LVS.Training.Android.Lib.DaggerRetrofit

How it works ?

# Where does it start ?

In AndroidManifest.xml we have this :

android:name=".root.App"

Extract from [developer.android.com](https://developer.android.com/guide/topics/manifest/application-element.html)

android:name

The fully qualified name of an [Application](https://developer.android.com/reference/android/app/Application.html) subclass implemented for the application. When the application process is started, this class is instantiated before any of the application's components.

The subclass is optional; most applications won't need one. In the absence of a subclass, Android uses an instance of the base Application class.

It means the entry point of the application is App.java

# App.java

This application object is where dagger will live throughout the entire life span of the application.

Here we instantiate the component (ApplicationComponent) and define all of the module we have in our app. Normally there is one module per feature. So here we instantiate our ApplicationModule (we always have to do this). And we have one feature, our API so we instantiate the ApiModule too.

private ApplicationComponent component;

@Override

public void onCreate() {

super.onCreate();

component = DaggerApplicationComponent.builder()

.applicationModule(new ApplicationModule(this))

.apiModule(new ApiModule())

.build();

}

public ApplicationComponent getComponent() {

return component;

}

The second method getComponent() always have to be here, it will return our component to inject all the dependencies.

# ApplicationModule.java

ApplicationModule is where Dagger will keep track of the dependencies.

@Module

public class ApplicationModule {

private Application application;

public ApplicationModule(Application application) {

this.application = application;

}

@Provides

@Singleton

public Context provideContext() {

return application;

}

}

# ApiModule.java

The methods we implement in ApiModule.java can be used by Dagger to provide an instance of the Twitch API client.

@Module

public class ApiModule {

public final String BASE\_URL = "<https://api.twitch.tv/kraken/>";

@Provides

public OkHttpClient provideClient() {

HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();

interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);

return new OkHttpClient.Builder().addInterceptor(interceptor).build();

}

@Provides

public Retrofit provideRetrofit(String baseURL, OkHttpClient client) {

return new Retrofit.Builder()

.baseUrl(baseURL)

.client(client)

.addConverterFactory(GsonConverterFactory.create())

.build();

}

@Provides

public TwitchAPI provideApiService() {

return provideRetrofit(BASE\_URL, provideClient()).create(TwitchAPI.class);

}

}

In the **first method** we return a **OkHttpClient** (this method is only used for login purpose when getting data from API).

In the **second method** we return a **Retrofit**. Retrofit is used for deserialized our JSON data from Twitch API to POJOs.

In the **last method** we return a **TwitchAPI**. As we will use Retrofit for parsing our API data we use our second method and pass to it our TwitchAPI class.

At compilation Dagger will generate a class for every method of every module, so here 3 class implementing 3 factories :

* ApiModule\_ProvideApiServiceFactory implements Factory<TwitchAPI>
* ApiModule\_ProvideRetrofitFactory implements Factory<Retrofit>
* ApiModule\_ProvideClientFactory implements Factory<OkHttpClient>

Retrofit and OkHttpClient are libraries, so I'll not show details about what they contains. But we must have our TwitchAPI in the project.

# TwitchAPI.java

TwitchAPI is just an interface defining all the method we will need.

public interface TwitchAPI {

@GET("games/top")

Call <Twitch> getTopGames(@Header("Client-Id") String clientId);

}

"games/top" is the relative path to our API end point as provided with the Twitch API documentation.

Here we sent the API client id too with @Header("client-id") String clientId

We call this getTopGames method in the MainActivity

# MainActivity.java

First we call the App.getComponent() method to inject all our modules defined in App.java (ApplicationModule, ApiModule) :

(App)getApplication()).getComponent().inject(this);

Then instantiate a call to the API method getTopGames defined in our interface TwitchAPI.java. This will get the data from Twitch API throughout the network :

Call<Twitch> call = twitchAPI.getTopGames(getResources().getString(R.string.client\_id));

Retrofit will deserialized the JSON data to the POJO Twitch.java. This Twitch.java class was created by pasting the online Twitch API JSON response from our internet browser to the website [http://www.jsonschema2pojo.org](http://www.jsonschema2pojo.org/). In our case only the getTop() method will be used.

public class Twitch {

@SerializedName("\_total")

@Expose

private Integer total;

@SerializedName("\_links")

@Expose

private Links links;

@SerializedName("top")

@Expose

private List<Top> top = null;

public Integer getTotal() {

return total;

}

public void setTotal(Integer total) {

this.total = total;

}

public Links getLinks() {

return links;

}

public void setLinks(Links links) {

this.links = links;

}

public List<Top> getTop() {

return top;

}

public void setTop(List<Top> top) {

this.top = top;

}

}

Now we are making the call to the API :

call.enqueue(new Callback<Twitch>() {

@Override

public void onResponse(Call<Twitch> call, Response<Twitch> response) {

List<Top> gameList = response.body().getTop();

for (Top top : gameList){

Log.d(LOG\_TAG,top.getGame().getName());

}

}

@Override

public void onFailure(Call<Twitch> call, Throwable t) {

t.printStackTrace();

}

});

When we receive the response we can call the getTop() method on it who will return our top games.