Project Week 1

November 2018

1 Introduction

In the GitHub page, we mentioned what will be the project like. In brief, it will demonstrate something about the context of the application. The app will also allow users to archive these things and make conversation about it.

The tasks of the first week were:

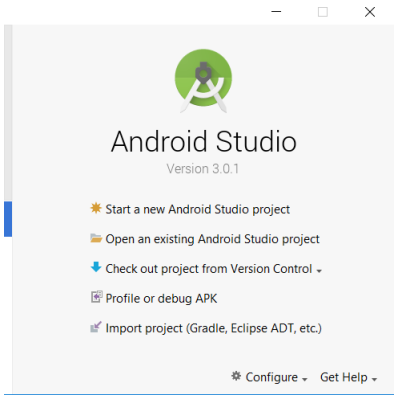
1. Find a topic that you have interested in.
2. Create a repository on GitHub.
3. Create a project from Android Studio by using clone feature.
4. Implement Http get method to get a response from API.
5. Parse the response using Gson our your own custom parser functions.

2 Conquering Tasks

Firstly we have to choose our topic. The IMDB company has a wonderful API called OMDB.
Almost everyone likes watching films and everyone wants to watch cool films. Therefore, we thought that showing films to user will be magnificent for them. If you will use the same API, you will need an apikey. You can retrieve it by signing to the OMDB. After that it will asks you that key every time you send a response.
Take a look at the URLs and responses of your API and analyze them carefully.

2.1 Creating Android Studio Project

After we choose our topic and understood the structure of the responses, it is time for creating the project. We assume that you have already installed Android Studio. Thus, we will continue from creating the project.



Click the *Start a new Android Studio project*. After that it will ask the project name and the path. After entering these values create your project by clicking next and finish.

2.2 Implementing the Http Get method

Our API returns responses for the GET method only. Therefore we will implement only this method. But remember than you can always check the Internet for other methods.
In the Android Environment, the main thread should not be waited for too long. Therefore, we have to create an asynctask class to implement the get method.

```
class HttGet : AsyncTask<Params, Progress, Result>() {}
```

Here is the basic definition of our HttpGet Class. It inherits from AsyncTask class provided by the Android Environment.
We have to define these types: Parameter, Progress and Result.
We will make an Internet request so our param will be a string. We will not use progress part so we can pass this as void. Lastly, the API will return the response as string. Therefore, our last parameter (result) will be a string.

After defining these parameters, we should add two methods: doInBackGround and onPostExecute. In the doInBackround method we will open a connection and make a request.
IonPostExecute will receive the response which is returned from doInBackground and pass it to MainActivity.

After adding these methods, the skeleton of the class will be like:

```
class HttGet : AsyncTask<String, Void, String>() {
    override fun doInBackground(vararg p0: String?): String? {
        var response:String? = null
```

```
        return response
    }

    override fun onPostExecute(result: String?) {

    }
}
```

Now it is time to create a connection. First of all we need to create a URL to connect. The OMDB API gives the base URL as:

$http://www.omdbapi.com/?apikey=<yourapikey>$.

There will be some parameters which will be the same for all http requests. Therefore we will define them as a class' variable:

```
class HttGet(val internetInterface: InternetInterface) : AsyncTask<String, Void,
    // Put your api key below with replacing <yourKey>
    private var baseUrl ="http://www.omdbapi.com/?apikey=<yourKey>"
    private val readTimeOut = 2000
    private val connectTimeOut = 2000
    private val requestMethod = "GET"

    override fun doInBackground(vararg p0: String?): String? {
```

To shape the URL to its last form we will concatenate an id of a film with the base url to connect. The last shape will be something like:

$http://www.omdbapi.com/?apikey=<yourapikey>\&i=<imdbID>$.

The imdbID is unique for all films and will be taken as input for HttpGet class. Here is the final version to create the url:

```
override fun doInBackground(vararg p0: String?): String? {
    var response:String? = null
    val urlStr = baseUrl + "&i=" + p0[0]
    val url = URL( urlStr )
```

After that, we need to create the connection:

```
    val url = URL( urlStr )
    val connection = url.openConnection() as HttpURLConnection
```

```
    connection.readTimeout = readTimeOut
    connection.connectTimeout = connectTimeOut
    connection.requestMethod = requestMethod

    connection.connect()
```

After the connection is created, it try to connect to the *url* with the given *connection.requestMethod* in *connection.connectTimeout* milliseconds. Later, it will try to read the response in *connection.readTimeOut* milliseconds.

It will either read successfully or something will fail. Let us analyze these possibilities.
The first possibility is making a successful connection. If that is the case we will read the response line by line from the buffer which filled with connection's inputStream.

```
// Connection successful
    if (connection.responseCode == HttpURLConnection.HTTP_OK){
        // Read the result
        val streamReader = InputStreamReader(connection.inputStream)
        val bufferedReader = BufferedReader(streamReader)
        val stringBuilder = StringBuilder()
        var line:String? = bufferedReader.readLine()
        while(line != null){ // Create result from buffer
            stringBuilder.append(line).append('\n')
            line = bufferedReader.readLine()
        }
        bufferedReader.close()
        streamReader.close()
        response = stringBuilder.toString()
    }
```

The other possibility is failure. In this case, we will read the error from connection's errorStream but we will not show that to users. We will only log it.

```
    else{
        var errorStr = ""
        // Read error message but only print it to log.
        val streamReader = InputStreamReader(connection.errorStream)
        val bufferedReader = BufferedReader(streamReader)
        val stringBuilder = StringBuilder()
        var line:String? = bufferedReader.readLine()
        while(line != null){ // Create result from buffer
            stringBuilder.append(line).append('\n')
```

```
            line = bufferedReader.readLine()
        }
        bufferedReader.close()
        streamReader.close()
        errorStr = stringBuilder.toString()
        Log.e("Internet Error:",errorStr)
    }
```

After these operations finished, we will disconnect from the connection and return the result.

```
    connection.disconnect()
    return response
```

If you analyze the code, the response variable initialized as null. It will change only if the connection successes.
Thus the response object will be null in *onPostExecute* function if the connection failed.
We will go deeper with the *onPostExecute* function but we need to talk about very useful thing.

2.2.1 Creating an Interface

When the HttpGet class which is an asynctask at same time, finishes it operation, how it will return the response to the MainActivty?
Since it is an asynchronous call, we could not make a regular assignment.
Therefore, we need an additional structure called interfaces.
The main idea of the interfaces is something(Internet operation for our case) will happen in a class (HttpGet in our project) but the implementation will be done by another class (MainActivity in our case).
You can also think this like informing the MainActivity about the Internet call and its result.

First of all we will create an Interface file:

```
interface InternetInterface{
    fun onSuccess(result:String)
    fun onError()
}
```

It has 2 methods. One for successful calls in which we will parse the response and another for failed calls.
To make function calls from of the Interface we need to create an object of it. Also, these function will be invoked by HttpGet class. Therefore, this object needs to be in HttpGet class.
We will change its constructor to:

```
class HttGet(val internetInterface: InternetInterface) : AsyncTask<String, Void,
```

After our interface is created it is time to finish the *onPostExecute* method. It will invoke *onError* method of the interface if the response is null (connection failed) and it will invoke *onSuccess* method with response object to pass the result to the MainActivity if connection was successful.

```
override fun onPostExecute(response: String?) {
    if(response != null){
        internetInterface.onSuccess(response)
    }
    else{
        internetInterface.onError()
    }
}
```

2.2.2 Calling the API

We will call the API from *MainActivity* as soon as the application will load. But it will be a temporary case until we implement the layouts.
Before doing anything with *MainActivity* we have to get permission.
Permission of the application are written to manifest file. We will add:

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

to previous line of *application* tag After that, we will invoke the HttpGet method in *onCreate* method of the MainActivity:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    HttGet(this).execute("tt2313197")
}
```

As you can see from the code above, we are passing *this* keyword to the constructor of the HttpGet class. It is the interface object and it refers MainActivity will implement the interface methods.
The MainActivity will be like:

```
class MainActivity : AppCompatActivity(), InternetInterface {
    override fun onError() {
        Log.e("Main Act ","Something Wrong With Internet!")
    }

    override fun onSuccess(result: String) {
```

```
        Log.e("Main Act ",result)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        HttGet(this).execute("tt2313197")
    }
}
```

Also, parameter of the execute method is one of the film id's which is used to create final url in *doInBackground* method.

2.3 Parsing The JSON

If you implemented everything, you should see an JSON response when you open the app.
It is a huge progress yet not enough. We should convert this JSON response to an object so that we can use it later.
We will create an object class to handle these operations. It will provide us to write static method which can be called from every class without creating an object.
If use a web browser to get a response we will see something like in the Figure 1


```
{"Title":"The Lego Batman Movie","Year":"2017","Rated":"PG","Released":"10 Feb 2017","Run
(screenplay by), Chris McKenna (screenplay by), Erik Sommers (screenplay by), Jared Stern
Finger (Batman created by), Jerry Siegel (Superman created by), Joe Shuster (Superman cre
by), Godtfred Kirk Christiansen (based on LEGO Construction Toys created by), Jens Nygaar
Fiennes","Plot":"A cooler-than-ever Bruce Wayne must deal with the usual suspects as they
sidekick.","Language":"English","Country":"USA, Denmark","Awards":"11 wins & 58 nominatio
amazon.com/images/M/MV5BMTcyNTEyOTY0M15BMl5BanBnXkFtZTgwOTAyNzU3MDI@._V1_SX300.jpg","Rati
{"Source":"Metacritic","Value":"75/100"}],"Metascore":"75","imdbRating":"7.3","imdbVotes"
Pictures","Website":"http://www.legobatman.com/","Response":"True"}
```


Figure 1: JSON response

As you can see it is very hard to understand the structure of the response. But do not worry, we can use additional tools such as JSON formatter.
An example of structured response will be like Figure 2:

Figure 2: Formatted JSON

2.3.1 Structure of the JSON

JSON (JavaScript Object Notation) is a notation to indicates objects. It has 2 main attributes: *keys and* values.
Mostly keys will be a string. Key indicates the name of the attribute of an object.
For an example: if we say that the whole response in the Figure 2 is a Film object. The **Title**, **Year**, **Rated**, **Director**, **Actors** , **Ratings** , etc. will be the attributes of that Film object.
In the other hand the *value* attribute of the JSON structure could be anything. It can be an **int**, a **char**, a **string** and even an **object**.
Below there are some JSON examples.
In the Figure 3 it is clear that, the **Title** is an attribute of the Film Object and its value will be a string.

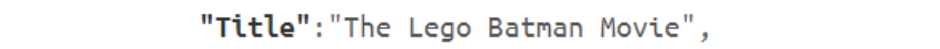


Figure 3: Caption

In the Figure 4 it is clear that, the **Title** is an attribute of the Film Object and its value will be a string.

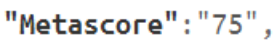


Figure 4: Caption

In the Figure 4 A **Metascore** is a attribute and its value will be an integer or a float.
These two was easy, but how will we understand if the value is an object?
If you look clearly, you will see that an object in JSON format starts with, '{' and ends with '}'.
Also, an array starts with '[' and ends with ']'. Thus, in the Figure 5 there is an array called **Ratings** which keeps objects that have **Source** and **Value** attributes.

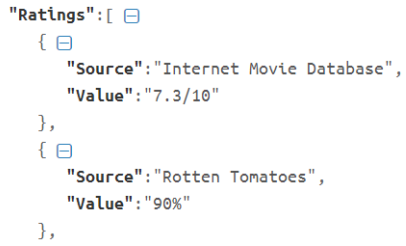


Figure 5: Caption

2.3.2 Creating Objects

We briefly talked about the object notations in the JSON format.
It is now time to create our own object which will be created from JSON.
The OMDB API returns a lots of attributes within an Film object.
We will show brief information about the films in the application. Therefore, there is no need to parse whole object.
We will take:

- Title
- Year
- Plot
- Runtime
- imdbID
- Ratings

attributes to create our object.
We created our classes in Android Studio as:

```
class Ratings(var source:String,
              var value:String)
{
    constructor():this( source: "", value: "")
}
```

Figure 6: Caption

```
class Film(var title:String,
           var year:Int,
           var plot:String,
           var runtime: String,
           var imdbID:String,
           var ratings:MutableList<Ratings>)
{
    constructor():this( title: "", year: 0, plot: "", runtime: "", imdbID: "", mutableListOf())
}
```

Figure 7: Caption

We also written a constructor without argument because while parsing the JSON, first we will create an empty object. Later we will fill the attributes with parsed values.
To create an empty object, we need a constructor without argument.

2.3.3 Parsing JSON

While parsing the JSON, we are not completely alone.
We can use these functions:

- **JSONObject(response:String)**: Constructor to create a JSONObject.
- **has(key:String)**: Checks wheter given key exists in JSONObject or not.
- **get(key:String)**: Fetches the value of the key given as argument.
- **getJSONArray(key:String)**: Returns an array which have JSONObjects in it. It will return the value whose key is given by argument.
- **getJSONObject(index:Int)**: Return the JSON object given by index from the JSONObjectArray.

As an example, acquiring the title will be like:

```
if (responseJSON.has("Title"))
    film.title = responseJSON.getString("Title")
```

If the value of the key was not a string:

```
if (responseJSON.has("Year"))
    film.year = (responseJSON.get("Year") as String).toInt()
```

Using **.has** is not a must. However, we are not completely sure about the response coming out from the server. If we use only **get** method without checking the existence of that key with **has** method, the may crash if the response

we are parsing has not a key.
In below there is a code to parse **Ratings** objects. First, it checks that **Ratings** key is exist. If yes, it creates an JSON object from it.
Later, it parses the rating object and add it to the ratings array of the film object.

```
if(responseJSON.has("Ratings")) {
        val ratingsJSON = responseJSON.getJSONArray("Ratings")
        for (i in 0 until ratingsJSON.length()){
            val ratingJSON = ratingsJSON.getJSONObject(i)
            val rating = Ratings()
            if (ratingJSON.has("Source"))
                rating.source = ratingJSON.getString("source")
            if (ratingJSON.has("Value"))
                rating.value = ratingJSON.getString("value")
            film.ratings.add(rating)
        }
    }
```

Here is the complete parsing below. After you complete this, try to send the response in onSuccess method and try to get an object as return value.

```
object JSONParser {
    fun retObjectFromJSON(response:String):Film{
        // constructor without arguments
        val film = Film()
        val responseJSON = JSONObject(response)
        //Check for existence of Title key.
        if (responseJSON.has("Title"))
            film.title = responseJSON.getString("Title")

        if (responseJSON.has("Year"))
            film.year = (responseJSON.get("Year") as String).toInt()

        if (responseJSON.has("Plot"))
            film.plot = responseJSON.getString("Plot")

        if (responseJSON.has("imdbID"))
            film.imdbID = responseJSON.getString("imdbID")

        if(responseJSON.has("Ratings")) {
            val ratingsJSON = responseJSON.getJSONArray("Ratings")
            for (i in 0 until ratingsJSON.length()){
                // Get JSON object given by index
                val ratingJSON = ratingsJSON.getJSONObject(i)
                // constructor without arguments
```

```
                val rating = Ratings()
                if (ratingJSON.has("Source"))
                    rating.source = ratingJSON.getString("source")
                if (ratingJSON.has("Value"))
                    rating.value = ratingJSON.getString("value")
                film.ratings.add(rating)
            }
        }
        return film
    }
}
```

That is all for that week.