puede acceder a todo el código actualizado.

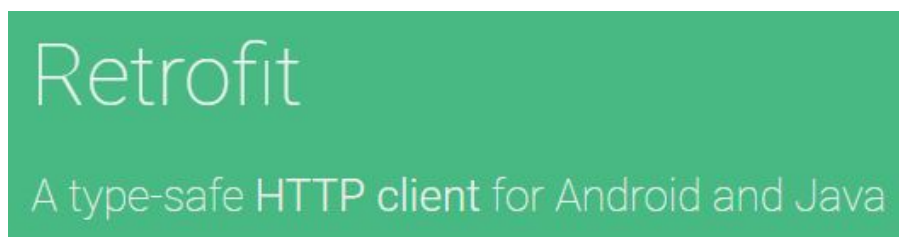
Dado que en Android se tiene que especificar cual el límite inferior de los dispositivos compatibles con la aplicación, basándonos en la información [oficial de google](#) donde muestra una gráfica con las versiones de los dispositivos activos que han accedido en los últimos 7 días a su market, he decido trabajar con la SDK 15 como mínimo, la cual funciona en las versiones 4.0.3 Ice Cream Sandwich de Android.

Librerías:

El proyecto utiliza un conjunto de librerías de las cuales es necesario explicar su papel dentro de la aplicación.

Para su uso es necesario declararlas en la zona de dependencias del build.gradle de la aplicación, de esta manera el propio android studio se encargará descargarlas y agregarlas a nuestro proyecto.

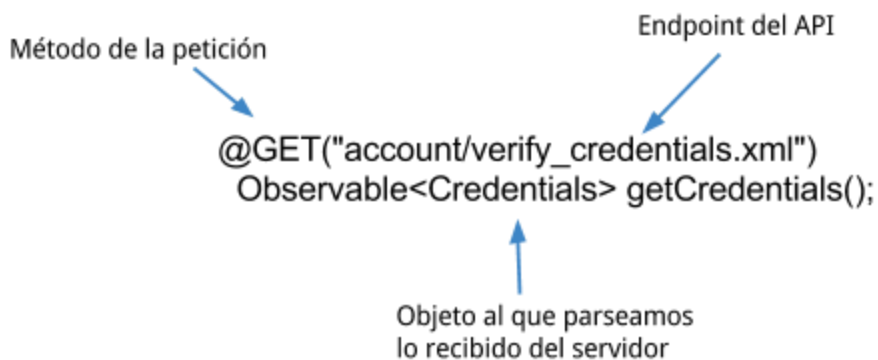
```
compile 'com.squareup.retrofit2:retrofit:2.2.0'
```



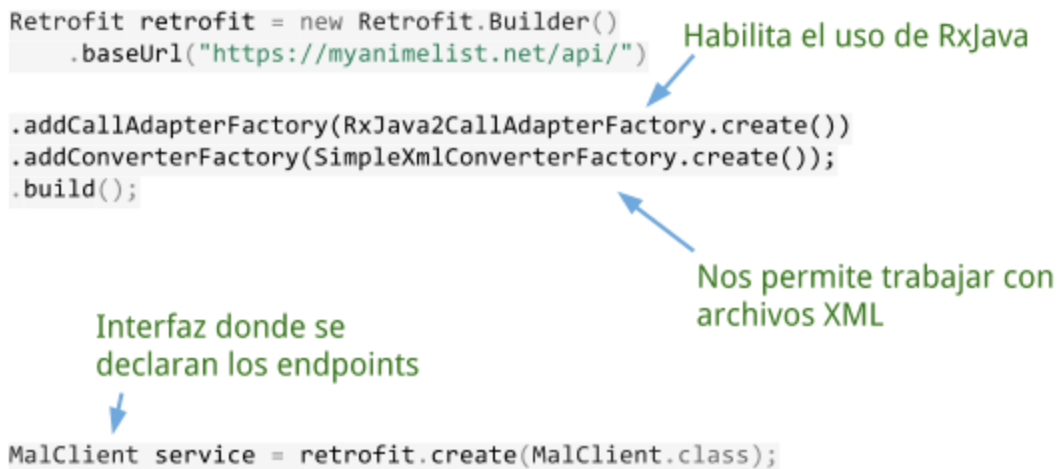
RETROFIT

Retrofit es un cliente REST para Android, permite hacer peticiones GET, POST, PUT, PATCH, DELETE y HEAD, gestiona diferentes tipos de parámetros y parsea la respuesta a un POJO (Plain Old Java Object).

Este requiere la creación de una interfaz en la que se declaran los endpoints de las llamadas al API.



Ahora creamos el servicio:



```
Observable<Credentials> credenciales = service.getCredentials();
```

Llamada al API



[RXJAVA](#)

RxJava es una librería que trata de facilitar la programación asíncrona y basada en eventos mediante el uso de Observables. Esta nos permitirá integrar con Retrofit para realizar peticiones al servidor de manera asíncrona es decir crear hilos para las llamadas, esto es fundamental por la forma en la que está concebida una aplicación en Android.

Operaciones en Segundo plano

Todas las aplicaciones tienen su hilo principal en el que trabajan, pero tareas que pueden llevar más tiempo de lo habitual están prohibidas, si creamos una aplicación en la que realiza una operación que puede llevar mucho tiempo como un cálculo muy complejo en el hilo principal que bloquee la aplicación por más de cinco segundos, el mismo sistema android se encargará de notificar al usuario por medio de un mensaje que nuestra aplicación no responde y le dará la opción de forzar su cierre.

Por ello todas las operaciones que se conecten con un servicio en red, no se pueden realizar en el hilo principal, además que el mismo android Studio en el momento de compilación saltará la siguiente excepción [NetworkOnMainThreadException](#).

RxJava nos ayuda en esta atolladero, nos permite crear hilos para este tipo de tareas y es una mejora sobre **AsyncTask** solventando alguno de sus problemas como los leaks de memoria.

Además de esto RxJava nos dota de más posibilidades en la programación reactiva de las cuales necesito más tiempo en su estudio.

En nuestro proyecto lo utilizaremos de la siguiente manera para la realizar todas las llamadas al API:

```

Observable<Credentials> credentialsObservable = malClient.getCredentials();

credentialsObservable
    .subscribeOn(Schedulers.newThread())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Observer<Credentials>() {
        @Override
        public void onSubscribe(@NonNull Disposable d) {
            Logger.d("onSubscribe getCredentials");
        }

        @Override
        public void onNext(@NonNull Credentials credentials) {
            Logger.d(credentials.toString());
            apiResult.SuccessCall(credentials);
        }

        @Override
        public void onError(@NonNull Throwable e) {
            Logger.e("onError getCredentials: ", e.getMessage());
            apiResult.ErrorCall(e);
        }

        @Override
        public void onComplete() {
            Logger.d("onComplete getCredentials");
        }
    });

```

Acceso al servicio

Hilo en el que se mostrarán los resultados

Crea un hilo para la operación

Si todo marcha correctamente recibe la respuesta esperada del servidor

Cuando existe un problema en la llamada

Finaliza la operación correctamente

```

//////////dagger2
component = DaggerAppComponent.builder()
    .databaseApplicationComponent(ChaikaApplication.get(this).component())
    .build();

component.injectMain(this);

```

DAGGER2

Dagger2 es un inyector de dependencias, compuesto por módulos y componentes.

Que es la inyección de dependencias?

Es una herramienta comúnmente utilizada en varios patrones de diseño orientado a objetos, consiste en inyectar comportamientos a componentes, se suministran objetos a una clase en lugar de ser la propia clase la que cree el objeto.

El patrón de inyección de dependencias consiste en hacer que nuestras piezas de software sean independientes comunicándose únicamente a través de un interface.

Con esto buscamos optimizar los recursos que consume la aplicación reutilizando instancias creadas y evitando crear nuevas innecesariamente.

En nuestro caso, la aplicación no está integrada completamente con Dagger2 debido a la complejidad de esta y la falta de tiempo, únicamente la he utilizado para acceder a la base de datos local de la aplicación.

Para ello he creado los módulos y el componente necesario para ello.

En los módulos se declaran el contexto y las clases de la base de datos, mientras en el componente se declaran como se accede a estos recursos.

Finalmente en la clase que extiende de Application la cual es arrancada cuando se activa la aplicación, declaramos los componentes de Dagger para su posterior consumo en cualquier parte de la Aplicación.

Diagrama de anotación de código que muestra la inyección de dependencias en ChaikaApplication:

```
ChaikaApplication.get(this).component().getData().getMyAnimeById(animeId);
```

Las anotaciones y flechas indican:

- Clase que extiende de Application**: Flecha hacia `ChaikaApplication`.
- Actividad**: Flecha hacia `this`.
- Acceso al componente**: Flecha hacia `component()`.
- Acceso a la base de datos**: Flecha hacia `getData()`.
- Uno de los metodos de la DB**: Flecha hacia `getMyAnimeById(animeId)`.



GLIDE

Es una librería fácil de usar utilizada para el tratamiento de la imágenes presentes dentro de la aplicación, tiene muchas opciones de personalización y soporta distintos formatos.

Actividad o Contexto Imageview donde se mostrará

```
Glide.with(this).load("http://goo.gl/gEgYUd").into(imageView);
```

Ruta de la imagen, local como remota

jsoup: Java HTML Parser

`jsoup` is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.

JSOUP

Librería que permite acceder al html de una url y extraer toda la información relevante que nos pueda interesar utilizando los identificadores de contenido que contiene la página.

Utiliza la técnica de web scraping, consistente en extraer información de páginas web de forma automatizada.

Dado que la librería funciona de manera síncrona y requiere conectarse con una url determinada, haremos uso de RxJava para realizar sus operaciones en segundo plano y mostrar sus resultados por pantalla una vez completada.

Conecta con la url y recupera su información



```
Document document = Jsoup.connect(UrlAPIs.BASE_URL_MAL + "anime/" + animeId).get();
```

Del fichero recuperado, extraemos la información que nos interesa
(tenemos que revisar previamente el HTML para conocer el identificador del contenido)



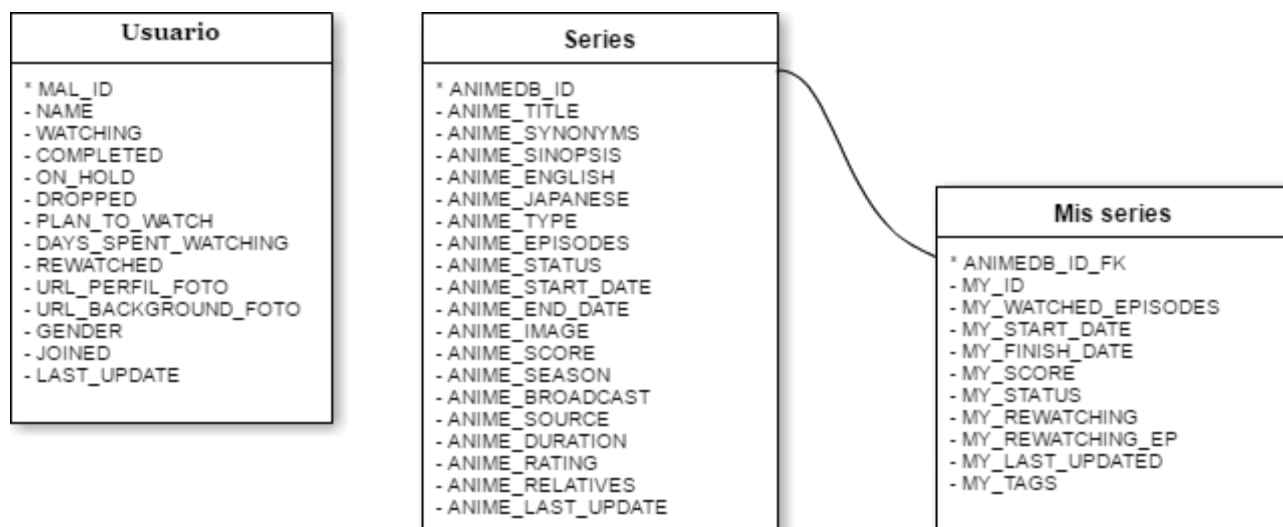
```
Elements sinopsisScrap = document.select("span[itemprop=\"description\"]");
```

Creación de la base de datos local

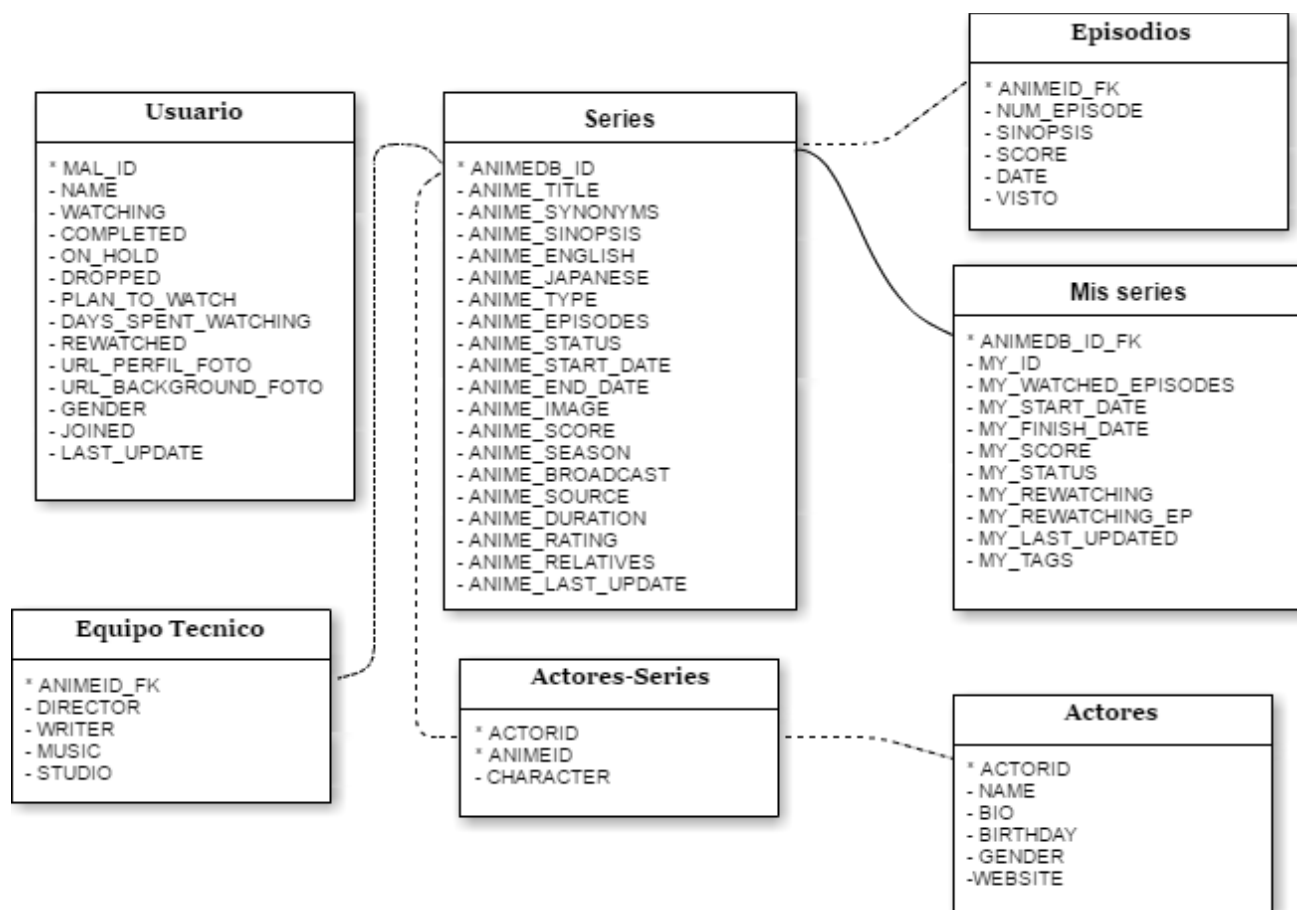
Una de los primeros pasos para la creación del proyecto fue concebir la base de datos que necesitaría la aplicación, como esta se basa en la información de series y el usuario, para esta primera versión partiremos con dos entidades:

- Usuario.
- Series.

Pero dado que todas las series no pertenecen al usuario necesitaremos una nueva tabla en la que almacenar la información personal que el usuario tiene para sus series.



Para futuras mejoras, añadiremos información más detallada por cada episodio e información del equipo técnico de cada serie.



Ya dentro de Android, la creación de la base de datos se llevará a cabo con la utilización de dos clases, la primera MalDBHelper que es una extensión de SQLiteOpenHelper, la cual nos permite la creación de una base de datos SQLite, donde crearemos todas las tablas utilizadas y que en el momento de su ejecución, esta se encargará de crearla.

_id	MAL_ID	NAME	WATCHING	COMPLETED	ON_HOLD	DROPPED	DAYS_
	6176877	ricardoAlexis	68	469	12	4	113.335

Showing 1 to 1 of 1 entries

[Tabla de usuario en la base de datos](#)

Previous

1

Next

Mientras para manipularla crearemos una clase Data, que contiene todos los métodos, que iremos necesitando, cada uno de estos métodos, abrirán una conexión con la base de datos para insertar, actualizar o borrar datos de ella.

ANIMEDB_ID	ANIME_TITLE	ANIME_SYNONYMS
17	Hungry Heart: Wild Striker	
19	Monster	; Monster
20	Naruto	NARUTO; Naruto
21	One Piece	; One Piece
22	Prince of Tennis	Tennis no Ouji-sama; The Prince of Tennis
25	Sunabozu	Sunabozu; Desert Punk
29	Zipang	
30	Neon Genesis Evangelion	Shinseiki Evangelion; Neon Genesis Evang
31	Neon Genesis Evangelion: Death & Rebirth	Shinseiki Evangelion Gekijouban: Shi to Sh
32	Neon Genesis Evangelion: The End of Evangelion	Shinseiki Evangelion Gekijouban: The End

Showing 1 to 10 of 586 entries Previous 1 2 3 4 5 ... 59 Next

[Tabla de Series en la base de datos](#)

Creación de las llamadas al API

Como ya mencionamos en el apartado de Retrofit la forma de crear cada uno los métodos necesarios para añadir, buscar, actualizar y borrar, se encuentran detallados en la [documentación oficial](#).

Para esto se creó un clase que genera un servicio para responder a cada una de las peticiones y que es utilizado para crear una nueva clase encargada RestApiMal, la cual contiene todos los métodos para realizar llamadas HTTP.

Pero existe un problema, dentro de esta documentación, no existe un método específico para recibir [la lista completa](#) de series que el usuario tiene almacenado en su perfil, lo cual supone algo difícil con lo que lidiar, sin embargo tras un búsqueda por los [foros del sitio](#), existe una llamada al API que no está documentada:

```
/malappinfo.php?u=username&status=all&type=anime
```

Devuelve toda la información importante del usuario con la cual comenzar a trabajar, ya que será la primera llamada en la primera instalación en el dispositivo.

Creación de la clase ApplicationConfig

Esta clase tiene por objetivo contener toda la configuración de la aplicación, en esta se definen parámetros a los cuales podemos acceder a ellos desde cualquier clase.

Para esto será necesario la creación de un Singleton, una instancia única de la clase con la que acceder a los recursos de esta .

```
private static ApplicationConfig instance;

public static ApplicationConfig getInstance(){
    if (instance == null){
        instance = new ApplicationConfig();
    }
    return instance;
}
```

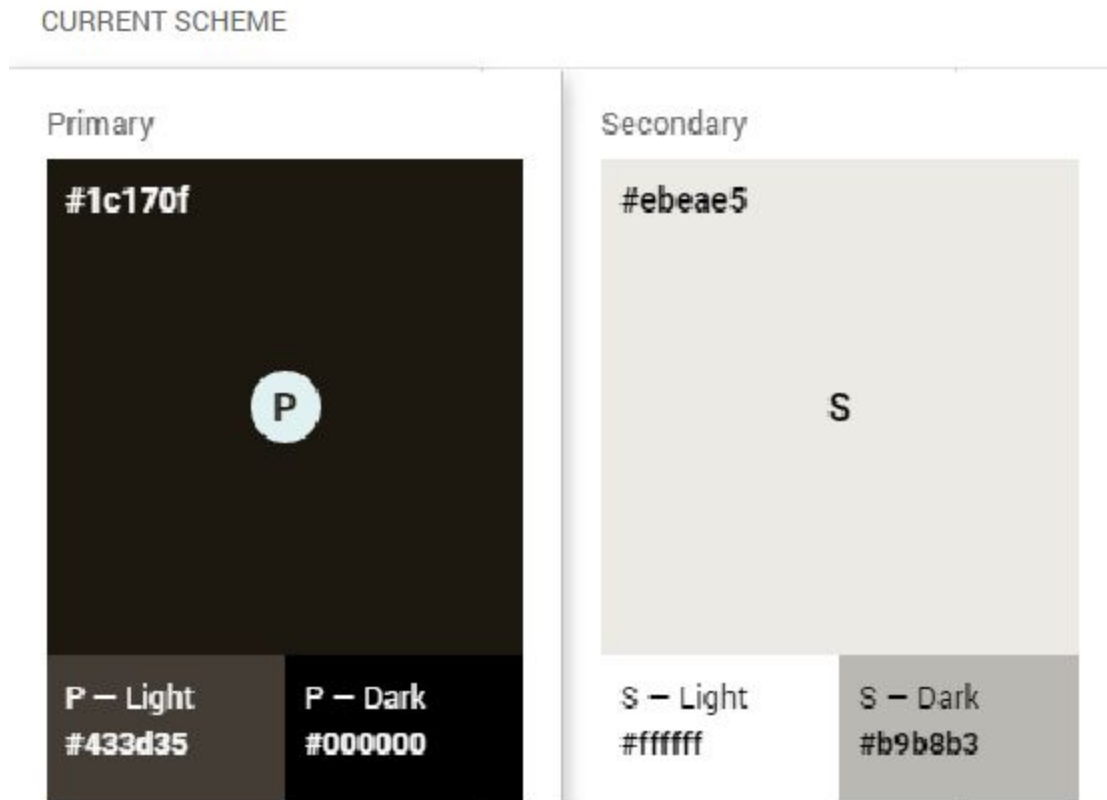
Aquí almacenaremos un contexto, una actividad y las credenciales del usuario necesarias para realizar las peticiones que requieren su identificación.

De esta manera podemos tener acceso a recursos que requieran una actividad o un contexto fuera de Actividades.

Creación de las pantallas de la aplicación.

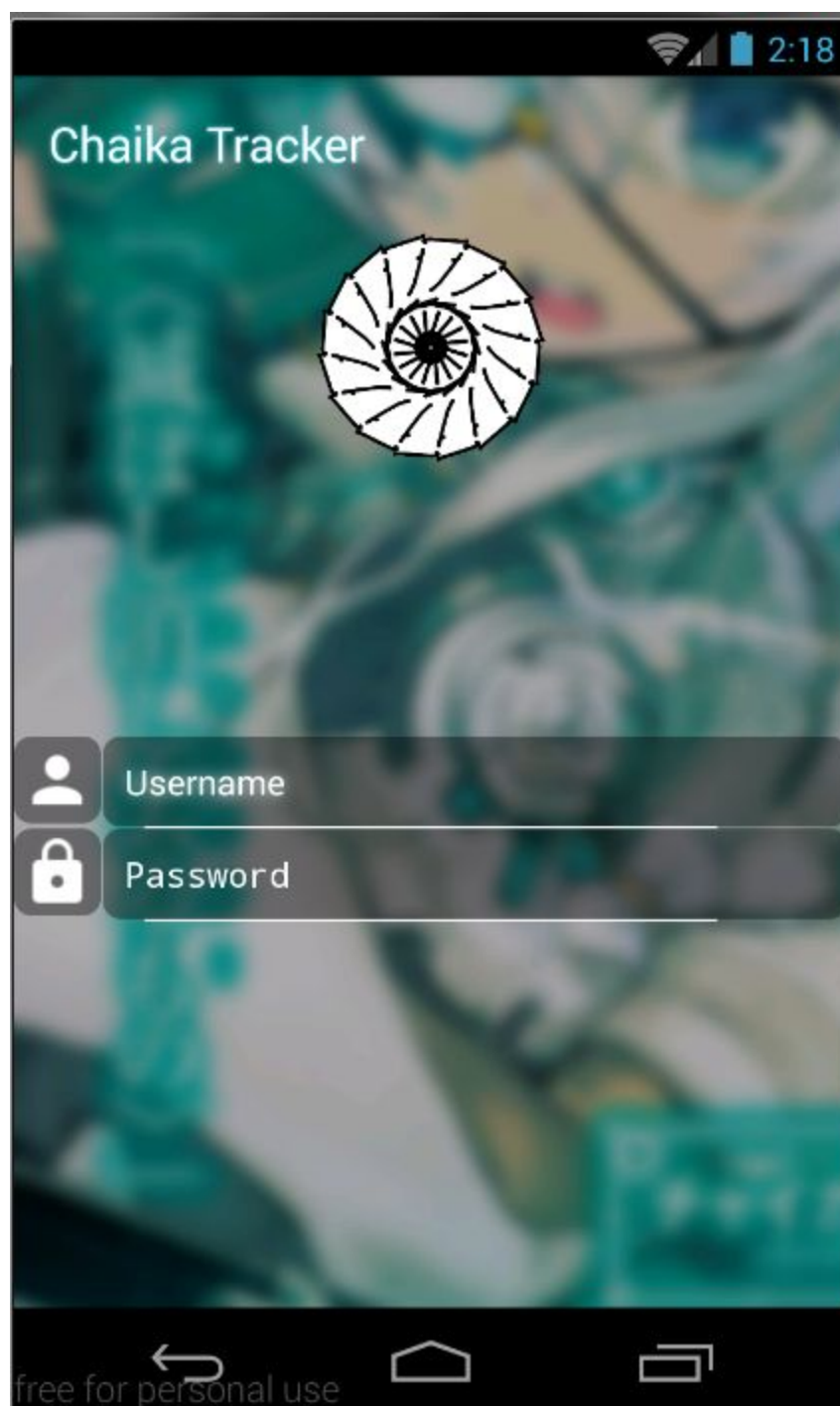
Una vez solucionados los problemas que existieron en la parte anterior. La parte gráfica representa otro punto fundamental y complicado de abordar, el objetivo es ser agradable y atractiva a la vista para nuestro futuro usuario, además de adaptarse a los distintos tamaños de pantallas que existen en el mercado.

Primero seleccionamos un patrón de colores que utilizaremos en todas pantallas, para la cual utilizamos la herramienta que nos proporciona material.io.



La Creación de pantallas requiere de dos partes, por un lado el diseño está realizado en XML y la parte de programación en la que definimos las partes de esta, así como lo necesario para manipular su contenido.

Actualmente la aplicación consta de una pantalla de Login, en esta el usuario tiene dos campos para rellenar, nombre de usuario y contraseña, el objetivo de esta pantalla es controlar los datos personales que el usuario introduce en la aplicación. Está hace un control de la información, comprobando la validez de los caracteres del nombre, los posibles valores nulos y un mínimo de caracteres tanto en el nombre como en la contraseña. Si estos pasan el primer control local, se habilitará el botón de login, para verificar la validez de las credenciales, si la llamada es exitosa informaremos al usuario y pasamos a la siguiente pantalla, en caso contrario informaremos al usuario que se han introducidos datos incorrectos para un nuevo intento.



La pantalla principal de la aplicación y a la cual accederemos siempre y cuando tenga las credenciales del usuario, es lo que siempre se mostrará cuando se arranca la aplicación en posteriores usos, para acceder a ella completamente, en la pantalla de login previamente se ha realizado una petición para recibir todas las series.

Aquí es donde la cosa se complica un poco porque se están realizando varias operaciones a la vez, esta pantalla a la cual corresponde el MainActivity realizará las siguientes tareas:

- Inicializa la clase ApplicationConfig definiendo su actividad y contexto.
- Hace uso de la clase ViewPagerAdapter que es una extensión de FragmentStatePagerAdapter, un adaptador de fragmentos que permite mostrar distintos fragmentos en una misma actividad a través de un desplazamiento horizontal de la pantalla.

Qué es un Fragmento?

Es una clase reutilizable que utiliza una porción de una actividad, se podría considerar una versión pequeña de una actividad, esta también tiene una parte en XML en la que definir su parte gráfica, pero no puede utilizarse fuera de una actividad, son muy utilizados, mejoran la navegación y evita la creación de nuevas actividades innecesariamente, ya que pueden ser intercambiables con otras dentro de la misma actividad.

- Varios fragmentos se agregan al adaptador, donde cada uno de estos fragmentos tienen sus tareas de arranque.
- En la primer arranque de la aplicación, el primer fragmento que se agrega al adaptador AllSeriesFragment, recibe la respuesta del servidor con todos los datos de la series, haciendo uso de una interface previamente creada.

- Creación de una interface que gestiona la respuestas de servidor.

Dado que dependiendo de la respuesta que recibimos del servidor procederemos de una manera o de otra, he creado la interface ApiResult.

Esta contiene todas las respuestas posibles que obtendremos del servidor,

```
void SuccessCall(MyAnimeList myAnimeList);//llamada exitosa
```

Para hacer uso de ella simplemente la implementamos en donde queremos recibir respuesta. `implements ApiResult{`

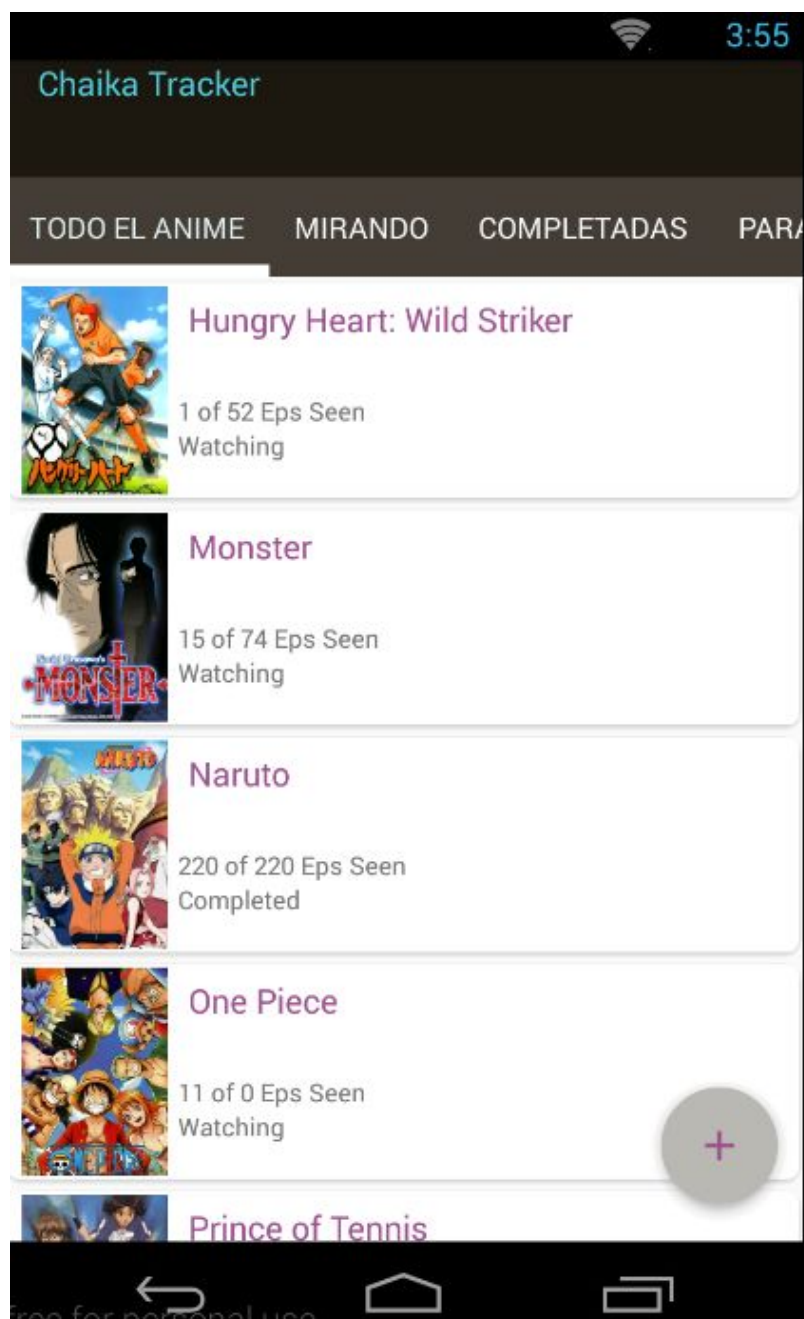
Y en la llamada indicaremos donde tiene que enviarse la respuesta.

```
RestApiMal.getInstance().  
    getMalUserProfile(ApplicationConfig.getInstance().getUsername()  
        , "all", "anime",  
        AllSeriesFragment.instance());
```

Diagram annotations:

- Acceso al servicio**: Arrow pointing to `RestApiMal.getInstance()`
- Acceso al método al cual le proveemos del nombre del usuario almacenada en la configuración**: Arrow pointing to `ApplicationConfig.getInstance().getUsername()`
- Instancia de la clase que recibe la respuesta**: Arrow pointing to `AllSeriesFragment.instance()`

- En el primer fragmento cargado `AllSeriesFragment`, el cual recibe la respuesta del servidor con todas las series de la cuenta del usuario. Por un lado carga toda esa información para ser mostrada por pantalla en un `RecyclerView`, esta clase a través de un adaptador permite mostrar información en forma de lista, mientras por otra parte esa misma información recibida de la llamada, crea un hilo para almacenarla en base de datos, para que en posteriores arranques de la aplicación esta muestra la información almacenada en la base de datos local.



Como podemos ver en la pantalla principal, esta muestra una lista con todas las series que contiene la aplicación, permitiendo que cada uno estos elementos pueda generar una nueva actividad en el que mostrar con más detalle información específica de cada serie.

Esta nueva pantalla, llamada DetailSerie es una nueva actividad que carga la información disponible sobre la serie de la base de datos y la información que no es conseguible por

medio del API, hace uso del Web Scraping mencionado anteriormente para conectar con la url específica de cada serie para recuperar la sinopsis y la calificación global en la comunidad.

Además la pantalla permite al usuario actualizar la información personal respecto a la serie, como cuantos episodios ha visto , la calificación que este le da y está dotada de un botón adicional en la barra superior con la que eliminar la serie de nuestra lista personal. Tanto para el botón de actualizar como el de borrar al momento de ser accionados, la aplicación se encargará de realizar la petición pertinente al servidor e informar del resultado por medio de un mensaje emergente.

Queda pendiente implementar la realización de búsquedas que se muestren por pantalla y según ello añadir nuevas series a la lista personal

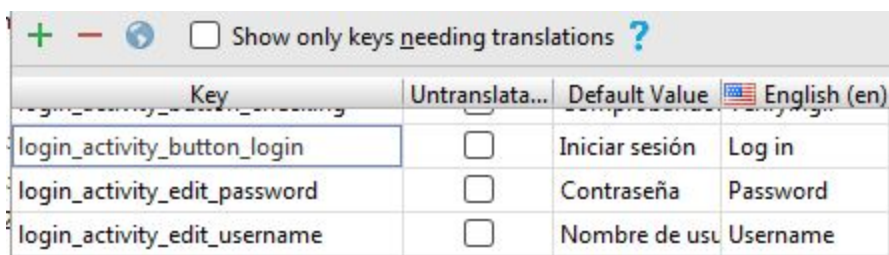


Traducción de la aplicación a diferentes lenguajes.

Con el objetivo de lanzar una aplicación en el mercado internacional, Android Studio provee al desarrollador una forma por la cual traducir todos los textos de la aplicación a distintos idiomas, para esto tenemos que declarar todos nuestros textos en el fichero strings.xml.

```
<string name="login_activity_button_login">Iniciar sesión</string>
```

Y en su editor de idiomas traducir todos los textos presentes.



Key	Untranslate...	Default Value	English (en)
login_activity_button_login	<input type="checkbox"/>	Iniciar sesión	Log in
login_activity_edit_password	<input type="checkbox"/>	Contraseña	Password
login_activity_edit_username	<input type="checkbox"/>	Nombre de usu	Username

Mientras en nuestro código referenciamos a estos de la siguiente manera:

```
getString(R.string.login_activity_button_login)
```

Android se encargará de localizar los textos pertinentes para el idioma del dispositivo.

Pruebas realizadas

Las pruebas más exhaustivas realizadas en la aplicación están ubicadas en la parte donde el usuario introduce valores manualmente esto es principalmente en la zona de login, he acotado al máximo los posibles errores a la hora de introducir caracteres erróneos o nulos, por ello la verificación de datos se realiza en dos niveles, en el primero impedimos al usuario acceder a botón para verificar sus credenciales siempre y cuando no cumpla nuestros parámetros, informando en todo momento de sus posibles fallos para su corrección, cuando este cumpla nuestros parámetros mostraremos el botón de Login con el que se realiza una petición al servidor, dependiendo lo que este nos devuelva procederemos a informar al usuario del resultado.

Una vez superado este primer escenario, en el resto de la aplicación la introducción de datos por el usuario de manera manual están reducidas al mínimo, siempre ofreciendo valores predefinidos por nosotros los cuales están controlados.

En cuanto al estado de conexión en el que la aplicación puede realizar operaciones todavía se encuentran pendientes por controlar, pero esto no quiere decir que no exista un control de las operaciones que se realizan, el usuario siempre se encuentra informado del resultado de las acciones realizadas por este.

Por otro lado en cuanto a la estabilidad de la aplicación, está responde bien al uso indebido que un usuario puede provocar, como navegación rápida por la aplicación, conectar y desconectar la conexión con la red reiteradas veces o el presionado de botones sin ningún tipo de criterio.

Pero este tipo de acciones RxJava es un buen aliado en la programación, ya que contempla todos los posibles errores en su funcionamiento enviando todas la posibles incidencias a través de onError, en el que nos informa que ha ocurrido y obrar según lo ocurrido.

Futuras Mejoras

Como se puede apreciar por lo comentado a lo largo de esta memoria, quedan pendiente la implementación de varias funciones para un futuro lanzamiento real en el mercado, como la búsqueda de series, mejorar la forma en que se muestran las series en la pantalla principal, crear una nueva distribución de los contenidos en pantalla más grandes como tablets para un mejor aprovechamiento de la pantalla, mejorar la pantalla de detalle añadiendo más funcionalidades y contenido, nuevas pantallas, una destinada a mostrar la información del perfil del usuario o permitir la exploración de nuevos contenidos.

Además de profundizar en el uso de Dagger2 y RxJava, optimizando el consumo de recursos, mejorar el procesamiento de las tareas y crear los servicios necesarios para llevar a cabo tareas pendientes.

Biografía

Código del proyecto

- <https://github.com/ricardo7227/ProyectoDAM-RicardoRemache>

Fragmentos

- <https://guides.codepath.com/android/Creating-and-Using-Fragments>
- <https://guides.codepath.com/android/ViewPager-with-FragmentManager>
-

Retrofit

- <https://futurestud.io/tutorials/android-basic-authentication-with-retrofit>
- <https://guides.codepath.com/android/Consuming-APIs-with-Retrofit>

Web Scraping

- <https://stackoverflow.com/questions/4714244/web-scraping-android-application>
- <https://jsoup.org/>

Documentación MyAnimeList API

- <https://myanimelist.net/modules.php?go=api>

RxJava

- https://github.com/codepath/android_guides/wiki/RxJava
- <https://github.com/arriolac/GitHubRxJava/wiki/Tutorial>
- <https://www.youtube.com/watch?v=k3D0cWyNno4>

Dagger2

- <https://www.youtube.com/watch?v=Qwk7ESmaCq0>
- <https://guides.codepath.com/android/Dependency-Injection-with-Dagger-2>
- <https://medium.com/@methodsignature/dagger-2-dependency-injection-for-android-developers-51d60e7397e6>
- <https://blog.mindorks.com/introduction-to-dagger-2-using-dependency-injection-in-android-part-1-223289c2a01b>
-

Paleta de colores

- <https://material.io/color/#!/view.left=1&view.right=1&primary.color=1C170F&secondary.color=EBEAE5&primary.text.color=E1F1F1&secondary.text.color=9E4F91>

Apariencia

- <https://mzgreen.github.io/2015/06/23/How-to-hideshow-Toolbar-when-list-is-scrolling%28part3%29/>
- <https://material.io/guidelines/components/snackbars-toasts.html>
- <https://guides.codepath.com/android/Using-the-RecyclerView>
- <https://guides.codepath.com/android/Using-the-CardView>
-

Permisos

- <https://guides.codepath.com/android/Understanding-App-Permissions>

Base de datos

- <https://guides.codepath.com/android/Storing-and-Accessing-SharedPreferences>
- <https://guides.codepath.com/android/Local-Databases-with-SQLiteOpenHelper>

JavaDoc del proyecto

<https://github.com/ricardo7227/ProyectoDAM-RicardoRemache/tree/master/JavaDoc>

