CS425 - Web and Mobile Software Engineering
Dr. Lilia Sfaxi
1 mai 2017

MedTech

Web Services

Android Lab 05

I. Retrofit

Retrofit[1] is a REST client for Android and Java, enabling the grouping and organisation of all the APIs in a simple and clean way.

Annotations are used to control the parameters or the API, defined in interfaces, called from classes.

All the http APIs are represented as Interfaces, as follows:

```
public interface GitHubService {
  @GET("users/{user}/repos")
  Call<List<Repo>> listRepos(@Path("user") String user);
}
```

1. APIs Declaration

Each method must have an http annotation that provides the method and the URL. Five annotations are provided: GET, POST, PUT, DELETE and HEAD. Here is an example:

```
@GET("users/list")
```

It is possible to specify the parameters of the requesting the URL:

```
@GET("users/list?sort=desc")
```

For a dynamic replacement of blocks and parameters in the method, use {}, then reference the block with the annotation @PATH, as follows:

```
@GET("group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId);
```

It is also possible to use the query parameters as follows:

[1] Retrofit : http://square.github.io/retrofit/

```
@GET("group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId, @Query("sort")
String sort);
```

The resulting URL will be as follows:

```
@GET("group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId, @QueryMap Map<
String, String> options);
```

group/id_val/users ?sort=sort_val

To insert several parameters in the request, use a Map:

Go to the Retrofit website for more examples.

2. Data Format

By default, Retrofit can only deserialise HTTP bodies as ResponseBody objects of the library OkHttp, and accepts only request of type RequestBody for the @Body.

But several converters exist to support other types, such as Gson. Gson is a Java library used to convert Java objects into their JSON representation, and vice-versa.

To support Retrofit, you should first add it to the Gradle dependencies of your project. In Retrofit 2.x, OkHttp is included by default with retrofit, you don't need to explicitly import it.

Activity 1: Create a project called Lab5. Add the following dependencies, by inserting the following line in the file *build.gradle (Module:app)* in the dependencies part:

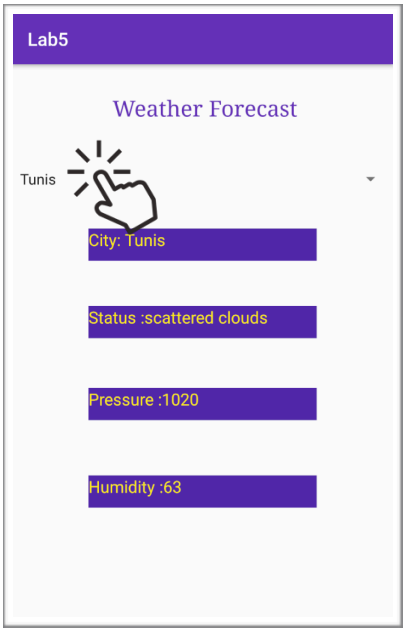
```
compile 'com.squareup.retrofit2:retrofit:2.1.0'
compile 'com.google.code.gson:gson:2.6.2'
compile 'com.squareup.retrofit2:converter-gson:2.1.0'
```

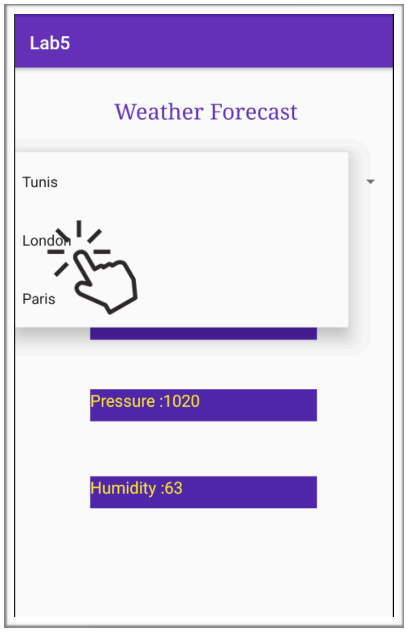
Don't forget to synchronise your project for the dependencies to be taken into account.

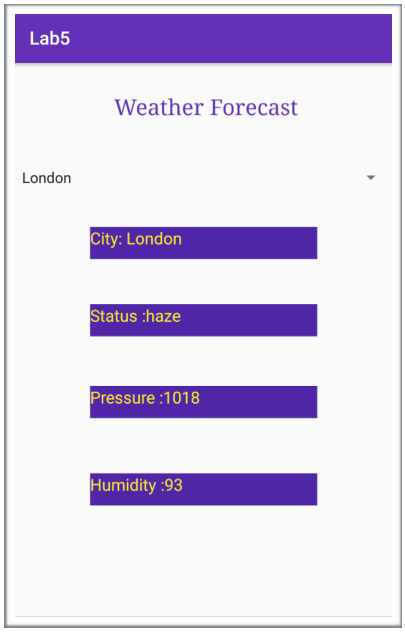
III. Exercise : Access to a Web Service

1. Objective

The main objective of this part is to create a simple application that accesses data coming from an external web service.







2. Generation of POJO classes from JSON data

The web service that we call is a public weather service. To see the current weather in Tunis, use the following URL, for example:

```
http://api.openweathermap.org/data/2.5/weather?
    q=Tunis&APPID=17db59488cadcad345211c36304a9266
```

The obtained response is the following:

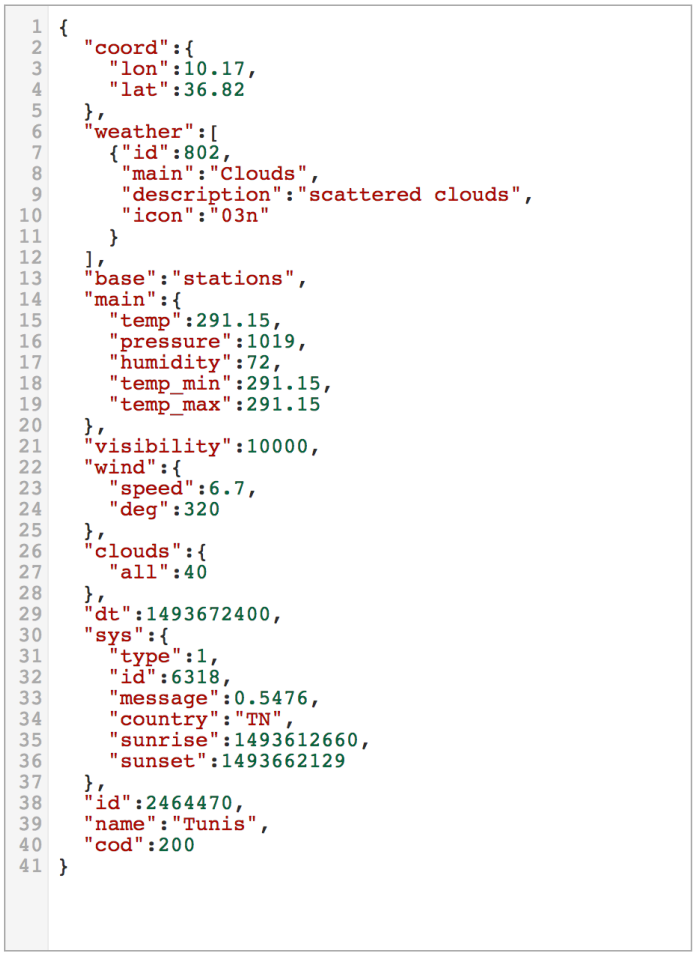
```
{
  - coord: {
        lon: 10.17,
        lat: 36.82
    },
  - weather: [
    - {
          id: 802,
          main: "Clouds",
          description: "scattered clouds",
          icon: "03n"
      }
    ],
    base: "stations",
  - main: {
        temp: 291.15,
        pressure: 1019,
        humidity: 72,
        temp_min: 291.15,
        temp_max: 291.15
    },
    visibility: 10000,
  - wind: {
        speed: 6.7,
        deg: 320
    },
  - clouds: {
        all: 40
    },
    dt: 1493672400,
  - sys: {
        type: 1,
        id: 6318,
        message: 0.5476,
        country: "TN",
        sunrise: 1493612660,
        sunset: 1493662129
    },
    id: 2464470,
    name: "Tunis",
    cod: 200
}
```

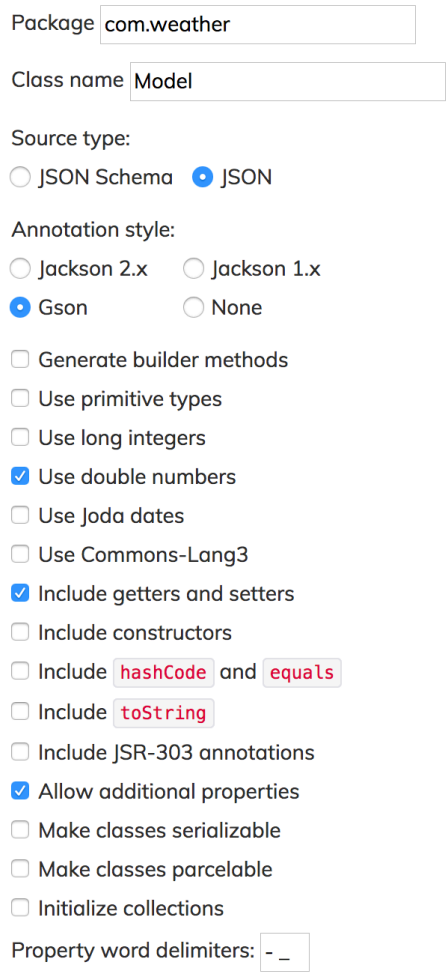

To obtain a valid APPID, you have to sign up to the site: http://openweathermap.org, an APPID is generated for you, you will find it in your profile. You can then just replace the APPID of the previous request with yours.

Once a valid response obtained, as a JSON document, it is possible to generate POJO classes that you are going to use to handle the data of the web service easily.

To do that:

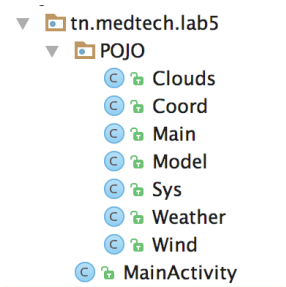
1. Go to: http://www.jsonschema2pojo.org/
2. Paste the JSON document you just obtained in the left area.
3. In the right area, write the name of the package (for example com.weather), the name of the main class (for example Model), use JSON as a source type and Gson as an annotation style, like the following:





4. Click on Zip to download the project containing the generated classes.

5. Copy the generated classes in a new POJO package of your source directory:



3. Layout Creation

Start by creating a layout with 4 Textfields, called respectively txt_city, txt_status, txt_humidity and txt_press.

Also add in the Manifest file, a permission for internet access.

```
<uses-permission android:name=« android.permission.INTERNET"/>
```

4. RestInterface

1. Create an interface classed *RestInterface* in your project. This interface will map a given URL to a method call, thanks to annotations of Retrofit API.

2. To obtain the weather of Tunis, the code of RestInterface will be as follows:

```
public interface RestInterface {

    @GET("/weather?q=Tunis&APPID=17db59488cadcad345211c36304a9266")
    void getWeatherReport(Callback<Model> cb);
}
```

Notice that the URL starts with /weather. In fact, the main path will be defined in the main class, when the service is called.

5. Retrofit Builder

In the same RestInterface, add a Retrofit Builder to build your request:

```
public static Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(url)
        .addConverterFactory(GsonConverterFactory.create())
        .build();
```

The url used here is a static String attribute representing the main URL of the service:

http://api.openweathermap.org/data/2.5/

6. Build the Activity

To call the service, follow these steps:

1. Create the appropriate textviews

2. Create an object *RestInterface* :

```
RestInterface restInterface = RestInterface.retrofit.create(RestInterface.class);
```

3. Call the service:

```
Call<Model> call = restInterface.getWeatherReport();
```

4. Schedule the call to happen asynchronously and provide the callback to be executed upon completion.

```
call.enqueue(new Callback<Model>() {
    @Override
    public void onResponse(Call<Model> call, Response<Model> response) {
        Model report = response.body();
        city.setText("City: " + report.getName());
        status.setText("Status :" + report.getWeather().get(0).getDescription());
        humidity.setText("Humidity :" + report.getMain().getHumidity().toString());
        pressure.setText("Pressure :" + report.getMain().getPressure().toString());

    }

    @Override
    public void onFailure(Call<Model> call, Throwable t) {

    }

});
```

Activity 2: Test the previous steps and verify that the textviews are loaded with valid information from the web service.

7. Adding Parameters

We want the call to the web service to be parametrized, which means that the city name will be provided dynamically by the user of the application.

Activity 3: Add a parameter to the *getWeatherReport* function. Then add a spinner to your layout, and make sure the weather information changes according to the selected name of the city in the spinner.