Retrofit as a type-safe REST Client for Android Development

Albert Logianto 13514046
Informatics Study Programme
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jalan Ganesha 10 Bandung 40132, Indonesia
13514046@std.stei.itb.ac.id

***Abstract*—Almost everyone this day have a smartphone as their daily needs. More than half of them is using Android as their operation system. One important feature for a smartphone is the ability to connects to the internet. Therefore, Majority of applications in a smartphone is connected to REST based webservices. There are many ways to communicate to a REST webservices. One common method is using *httpurlconnection*, which is pretty slow and bad support for scalability. Retrofit solves that issues by making easy implementation of retrieving and uploading data (typically *JSON*) via a REST based webservice. Retrofit then provides a data converter using *POJO* for data serialization to easily map webservices response to Java Objects. Many popular applications use retrofit because of the great supports and flexibility in developing Android Applications.**

Keywords—Retrofit; webservices; POJO; Scalability; REST; http

I. Introduction

According to PEW Research Center (2017), there are roughly three-quarters (77%) of American own a smartphone. Almost all applications in a smartphone will be using internet to connect to webservices. Webservice is a service that provides communication between client and server. There can be many type of structure used in data exchanged between client and server.

Common architecture used in building a web service is using REST and SOAP. Roy Fielding first introduce REST in 2000 as a standard in developing web services. REST is more popular in mobile aspect because of its' flexibility and scalability compared to SOAP. One of the most popular data structure used in REST webservices is JSON (JavaScript Object Notation). REST web services is an architecture which provides stateless operation and has a URI (Uniform Resource Identifiers) to identifies resources needed.

One of the most popular techniques in developing a REST client in Android is using a framework called Retrofit. In this technique, the framework provides authentication and interaction to REST webservices by using OkHttp, which is a library for sending network requests and handling network response. The response (typically JSON) from the REST webservices is converted to a Plain Old Java Object for data serialization. By using this technique, developing client for connecting to webservices will be more simple and fast because every resource in the network response already handled and parsed by the framework, making it more straightforward to accessing the data resources retrieved from the webservices.

In this paper, we present an example of developing a simple application for user to discover popular movies by fetching data to a movie database REST API using Retrofit. Retrofit will be used to handling network request and response between client and the webservices. Using retrofit will speeds up the development time because almost all network aspects will be handled by the framework.

This paper consists of four sections. The first section will be the introduction. The second section will explain what is a REST webservices and Retrofit. The third section will discuss the movie database project. In the last section, the author will give some conclusion and benefits of using Retrofit in Android Development.

II. Theory

A. REST Webservice

First, we describe what is a web service. A web service is a protocol used in communicating between applications or server. One of the most popular architecture used in building a web service is REST. REST, which stands for Representational State Transfer is an architecture using stateless protocol which is HTTP for exchanging data in network communication. One of key aspect in REST web services is mapping every resource to an URI. There many data structure used in REST for representing the resource, such as plain text, XML, and JSON. JSON is one of emerging and most popular format used in many web services.

REST web services implement HTTP method, which is *GET, POST, PUT, DELETE* to represent the type of resource operation being used. The operation will be also linked to the URI (Uniform Resource Identifiers) to identify which resource will be manipulated. In REST architecture, everything will be treated as a resource. Therefore, REST web services are designed to be simple, lightweight, fast, and have a great scalability. So it suits perfectly and popular in developing web services for mobile application.

B. Retrofit

Retrofit is a type-safe REST/HTTP client for Android, it is developed by Square. It is a framework that turns HTTP API to a Java Interface. Typical data exchanged is in JSON format, to parsed it to POJO, GSON library is commonly used. To develops mobile application using Retrofit, we need to have three classes:

1. Model, which is a class to defined the JSON to a POJO
2. Interfaces, to declare set of operations and parameters used to web services
3. Retrofit.Builder class, to define instance to connect to an URL endpoint and converting the response to the model class that has been created.

III. Case Study

Before we begin to use Retrofit for our project, there are some setups that need to be done. First, make sure our project has the Internet Permissions in out AndroidManifest.xml file:

```
<manifest
xmlns:android="http://schemas.android.com/a
pk/res/android">
    <uses-permission
    android:name="android.permission.INTERN
    ET" />
</manifest>
```

Then, we need to add the dependencies to our gradle build settings which is located in app/build.gradle:

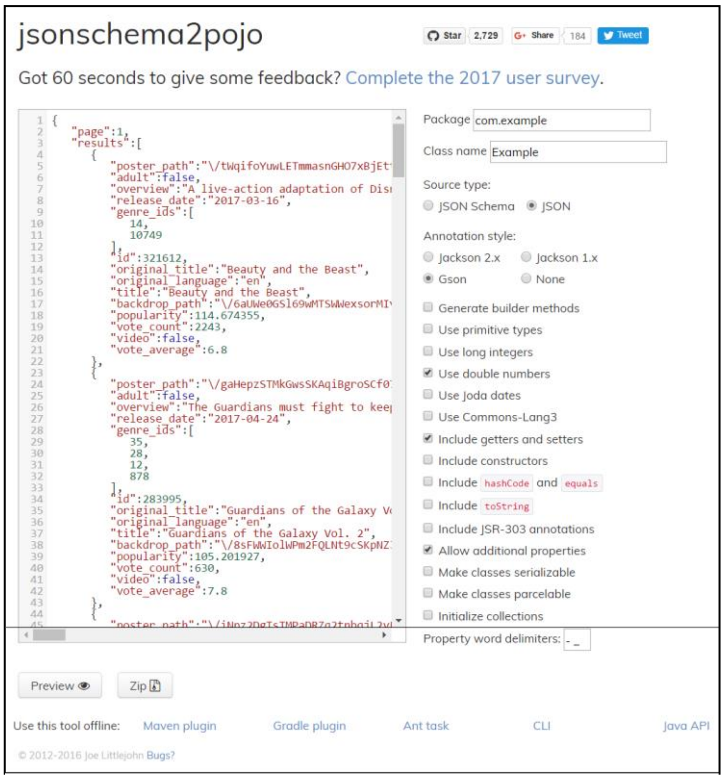
```
dependencies {
    compile 'com.google.code.gson:gson:2.7'
    compile
    'com.squareup.retrofit2:retrofit:2.1.0'
    compile
    'com.squareup.retrofit2:converter-
    gson:2.1.0'

}
```

We use GSON library for our project because we are dealing with JSON format in the web service response.

After the setup is done, we need to create our model classes based on the response format. There are automated and manual ways to create the model classes, the author will explain the automated method. The automated method is using a web site called jsonschema2pojo (http://www.jsonschema2pojo.org/). In this case, our URL endpoint will be https://api.themoviedb.org/3/movie/popular?api_key=<key> which return a JSON response:

```
{"page":1,"results":[{"poster_path":"\/tWqi
foYuwLETmmasnGHO7xBjEtt.jpg","adult":false,
"overview":"A live-action adaptation of
Disney's version of the classic 'Beauty and
the Beast' tale of a cursed prince and a
beautiful young woman who helps him break
the spell.","release_date":"2017-03-
16","genre_ids":[14,10749],"id":321612,"ori
ginal_title":"Beauty and the
Beast","original_language":"en","title":"Be
auty and the
Beast","backdrop_path":"\/6aUWe0GSl69wMTSWW
exsorMIvwU.jpg","popularity":114.674355,"vo
te_count":2243,"video":false,"vote_average"
:6.8},{"poster_path":"\/gaHepzSTMkGwsSKAqiB
groSCf07.jpg","adult":false,"overview":"The
Guardians must fight to keep their newfound
family together as they unravel the
mysteries of Peter Quill's true
parentage.","release_date":"2017-04-
24","genre_ids":[35,28,12,878],"id":283995,
"original_title":"Guardians of the Galaxy
Vol.
2","original_language":"en","title":"Guardi
ans of the Galaxy Vol.
2","backdrop_path":"\/8sFWWIolWPm2FQLNt9cSK
pNZJcz.jpg","popularity":105.201927,"vote_c
ount":630,"video":false,"vote_average":7.8}
```

Copy and paste that response to the jsonschema2pojo textbox. Fill in the package name and the model class name and make sure we select JSON as the source type and GSON as the annotation style. The result will look like this.


jsonschema2pojo
Star 2,729
Share 184
Tweet
Got 60 seconds to give some feedback? Complete the 2017 user survey.
Preview
Zip
Use this tool offline: Maven plugin Gradle plugin Ant task CLI Java API
© 2012-2016 Joe Littlejohn Bugs?

```
{
   "page":1,
   "results":[
      {
         "poster_path":"\/tWqifoYuwLETmmasnGHO7xBjEt
         "adult":false,
         "overview":"A live-action adaptation of Dis
         "release_date":"2017-03-16",
         "genre_ids":[
            14,
            10749
         ],
         "id":321612,
         "original_title":"Beauty and the Beast",
         "original_language":"en",
         "title":"Beauty and the Beast",
         "backdrop_path":"\/6aUWe0GSl69wMTSWWexsorMI
         "popularity":114.674355,
         "vote_count":2243,
         "video":false,
         "vote_average":6.8
      },
      {
         "poster_path":"\/gaHepzSTMkGwsSKAqiBgroSCf0
         "adult":false,
         "overview":"The Guardians must fight to kee
         "release_date":"2017-04-24",
         "genre_ids":[
            35,
            28,
            12,
            878
         ],
         "id":283995,
         "original_title":"Guardians of the Galaxy V
         "original_language":"en",
         "title":"Guardians of the Galaxy Vol. 2",
         "backdrop_path":"\/8sFWWIolWPm2FQLNt9cSKpNZ
         "popularity":105.201927,
         "vote_count":630,
         "video":false,
         "vote_average":7.8
      },
      {
```

Package: com.example

Class name: Example

Source type: JSON Schema [✓ JSON]

Annotation style: Jackson 2.x Jackson 1.x [✓ Gson] None

- ☐ Generate builder methods
- ☐ Use primitive types
- ☐ Use long integers
- ☑ Use double numbers
- ☐ Use Joda dates
- ☐ Use Commons-Lang3
- ☑ Include getters and setters
- ☐ Include constructors
- ☐ Include hashCode and equals
- ☐ Include toString
- ☐ Include JSR-303 annotations
- ☑ Allow additional properties
- ☐ Make classes serializable
- ☐ Make classes parcelable
- ☐ Initialize collections

Property word delimiters: - _

Figure 1. jsonschema2pojo preview

After that, we need to download the file generated by the jsonschema2pojo by clicking the Zip button. The generated file should look like this:

```
package com.example;

import java.util.List;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

public class Example {

@SerializedName("page")
@Expose
private Integer page;
@SerializedName("results")
@Expose
private List<Result> results = null;
@SerializedName("total_results")
@Expose
private Integer totalResults;
@SerializedName("total_pages")
@Expose
private Integer totalPages;

public Integer getPage() {
return page;
}

public void setPage(Integer page) {
this.page = page;
}

public List<Result> getResults() {
return results;
}
```

This is a Plain Old Java Object converted from the JSON format. We need to create the interface used in this project to define the endpoint used and set of operation needed for this project, which is in this case is to get list of popular movies from the movieDB API. The interface should look like this:

```
public interface APIService {

    String baseURL =
       "https://api.themoviedb.org/3/movie/;

    @GET("popular")
    Call<Pages>
       getResultPopular(@Query("api_key")
       String api_key);

}
```

Basically, we just need a HTTP *GET* methods to get the list of popular movies and the parameter of api_key. From the above methods, the URL endpoint is https://api.themoviedb.org/3/movie/popular?api_key=<key>.

The next and the last thing we need to do is to create the Retrofit instance which will link our endpoint and convert it to POJO. The instance created should look like this:

```
String baseURL =
        "https://api.themoviedb.org/3/movie/;

Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(baseURL)
        .addConverterFactory(
        GsonConverterFactory.create())
        .build();

APIService service =
retrofit.create(APIService.class);
```

After creating the three classes needed for the retrofit to works, in our Android activity class we just need to import these packages:

```
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
```

To get the parsed response from the web services, we need to create the object corresponds to the operations which is getting list of popular movies.

```
Call<Pages> responseCall;
responseCall =
APIService.service.getResultPopular(api_ke
y);
responseCall.enqueue(new Callback<Pages>()
{

    @Override
    public void onResponse(Call<Pages>
       call, Response<Pages> response) {
         if (response.isSuccessful()) {
              Pages body = response.body();
              //do your things

         }
    }

    @Override
    public void onFailure(Call<Pages>
        call, Throwable t) {
         //handle the exception if the
              request is failed
         Log.e("Error", t.toString());
    }
});
```

This process of sending the network request to the REST web services is asynchronous, so it won't disturb the application's main thread which is the UI thread. The result of this is the process won't make the UI operation being frozen or paused. After getting the response, we will need to handle the things needed if our request is successful. We also need to handle if the request has failed. The good thing is we don't need to manually check each request is successful or not, it has been determined automatically by the framework.

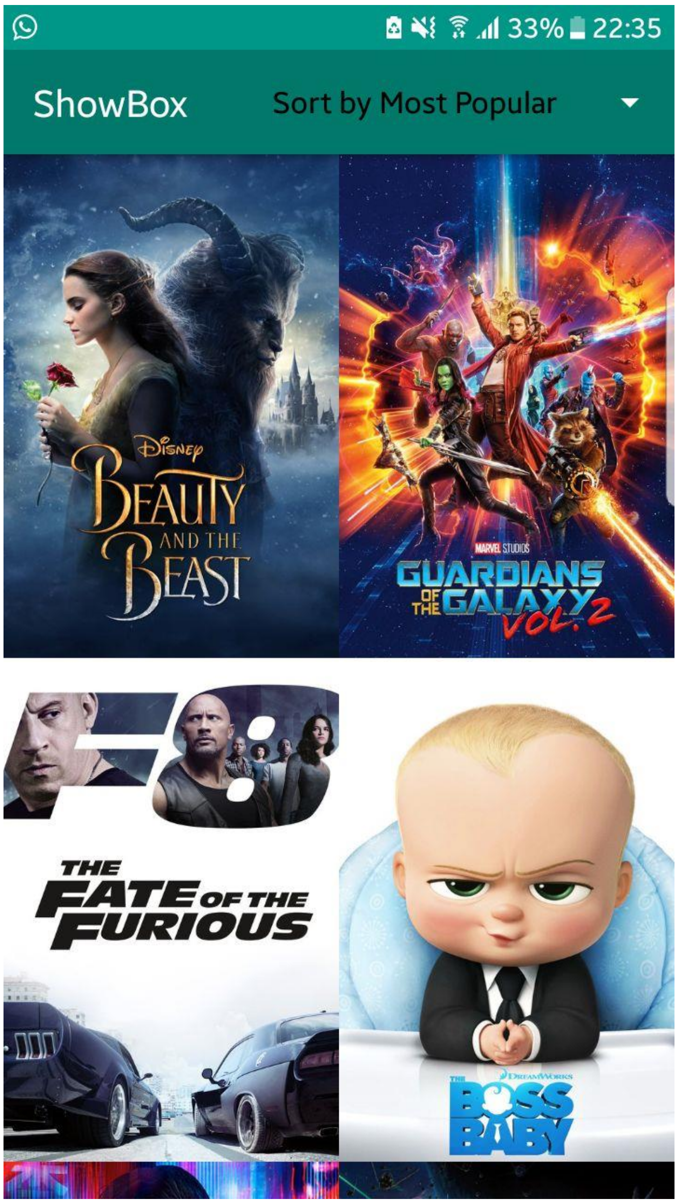

33% 22:35
ShowBox
Sort by Most Popular
Disney BEAUTY AND THE BEAST
MARVEL STUDIOS GUARDIANS OF THE GALAXY VOL. 2
F8 THE FATE OF THE FURIOUS
DreamWorks THE BOSS BABY

Figure 2. Author's movie database application preview

This is the author's Android apps after developing it using Retrofit as the REST client for connecting to the movie database API.

IV. Conclusion

Retrofit really provides easy, simple, and flexible framework for fast development in REST client for Android. The framework also provides a straightforward method to accessing resources in web services without any hassle. When using this framework compared to traditional techniques which is using *httpurlconnection* and *asynctask*, Retrofit provides better user experience with fast network connection while still maintaining responsive user interface. In a benchmark concluded by INSTRUCTURE (2013), Retrofit provides faster response hitting 50% to 90% faster comparing to asynctask method. Retrofit also have authentication features for web services implementing authentication for the requests. Using retrofit really create a stable, fast, and great scalability in developing REST client in Android application.

Acknowledgment

I would like to thank to God for letting me to finish this paper without any problems. The author would also thank to everybody involved in the making of this paper. The last and foremost, the author also really appreciate and grateful to all the IF3280 course lecturer, Dr. Ir. Rinaldi Munir, MT, Dr. Eng. Ayu Purwarianti, ST., MT, and Dessi Puji Lestari ST,M.Eng.,Ph.D. for all the lesson and guidance throughout this great semester.

References

[1] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (Doctoral dissertation, University of California, Irvine). Retrieved from https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[2] Haas, H., & Brown, A. (2004). *W3C Working Group Note 11 February 2004*. Retrieved from https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211

[3] Codepath (2017, March 12). Consuming APIs with Retrofit. Retrieved from https://guides.codepath.com/android/Consuming-APIs-with-Retrofit

[4] Instructure Tech (2013, December 9). Android Async HTTP Clients: Volley vs Retrofit. Retrieved from http://instructure.github.io/blog/2013/12/09/volley-vs-retrofit/

Statement

I hereby declare that this paper is my own work not a copy, translation, no plagiarism of somebody else's work.

Bandung, May 5th 2017

[illegible]

Albert Logianto
13514046