Using Retrofit 2.x as REST client - Tutorial

Lars Vogel, Simon Scholz, David Weiser (c) 2012, 2017 vogella GmbH
Version 2.1,06.02.2017

Table of Contents

This tutorial explains the usage of the Retrofit library as REST client.

1. Retrofit

1.1. What is Retrofit

Retrofit is a REST Client for Android and Java by Square. It makes it relatively easy to retrieve and upload JSON (or other structured data) via a REST based webservice. In Retrofit you configure which converter is used for the data serialization. Typically for JSON you use GSon, but you can add custom converters to process XML or other protocols. Retrofit uses the OkHttp library for HTTP requests.

> You can generate Java objects based on JSON via the following URL: http://www.jsonschema2pojo.org/ This can be useful to create complex Java data structures from existing JSON.

1.2. Using Retrofit

To work with Retrofit you need basically three classes.

- Model class which is used to map the JSON data to
- Interfaces which defines the possible HTTP operations
- Retrofit.Builder class - Instance which uses the interface and the Builder API which allows defining the URL end point for the HTTP operation.

Every method of an interface represents one possible API call. It must have a HTTP annotation (`GET`, `POST`, etc.) to specify the request type and the relative URL. The return value wraps the response in a *Call* object with the type of the expected result.

```
@GET("users")
Call<List<User>> getUsers()
```

You can use replacement blocks and query parameters to adjust the URL. A replacement block is added to the relative URL with `{}`. With the help of the `@Path` annotation on the method parameter, the value of that parameter is bound to the specific replacement block.

```
@GET("users/{name}/commits")
Call<List<Commit>> getCommitsByName(@Path("name") String name)
```

Query parameters are added with the `@Query` annotation on a method parameter. They are automatically added at the end of the URL.

```
@GET("users")
Call<User> getUserById(@Query("id") Integer id)
```

The `@Body` annotation on a method parameter tells Retrofit to use the object as the request body for the call.

```
@POST("users")
Call<User> postUser(@Body User user)
```

2. Retrofit converters and adapters

2.1. Retrofit Converters

Retrofit can be configured to use a specific converter. This converter handles the data (de)serialization. Several converters are already available for various serialization formats.

- To convert to and from JSON:
  - Gson: com.squareup.retrofit:converter-gson

    - Jackson: com.squareup.retrofit:converter-jackson
    - Moshi: com.squareup.retrofit:converter-moshi
- To convert to and from Protocol Buffers:
    - Protobuf: com.squareup.retrofit:converter-protobuf
    - Wire: com.squareup.retrofit:converter-wire
- To convert to and from XML:
    - Simple XML: com.squareup.retrofit:converter-simplexml

Besides the listed converters, you can also create custom converters to process other protocols by subclassing the *Converter.Factory class*.

2.2. Retrofit Adapters

Retrofit can also be extended by adapters to get involved with other libraries like RxJava 2.x, Java 8 and Guava.

An overview for available adapters can be found on Github square/retrofit/retrofit-adapters/.

For example the RxJava 2.x adapter can be obtained by using Gradle:

```
compile 'com.squareup.retrofit2:adapter-rxjava2:latest.version'
```

or using Apache Maven:

```
<dependency>
  <groupId>com.squareup.retrofit2</groupId>
  <artifactId>adapter-rxjava2</artifactId>
  <version>latest.version</version>
</dependency>
```

In order to add an adapter the `retrofit2.Retrofit.Builder.addCallAdapterFactory(Factory)` method has to be used.

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.example.com")
    .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
    .build();
```

With this adapter being applied the Retrofit interfaces are able to return RxJava 2.x types, e.g., Observable, Flowable or Single and so on.

```
@GET("users")
Observable<List<User>> getUsers();
```

3. Retrofit authentication

Retrofit supports you with API calls that need authentication. Authentication can be done by using a username and a password (*Http*

Basic authentication) or an API token.

There are two ways, how you can handle the authentication. The first method would be to manipulate the header for the request with the help of annotations. Another possibility would be to use an OkHttp interceptor for that.

3.1. Authentication with annotations

Assume, that you want to request your user details, which requires you to authenticate. You can do this by adding a new parameter to your API definition, like the following:

```
@GET("user")
Call<UserDetails> getUserDetails(@Header("Authorization")
String credentials)
```

With the help of the `@Header("Authorization")` annotation you tell Retrofit to add the *Authorization* field to the request header with the value you provide for *credentials*.

To generate Basic authentication credentials, you can use OkHttps *Credentials* class with its *basic(String, String)*method. The method takes the username and the password and returns the authentication credential for the Basic scheme.

```
Credentials.basic("ausername","apassword");
```

If you want to use an API token and no Basic authentication, just call the `getUserDetails(String)` method with your token instead.

3.2. Authentication with OkHttp interceptors

The above method only adds the credentials, if you request your user details. If you have more calls that require you to authenticate, you can use an interceptor for this. An interceptor is used to modify each request before it is performed and alters the request header. The advantage is,

that you don't have to add `@Header("Authorization")` to each API method definition.

To add an interceptor, you have to use the `okhttp3.OkHttpClient.Builder.addInterceptor(Interceptor)` method on the OkHttp Builder.

```
OkHttpClient okHttpClient = new
OkHttpClient().newBuilder().addInterceptor(new Interceptor() {
                    @Override
                    public okhttp3.Response intercept(Chain
chain) throws IOException {
                            Request originalRequest =
chain.request();

                            Request.Builder builder =
originalRequest.newBuilder().header("Authorization",
Credentials.basic("aUsername", "aPassword"));

                                Request newRequest =
builder.build();
                                return
chain.proceed(newRequest);
                        }
                }).build();
```

The created OkHttp client has to be added to your Retrofit client with the `retrofit2.Retrofit.Builder.client(OkHttpClient)` method.

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.example.com")
    .client(okHttpClient)
    .build();
```

You may again noticed the usage of the Credentials class for Basic authentication. Again, if you want to use a API token, just use the token instead.

4. Exercise: Using Retrofit to query Gerrit in Java

The following section describes how to create a minimal Java application which uses Retrofit to query the subjects of open changes from the Gerrit API. The results are printed on the console.

4.1. Project creation and setup

This exercise assumes that you are familiar with Gradle and Buildship for Eclipse.

Create a new Gradle project with the name *com.vogella.java.retrofitgerrit*. Add a new package to *src/main/java* with the name *com.vogella.java.retrofitgerrit*.

Add the following dependencies to your build.gradle file.

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

4.2. Define the API and the Retrofit adapter

In the JSON reply from Gerrit we are only interested in the subject of the changes. Therefore create the following data classes in the previously added default package.

```
package com.vogella.java.retrofitgerrit;

public class Change {
        String subject;

        public String getSubject() {
                return subject;
        }

        public void setSubject(String subject) {
                this.subject = subject;
        }
}
```

Define the REST API for Retrofit via the following interface.

```
package com.vogella.java.retrofitgerrit;

import java.util.List;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Query;

public interface GerritAPI {

        @GET("changes/")
        Call<List<Change>> loadChanges(@Query("q") String
status);
}
```

Create the following controller class. This class creates the Retrofit client, calls the Gerrit API and handles the result (prints the result of the call on the console).

```
package com.vogella.java.retrofitgerrit;

import java.util.List;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class Controller implements Callback<List<Change>> {

        static final String BASE_URL =
"https://git.eclipse.org/r/";

        public void start() {
                Gson gson = new GsonBuilder()
                        .setLenient()
                        .create();

                Retrofit retrofit = new Retrofit.Builder()
                                .baseUrl(BASE_URL)

.addConverterFactory(GsonConverterFactory.create(gson))
                                .build();

                GerritAPI gerritAPI =
                GerritAPI gerritAPI =
retrofit.create(GerritAPI.class);

                Call<List<Change>> call =
gerritAPI.loadChanges("status:open");
                call.enqueue(this);

        }

        @Override
        public void onResponse(Call<List<Change>> call,
Response<List<Change>> response) {
                if(response.isSuccessful()) {
                        List<Change> changesList =
response.body();
                        changesList.forEach(change ->
System.out.println(change.subject));
                } else {

System.out.println(response.errorBody());
                }
        }

        @Override
        public void onFailure(Call<List<Change>> call,
Throwable t) {
                t.printStackTrace();
        }
}
```

Create a class with a *main*-method to start the controller.

```
package com.vogella.java.retrofitgerrit;

public class Main {

        public static void main(String[] args) {
                Controller controller = new Controller();
                controller.start();
        }
}
```

5. Exercise: Using Retrofit to convert XML response from an RSS feed

This section describes the usage of Retrofit to convert a XML response with the help of the *Simple XML Converter*.

A minimal Java application is created that requests the Vogella RSS feed (http://vogella.com/article.rss) and prints out the channel title and the titles and the links of the articles.

5.1. Project creation and setup

This exercise assumes that you are familiar with Gradle and Buildship for Eclipse.

Create a new Gradle project with the name *com.vogella.java.retrofitxml*. Add a new package to *src/main/java* with the name *com.vogella.java.retrofitxml*.

Add the following dependencies to your build.gradle file.

```
compile 'com.squareup.retrofit2:retrofit:2.1.0'
compile 'com.squareup.retrofit2:converter-simplexml:2.1.0'
```

5.2. Define the XML mapping

An RSS feed looks like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
<channel>
        <title>Eclipse and Android Information</title>
        <link>http://www.vogella.com</link>
        <description>Eclipse and Android
Information</description>
        <language>en</language>
        <copyright>Creative Commons Attribution-NonCommercial-
ShareAlike 3.0 Germany (CC BY-NC-SA 3.0)</copyright>
        <pubDate>Tue, 03 May 2016 11:46:11 +0200</pubDate>
<item>
        <title>Android user interface testing with Espresso -
Tutorial</title>
        <description> This tutorial describes how to test
Android applications with the Android Espresso testing
framework.</description>

<link>http://www.vogella.com/tutorials/AndroidTestingEspresso/a
rticle.html</link>
        <author>lars.vogel@vogella.com (Lars Vogel)</author>

<guid>http://www.vogella.com/tutorials/AndroidTestingEspresso/a
rticle.html</guid>
</item>
<item>
        <title>Using the Gradle build system in the Eclipse IDE
- Tutorial</title>
        <description>This article describes how to use the
Gradle tooling in Eclipse</description>

<link>http://www.vogella.com/tutorials/EclipseGradle/article.ht
ml</link>
        <author>lars.vogel@vogella.com (Lars Vogel)</author>

<guid>http://www.vogella.com/tutorials/EclipseGradle/article.ht
ml</guid>
</item>
<item>
        <title>Unit tests with Mockito - Tutorial</title>
        <description>This tutorial explains testing with the
Mockito framework for writting software tests.</description>

<link>http://www.vogella.com/tutorials/Mockito/article.html</li
nk>
        <author>lars.vogel@vogella.com (Lars Vogel)</author>

<guid>http://www.vogella.com/tutorials/Mockito/article.html</gu
id>
</item>
</channel>
</rss>
```

Besides the XML header this file consists of various XML elements. The rss-element contains a channel-element which again contains other elements (i.e. title, description, pubDate) and several item-elements (the articles).

Create the following two data classes named *RSSFeed* and *Article*.

```
package com.vogella.java.retrofitxml;

import org.simpleframework.xml.Element;
import org.simpleframework.xml.Root;

@Root(name = "item", strict = false)
public class Article {

        @Element(name = "title")
        private String title;

        @Element(name = "link")
        private String link;

        public String getTitle() {
                return title;
        }

        public void setTitle(String title) {
                this.title = title;
        }


        public String getLink() {
                return link;
        }

        public void setLink(String link) {
                this.link = link;
        }
}
```

```
package com.vogella.java.retrofitxml;

import java.util.List;

import org.simpleframework.xml.Element;
import org.simpleframework.xml.ElementList;
import org.simpleframework.xml.Path;
import org.simpleframework.xml.Root;

@Root(name="rss", strict=false)
public class RSSFeed {

        @Element(name="title")
        @Path("channel")
        private String channelTitle;

        @ElementList(name="item", inline=true)
        @Path("channel")
        private List<Article> articleList;

        public String getChannelTitle() {
                return channelTitle;
        }

        public void setChannelTitle(String channelTitle) {
                this.channelTitle = channelTitle;
        }

        public List<Article> getArticleList() {
                return articleList;
        }

        public void setArticleList(List<Article> articleList) {
                this.articleList = articleList;
        }
}
```

The 'Article' class represents one single article and only stores the title and the link of it. These are the only fields we are interested in.

With the `@Root` annotation a class is marked to be (de)serialized. Optional, you can provide a name in the `@Root` annotation that represents the name of the XML element. If no name is provided, the class name is used as the XML element name. Because the class name (RSSFeed) is different to the XML element name (rss), you have to provide a name.

With *strict* set to *false*, strict parsing is disabled. This tells the parser to not fail and throw an exception, if a XML element or attribute is found for which no mapping is provided. As the rss-element has the version attribute which is not bound to a field, the application would crash, if it is not set to false.

With the help of the `@Element` annotation, a XML element is represented. Optional, you can provide a name for the XML element that is represented by that field. If no name is provided, the fields name is used.

The field *articleList* is annotated with `@ElementList` . That indicates, that this field is used to store a *Collection* (in our case: *List<Article>*) of XML elements with the same name. With *inline* set to *true* it is determined that the elements of the Collection are inlined. That means, that they have no enclosing element and are listed one after another within the XML response.

With the `@Path` annotation you can provide a path to an XML element inside the XML tree. This is helpful, if you don’t want to model the complete XML tree with Java objects. For the title of the channel and the several item-elements, we can directly point to the specific elements within the channel-element.

5.3. Define the API and the Retrofit adapter

Define the REST API for Retrofit via the following interface.

```
package com.vogella.java.retrofitxml;

import retrofit2.Call;
import retrofit2.http.GET;

public interface VogellaAPI {

        @GET("article.rss")
        Call<RSSFeed> loadRSSFeed();
}
```

Create the following controller class. This class creates the Retrofit client, calls the Vogella API and handles the result.

```
package com.vogella.java.retrofitxml;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.simplexml.SimpleXmlConverterFactory;

public class Controller implements Callback<RSSFeed> {


        static final String BASE_URL = "http://vogella.com/";

        public void start() {
                Retrofit retrofit = new
Retrofit.Builder().baseUrl(BASE_URL)

.addConverterFactory(SimpleXmlConverterFactory.create()).build(
);

                VogellaAPI vogellaAPI =
retrofit.create(VogellaAPI.class);

                Call<RSSFeed> call = vogellaAPI.loadRSSFeed();
                call.enqueue(this);
        }

        @Override
        public void onResponse(Call<RSSFeed> call,
Response<RSSFeed> response) {
                if (response.isSuccessful()) {
                        RSSFeed rss = response.body();
                        System.out.println("Channel title: " +
rss.getChannelTitle());
                        rss.getArticleList().forEach(
                                        article ->
System.out.println("Title: " + article.getTitle() + " Link: " +
article.getLink()));
                } else {

System.out.println(response.errorBody());
                }
        }

        @Override
        public void onFailure(Call<RSSFeed> call, Throwable t)
{
                t.printStackTrace();
        }
}
```

The last step is to create a class with a *main*-method to start the controller.

```
package com.vogella.java.retrofitxml;

public class Application {

        public static void main(String[] args) {
                Controller ctrl = new Controller();
                ctrl.start();
        }

}
```

6. Exercise: Using Retrofit to query Stackoverflow in Android

StackOverflow is a popular side for asking programming orientated questions. It also provides a REST API which is well documented. Queries can be build using this API. You find it documented via the following URL: https://api.stackexchange.com/docs/search In this exercise you use the Retrofit REST library to query stackoverflow for tagged questions and their answers. After authentication, you have the possibility to upvote an answer. Please make sure, that you have a Stackoverflow account in order to get this exercise fully working.

We use the following query URL in our example.

```
https://api.stackexchange.com/2.2/search?
order=desc&sort=votes&tagged=android&site=stackoverflow
```

6.1. Project creation and setup

Create an Android application called *com.vogella.android.retrofitstackoverflow*. Use *com.vogella.android.retrofitstackoverflow* as top level package name.

Add the following dependency to your build.gradle file.

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

Add the permission to access the Internet to your manifest file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.vogella.android.retrofitstackoverflow">
    <uses-permission
android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".WebViewActivity"/>
    </application>
```

```
</manifest>
```

6.2. Define the API and the Retrofit adapter

As we are interested in the questions and answers from Stackoverflow. Therefore we need a data class for each of them. Create the following two data classes named *Question* and *Answer*.

```
package com.vogella.android.retrofitstackoverflow;

import com.google.gson.annotations.SerializedName;

public class Question {

    public String title;
    public String body;

    @SerializedName("question_id")
    public String questionId;

    @Override
    public String toString() {
        return(title);
    }
}
```

```
package com.vogella.android.retrofitstackoverflow;

import com.google.gson.annotations.SerializedName;

public class Answer {

    @SerializedName("answer_id")
    public int answerId;

    @SerializedName("is_accepted")
    public boolean accepted;

    public int score;

    @Override
    public String toString() {
        return answerId + " - Score: " + score + " - Accepted:
" + (accepted ? "Yes" : "No");
    }
}
```

As you can see, we store the title, the link and the id of a question. For an answer, we store the id, the current score (the amount of upvotes) and if the answer was already accepted.

The Stackoverflow API wraps replies for questions or answers in a JSON object with the name *items* . To handle this, create the following data class named *ListWrapper*. This class accepts a type parameter and simply wraps a list of objects of that type.

> The list structure is needed, as Stackoverflow wraps its answer in an item object.

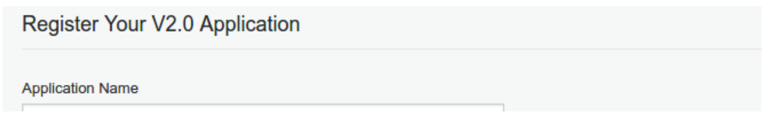
```java
package com.vogella.android.retrofitstackoverflow;

import java.util.List;

public class ListWrapper<T> {
    List<T> items;
}
```

Define the REST API for Retrofit via the following interface.

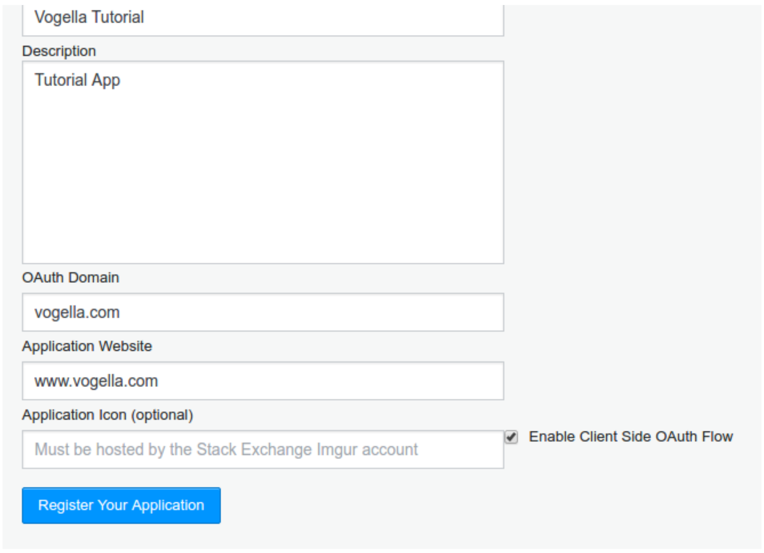
```java
package com.vogella.android.retrofitstackoverflow;

import java.util.List;

import okhttp3.ResponseBody;
import retrofit2.http.Field;
import retrofit2.http.FormUrlEncoded;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.Path;
import retrofit2.Call;

public interface StackOverflowAPI {
    String BASE_URL = "https://api.stackexchange.com";

    @GET("/2.2/questions?
order=desc&sort=votes&site=stackoverflow&tagged=android&filter=
withbody")
    Call<ListWrapper<Question>> getQuestions();

    @GET("/2.2/questions/{id}/answers?
order=desc&sort=votes&site=stackoverflow")
    Call<ListWrapper<Answer>> getAnswersForQuestion(@Path("id")
String questionId);

    @FormUrlEncoded
    @POST("/2.2/answers/{id}/upvote")
    Call<ResponseBody> postUpvoteOnAnswer(@Path("id") int
answerId, @Field("access_token") String accessToken,
@Field("key") String key, @Field("site") String site,
@Field("preview") boolean preview, @Field("filter") String
filter);
}
```

6.3. Register Application

As suggested by Stackoverflow, we use the implicit OAuth 2.0 flow here because we develop a client side application. First, head over to Stackapps and register a new application for your account. Fill the mask like the following and make sure, that you check the *Enable Client Side OAuth Flow* option.


Register Your V2.0 Application

Application Name


Vogella Tutorial
Description
Tutorial App
OAuth Domain
vogella.com
Application Website
www.vogella.com
Application Icon (optional)
Must be hosted by the Stack Exchange Imgur account
✓ Enable Client Side OAuth Flow

After you registered your application, you get a *Client Id* and a *Key* . As we want to keep them out of a possible version control system, we import them into our project using gradle. To do so, go to your gradle home directory (*.gradle/* in your users home directory) and paste the following lines into your *gradle.properties* file (create one, if you can't find it). Of course, replace *yourKey* and *yourClientId* with your corresponding values from Stackapps.

```
key=yourKey
client_id=yourClientId
```

To later make these two values accessible in your project like normal String resources, add the following two lines to the *defaultConfig* in your build.gradle.

```
resValue("string", "key", project.key)
resValue("string", "client_id", project.client_id)
```

One further step is required to get this tutorial working. You need to *publish* your app at Stackapps. This is done by asking a new question at Stackapps with the tag *app* . After you successfully posted your question, copy the link of it. Now, head back to the overview of your authorized apps, select your newly created app and paste the link of your question into the *Stack Apps Post* field.

6.4. Create Activity

Adjust the *activity_main.xml* layout or your activity.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="8dp"
    android:orientation="vertical"

tools:context="com.vogella.android.retrofitstackoverflow.MainAc
tivity">

    <Spinner
        android:id="@+id/questions_spinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <android.support.v7.widget.RecyclerView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

    <Button
        android:id="@+id/authenticate_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Authenticate" />
</LinearLayout>
```

Change your *MainActivity* activity code to the following.

```
package com.vogella.android.retrofitstackoverflow;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.Toast;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import java.util.ArrayList;
import java.util.List;

import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class MainActivity extends Activity implements
View.OnClickListener {

    private StackOverflowAPI stackoverflowAPI;
    private String token;

    private Button authenticateButton;

    private Spinner questionsSpinner;
    private RecyclerView recyclerView;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        questionsSpinner = (Spinner)
```

```
findViewById(R.id.questions_spinner);
        questionsSpinner.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent,
View view, int position, long id) {
                Question question = (Question)
parent.getAdapter().getItem(position);

stackoverflowAPI.getAnswersForQuestion(question.questionId).enq
ueue(answersCallback);
            }

            @Override
            public void onNothingSelected(AdapterView<?>
parent) {

            }
        });

        authenticateButton = (Button)
findViewById(R.id.authenticate_button);

        recyclerView = (RecyclerView) findViewById(R.id.list);
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new
LinearLayoutManager(MainActivity.this));

        createStackoverflowAPI();

stackoverflowAPI.getQuestions().enqueue(questionsCallback);
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (token != null) {
            authenticateButton.setEnabled(false);
        }
    }

    private void createStackoverflowAPI() {
        Gson gson = new GsonBuilder()
                .setDateFormat("yyyy-MM-dd'T'HH:mm:ssZ")
                .create();

        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(StackOverflowAPI.BASE_URL)

.addConverterFactory(GsonConverterFactory.create(gson))
                .build();

        stackoverflowAPI =
retrofit.create(StackOverflowAPI.class);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case android.R.id.text1:
                if (token != null) {

stackoverflowAPI.postUpvoteOnAnswer((Integer) v.getTag(),
token, getString(R.string.key), "stackoverflow.com", false,
"default").enqueue(upvoteCallback);
                } else {
                    Toast.makeText(this, "You need to
authenticate first", Toast.LENGTH_LONG).show();
                }
                break;
            case R.id.authenticate_button:
                startActivityForResult(new Intent(this,
WebViewActivity.class), 1);
                break;
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int 
resultCode, Intent data) {
        if (resultCode == RESULT_OK && requestCode == 1) {
            token = data.getStringExtra("token");
        }
    }

    Callback<ListWrapper<Question>> questionsCallback = new 
Callback<ListWrapper<Question>>() {
        @Override
        public void onResponse(Call<ListWrapper<Question>> 
call, Response<ListWrapper<Question>> response) {
            if (response.isSuccessful()) {
                ListWrapper<Question> questions = 
response.body();
                ArrayAdapter<Question> arrayAdapter = new 
ArrayAdapter<Question>(MainActivity.this, 
android.R.layout.simple_spinner_dropdown_item, 
questions.items);
                questionsSpinner.setAdapter(arrayAdapter);
            } else {
                Log.d("QuestionsCallback", "Code: " + 
response.code() + " Message: " + response.message());
            }
        }

        @Override
        public void onFailure(Call<ListWrapper<Question>> call, 
Throwable t) {
            t.printStackTrace();
        }
    };

    Callback<ListWrapper<Answer>> answersCallback = new 
Callback<ListWrapper<Answer>>() {
        @Override
        public void onResponse(Call<ListWrapper<Answer>> call, 
Response<ListWrapper<Answer>> response) {
            if (response.isSuccessful()) {
                List<Object> data = new ArrayList<>();
                data.add(questionsSpinner.getSelectedItem());
                data.addAll(response.body().items);
                recyclerView.setAdapter(new 
RecyclerViewAdapter(data, MainActivity.this));
            } else {
                Log.d("QuestionsCallback", "Code: " + 
response.code() + " Message: " + response.message());
            }
        }

        @Override
        public void onFailure(Call<ListWrapper<Answer>> call, 
Throwable t) {
            t.printStackTrace();
        }
    };

    Callback<ResponseBody> upvoteCallback = new 
Callback<ResponseBody>() {
        @Override
        public void onResponse(Call<ResponseBody> call, 
Response<ResponseBody> response) {
            if (response.isSuccessful()) {
                Toast.makeText(MainActivity.this, "Upvote 
successful", Toast.LENGTH_LONG).show();
            } else {
                Log.d("QuestionsCallback", "Code: " +
response.code() + " Message: " + response.message());
                Toast.makeText(MainActivity.this, "You already
upvoted this answer", Toast.LENGTH_LONG).show();
            }
        }

        @Override
        public void onFailure(Call<ResponseBody> call,
Throwable t) {
            t.printStackTrace();
        }
    };
}
```

To correctly create the views for your *RecyclerView* , add the following *RecyclerViewAdapter* to your project.

```
package com.vogella.android.retrofitstackoverflow;

import android.support.v7.widget.RecyclerView;
import android.text.Html;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import java.util.List;

public class RecyclerViewAdapter extends
RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder> {
    private final View.OnClickListener listener;
    private List<Object> data;

    private static final int VIEW_TYPE_QUESTION = 0;
    private static final int VIEW_TYPE_ANSWER = 1;

    public class ViewHolder extends RecyclerView.ViewHolder {
        public TextView text;


        public ViewHolder(View v) {
            super(v);
            text = (TextView)
v.findViewById(android.R.id.text1);
        }
    }

    public RecyclerViewAdapter(List<Object> data,
View.OnClickListener onItemClickListener) {
        this.data = data;
        this.listener = onItemClickListener;
    }

    @Override
    public RecyclerViewAdapter.ViewHolder
onCreateViewHolder(ViewGroup parent, int viewType) {
        View v;
        if (viewType == VIEW_TYPE_QUESTION) {
            v =
LayoutInflater.from(parent.getContext()).inflate(android.R.layo
ut.simple_list_item_1, parent, false);

v.setBackgroundColor(parent.getContext().getResources().getColo
r(android.R.color.darker_gray));
        } else {
            v =
LayoutInflater.from(parent.getContext()).inflate(android.R.layo
ut.simple_selectable_list_item, parent, false);
            v.setOnClickListener(listener);
        }
        return new ViewHolder(v);
    }

    @Override
    public void onBindViewHolder(RecyclerViewAdapter.ViewHolder
holder, int position) {
        if (getItemViewType(position) == VIEW_TYPE_QUESTION) {
            Question question = ((Question)
data.get(position));

holder.text.setText(Html.fromHtml(question.body).toString());
        } else {
            Answer answer = ((Answer) data.get(position));
            holder.text.setText(answer.toString());
            holder.itemView.setTag(answer.answerId);
        }
    }

    @Override
    public int getItemCount() {
        return data.size();
    }

    @Override
    public int getItemViewType(int position) {
        if (position == 0) {
            return VIEW_TYPE_QUESTION;
        } else {
            return VIEW_TYPE_ANSWER;
        }
    }
}
```

To handle authorization to upvote an answer, add the following activity named *WebViewActivity*. Ensure that you also declare this activity in your manifest file.

```
package com.vogella.android.retrofitstackoverflow;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class WebViewActivity extends Activity{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        WebView webView = new WebView(this);

webView.getSettings().setJavaScriptCanOpenWindowsAutomatically(
true);
        webView.getSettings().setJavaScriptEnabled(true);

webView.loadUrl("https://stackexchange.com/oauth/dialog?
client_id=" + getString(R.string.client_id) +
"&scope=no_expiry,write_access&redirect_uri=https://stackexchan
ge.com/oauth/login_success");
        webView.setWebViewClient(new WebViewClient(){
            @Override
            public boolean shouldOverrideUrlLoading(WebView
view, String url) {
                if(Uri.parse(url).getFragment() != null &&
!Uri.parse(url).getFragment().isEmpty()) {
```

```
!Uri.parse(url).getFragment().isEmpty()) {
                        String token =
Uri.parse(url).getFragment().replace("access_token=", "");

                        Intent intent = new Intent();
                        intent.putExtra("token", token);

WebViewActivity.this.setResult(Activity.RESULT_OK, intent);
                    }
                    return false;
                }
            });

            setContentView(webView);
        }
    }
```

Basically, the WebViewActivity just starts a *WebView* that lets you login into your Stackoverflow account to authorize this app. After you successfully granted permission, the Stackoverflow API returns your *access_token* to use the API functions that need authorization (we just need that for the upvote).

7. Exercise: Using Retrofit to access Github API in Android

7.1. Project setup

To test Retrofit create an application called *Retrofit Github*. Use the *com.vogella.android.retrofitgithub* top level package name.

To use Retrofit in your application, add the following dependency to your build.gradle file

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

Add the permission to access the Internet to your manifest file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.vogella.android.retrofitgithub" >

    <uses-permission
android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Add two buttons to your *activity_main.xml* layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <Button
        android:id="@+id/loadRepositories"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Load Repositories"
        android:gravity="center"
        android:layout_alignParentBottom="true"
        android:onClick="onClick"/>

    <Button

        android:id="@+id/loadUserData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Load user data"
        android:gravity="center"
        android:layout_above="@id/loadRepositories"
        android:onClick="onClick"/>
</RelativeLayout>
```

Create the following interface and class for the Retrofit API.

```
package com.vogella.android.retrofitgithub;

// This is used to map the JSON keys to the object by GSON
public class GithubRepo {

    String name;
    String url;

    @Override
    public String toString() {
        return(name + " " +  url);
    }
}
```

```
package com.vogella.android.retrofitgithub;

// This is used to map the JSON keys to the object by GSON
public class GithubUser {

    String login;
    String name;
    String email;

    @Override
    public String toString() {
        return(login);
    }
}
```

```java
package com.vogella.android.retrofitgithub;

import java.util.List;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;


public interface GithubAPI {
    String ENDPOINT = "https://api.github.com";

    @GET("/users/{user}")
    Call<GithubUser> getUser(@Path("user") String user);

    @GET("users/{user}/repos")
    Call<List<GithubRepo>> getRepos(@Path("user") String user);

}
```

```java
package com.vogella.android.retrofitgithub;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class MainActivity extends Activity implements
Callback<GithubUser> {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
    public void onClick(View view) {
        Gson gson = new GsonBuilder()
                .setDateFormat("yyyy-MM-dd'T'HH:mm:ssZ")
                .create();
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(GithubAPI.ENDPOINT)

.addConverterFactory(GsonConverterFactory.create(gson))
                .build();
        GithubAPI githubUserAPI =
retrofit.create(GithubAPI.class);

        switch (view.getId()) {
            case R.id.loadUserData:
                // prepare call in Retrofit 2.0

                Call<GithubUser> callUser =
githubUserAPI.getUser("vogella");
                //asynchronous call
                callUser.enqueue(this);
                break;
            case R.id.loadRepositories:
                case R.id.loadRepositories:
                    Call<List<GithubRepo>> callRepos =
githubUserAPI.getRepos("vogella");
                    //asynchronous call
                    callRepos.enqueue(repos);
                    break;
            }
        }


    Callback repos = new Callback<List<GithubRepo>>(){

        @Override
        public void onResponse(Call<List<GithubRepo>> call,
Response<List<GithubRepo>> response) {
            if (response.isSuccessful()) {
                List<GithubRepo> repos = response.body();
                StringBuilder builder = new StringBuilder();
                for (GithubRepo repo: repos) {
                    builder.append(repo.name + " " +
repo.toString());
                }
                Toast.makeText(MainActivity.this,
builder.toString(), Toast.LENGTH_SHORT).show();

            } else
            {
                Toast.makeText(MainActivity.this, "Error code "
+ response.code(), Toast.LENGTH_SHORT).show();
            }

        }

        @Override
        public void onFailure(Call<List<GithubRepo>> call,
Throwable t) {
            Toast.makeText(MainActivity.this, "Did not work " +
t.getMessage(), Toast.LENGTH_SHORT).show();
        }
    };

    @Override
    public void onResponse(Call<GithubUser> call,
Response<GithubUser> response) {
        int code = response.code();
        if (code == 200) {
            GithubUser user = response.body();
            Toast.makeText(this, "Got the user: " + user.email,
Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(this, "Did not work: " +
String.valueOf(code), Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onFailure(Call<GithubUser> call, Throwable t) {
        Toast.makeText(this, "Nope", Toast.LENGTH_LONG).show();

    }
}
```

Access the following URL in your browser: https://api.github.com/users/vogella and build a data model and interface with the data you are interested in. Read this data via Retrofit and show it in a list.

8. Exercise: Using Retrofit to post

comment on Github issue in Android

This exercise describes how to list all issues that are assigned to an user in an Android application using Retrofit. You can select an issue from a drop down field and post a comment on it. Make sure, that you have a Github account, as this tutorial doesn't work if you don't have one.

8.1. Project setup

Create an Android application with the name *Retrofit Github Comment*. Use *com.vogella.android.retrofitgithubcomment* as the top level package name.

To use Retrofit, add the following lines to your build.gradle file

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

Add the permission to access the Internet to your manifest file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.vogella.android.retrofitgithubcomment">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

8.2. Define the API

Create the following data class called *GithubIssue*.

```
package com.vogella.android.retrofitgithubcomment;

import com.google.gson.annotations.SerializedName;

public class GithubIssue {

    String id;
    String title;
    String comments_url;
    @SerializedName("body")
    String comment;

    public String getId() {
        return id;
    }


    public void setId(String id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getCommentsUrl() {
        return comments_url;
    }

    public void setCommentsUrl(String commentsUrl) {
        this.comments_url = commentsUrl;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }


    @Override
    public String toString() {
        return id +  " - " + title;
    }
}
```

We only show the id and the title of the issue in the drop down field, so we create a field for each of them. Furthermore, the response from Github contains the URL we have to post the comment to, which we store in the field *comments_url*. To later post a new comment to the Github API, we add a field called *comment*. The Github API specifies that the contents of a comment has to be bound to a field named *body* in the JSON request. As Retrofit (de)serializes all fields based on their name and as we don’t want to use *body* as the field name in our GithubIssue class, we use the `@SerializedName` annotation. With the help of this annotation, we can change the name a field is (de)serialized to in the JSON.

Define the REST API for Retrofit via the following interface:

```
package com.vogella.android.retrofitgithubcomment;

import java.util.List;

import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.GET;
import retrofit2.http.GET;
import retrofit2.http.Header;
import retrofit2.http.POST;
import retrofit2.http.Url;


public interface GithubAPI {
    String ENDPOINT = "https://api.github.com";

    @GET("/issues")
    Call<List<GithubIssue>> getIssues();

    @POST
    Call<ResponseBody> postComment(@Url String url, @Body
GithubIssue issue);
}
```

You might wonder about the `@Url` annotation. With the help of this annotation, we can provide the URL for this request. This allows us to change the URL for each request dynamically. We need this for the *comments_url* field of the GithubIssue class.

8.3. Create Activity

Add two *Buttons* (to load the issues and to send the comment), a *Spinner* (the drop down field to list the issues) and an *EditText* (to provide your comment) to your *activity_main.xml* layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

tools:context="com.vogella.android.retrofitgithubcomment.MainAc
tivity">

    <Spinner
        android:id="@+id/issues_spinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/comment_edittext"
        android:layout_below="@id/issues_spinner"
        android:enabled="false"
        android:maxLines="1"
        android:inputType="text"
        android:imeOptions="actionDone"
        android:hint="Your comment" />

    <Button
        android:id="@+id/loadIssues_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:gravity="center"
        android:onClick="onClick"
        android:text="Load issues" />
    <Button

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send comment"
        android:layout_above="@id/loadIssues_button"
        android:onClick="onClick"
        android:enabled="false"
        android:id="@+id/send_comment_button"/>
</RelativeLayout>
```

Change your activity code to allow querying for issues assigned to an user and to post a comment on an issue. Make sure, that you replace *aUsername* and *aPassword* with your Github login credentials.

```
package com.vogella.android.retrofitgithubcomment;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Toast;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import java.io.IOException;
import java.util.List;

import okhttp3.Credentials;
import okhttp3.Interceptor;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class MainActivity extends AppCompatActivity {

    GithubAPI githubAPI;
    String credentials = Credentials.basic("aUsername",
"aPassword");

    Spinner issuesSpinner;
    EditText commentEditText;
    Button sendButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        sendButton = (Button)
findViewById(R.id.send_comment_button);

        issuesSpinner = (Spinner)
findViewById(R.id.issues_spinner);
        issuesSpinner.setEnabled(false);


        commentEditText = (EditText)
findViewById(R.id.comment_edittext);

        createGithubAPI();
    }
    private void createGithubAPI() {
        Gson gson = new GsonBuilder()
                .setDateFormat("yyyy-MM-dd'T'HH:mm:ssZ")
                .create();

        OkHttpClient okHttpClient = new OkHttpClient.Builder()
                .addInterceptor(new Interceptor() {
                    @Override
                    public okhttp3.Response intercept(Chain
chain) throws IOException {
                        Request originalRequest =
chain.request();

                        Request.Builder builder =
originalRequest.newBuilder().header("Authorization",
                                credentials);

                        Request newRequest = builder.build();
                        return chain.proceed(newRequest);
                    }
                }).build();

        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(GithubAPI.ENDPOINT)
                .client(okHttpClient)

.addConverterFactory(GsonConverterFactory.create(gson))
                .build();

        githubAPI = retrofit.create(GithubAPI.class);
    }

    public void onClick(View view) {


        switch (view.getId()) {
            case R.id.loadIssues_button:
                Call<List<GithubIssue>> callIssues =
githubAPI.getIssues();
                callIssues.enqueue(issues);
                break;
            case R.id.send_comment_button:
                String newComment =
commentEditText.getText().toString();
                if (!newComment.isEmpty()) {
                    GithubIssue selectedItem = (GithubIssue)
issuesSpinner.getSelectedItem();
                    selectedItem.setComment(newComment);
                    Call<ResponseBody> postComment =
githubAPI.postComment(selectedItem.getCommentsUrl(),
selectedItem);
                    postComment.enqueue(comment);
                } else {
                    Toast.makeText(MainActivity.this, "Please
enter a comment", Toast.LENGTH_LONG).show();
                }
                break;
        }
    }

    Callback<List<GithubIssue>> issues = new
Callback<List<GithubIssue>>() {
        @Override
        public void onResponse(Call<List<GithubIssue>> call,
Response<List<GithubIssue>> response) {
            if (response.isSuccessful()) {
                List<GithubIssue> issuesList = response.body();
                GithubIssue[] idArray = issuesList.toArray(new
GithubIssue[issuesList.size()]);
                ArrayAdapter<GithubIssue> spinnerAdapter = new
ArrayAdapter<GithubIssue>(MainActivity.this,
android.R.layout.simple_spinner_dropdown_item, idArray);
                issuesSpinner.setAdapter(spinnerAdapter);
                issuesSpinner.setAdapter(spinnerAdapter);

                issuesSpinner.setEnabled(true);
                commentEditText.setEnabled(true);
                sendButton.setEnabled(true);
            } else {
                Log.e("onResponse failure", "Code: " +
response.code() + " , Message: " + response.message());
            }
        }

        @Override
        public void onFailure(Call<List<GithubIssue>> call,
Throwable t) {
            t.printStackTrace();
        }
    };

    Callback<ResponseBody> comment = new Callback<ResponseBody>
() {
        @Override
        public void onResponse(Call<ResponseBody> call,
Response<ResponseBody> response) {
            if (response.isSuccessful()) {
                commentEditText.setText("");
                Toast.makeText(MainActivity.this, "Comment
created", Toast.LENGTH_LONG).show();
            } else {
                Log.e("onResponse failure", "Code: " +
response.code() + " , Message: " + response.message());
            }
        }

        @Override
        public void onFailure(Call<ResponseBody> call,
Throwable t) {
            t.printStackTrace();
        }
    };
}
```

9. Exercise: Using Retrofit with OAuth to request user details from Twitter in Android

This exercise describes how to login into Twitter using Retrofit on Android. We write an application that can request and display user details for a provided user name. In this exercise, we use Twitters application-only authentication with OAuth 2 for

authorization. To make this tutorial working, you need to have a Twitter account. Furthermore, you need to head over to Twitter Apps and create a new app to get your consumer key and the corresponding secret. We need this later to request our OAuth token.

9.1. Project setup

Create an Android application with the name *Retrofit Twitter*. Use _com.vogella.android.retrofittwitter as the top level package name.

To use Retrofit, add the following lines to your build.gradle file

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

Add the permission to access the Internet to your manifest file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.vogella.android.retrofittwitter">

    <uses-permission
android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

9.2. Define the API

Create the following two data clases called *OAuthToken* and *UserDetails*.

```
package com.vogella.android.retrofittwitter;

import com.google.gson.annotations.SerializedName;

public class OAuthToken {

    @SerializedName("access_token")
    private String accessToken;


    @SerializedName("token_type")
    private String tokenType;

    public String getAccessToken() {
        return accessToken;
    }

    public String getTokenType() {
        return tokenType;
    }

    public String getAuthorization() {
        return getTokenType() + " " + getAccessToken();
    }
}
```

```
package com.vogella.android.retrofittwitter;

public class UserDetails {

    private String name;
    private String location;
    private String description;
    private String url;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }
}
```

The OAuthToken class is used to store the bearer token we request from Twitter with our consumer key and secret. We use the `@SerializedName` annotation to set the name Retrofit (de)serializes the fields with.

The UserDetails class simply stores a few fields from the response from Twitter when requesting the user details. We don't show all user details the response contains, but the name, location, url and description.

Define the REST API for Retrofit via the following interface:

```
package com.vogella.android.retrofittwitter;

import retrofit2.Call;
import retrofit2.http.Field;
import retrofit2.http.FormUrlEncoded;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.Query;

public interface TwitterApi {

    String BASE_URL = "https://api.twitter.com/";

    @FormUrlEncoded
    @POST("oauth2/token")
    Call<OAuthToken> postCredentials(@Field("grant_type")
String grantType);

    @GET("/1.1/users/show.json")
    @GET("/1.1/users/show.json")
    Call<UserDetails> getUserDetails(@Query("screen_name")
String name);
}
```

9.3. Create Activity

Modify your *activity_main.xml* layout file and the corresponding *MainActivity* class like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

tools:context="com.vogella.android.retrofittwitter.MainActivity
">

    <LinearLayout
        android:id="@+id/username_container"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/username_textview"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:enabled="false"
            android:gravity="center_vertical"
            android:text="Username:" />

        <EditText
            android:id="@+id/username_edittext"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:enabled="false"
            android:gravity="center"
            android:imeOptions="actionDone"
            android:inputType="text"
            android:maxLines="1" />
    </LinearLayout>


    <Button
        android:id="@+id/request_token_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:onClick="onClick"
        android:text="Request token" />

    <Button
        android:id="@+id/request_user_details_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_above="@id/request_token_button"
        android:enabled="false"
        android:onClick="onClick"
        android:text="Request user details" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
```

```
android:layout_height="match_parent"
android:layout_above="@id/request_user_details_button"
android:layout_below="@id/username_container"
android:gravity="center"
android:orientation=
```