Using Retrofit 2.x as REST client - Tutorial

Lars Vogel, Simon Scholz, David Weiser (c) 2012, 2017 vogella GmbH – Version 2.3, 12.09.2017

Table of Contents

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

1. Retrofit

1.1. What is Retrofit

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.

JAVA

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

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)
```
JAVA

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)
```
JAVA

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)
```
JAVA

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:

- 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/ (https://github.com/square/retrofit/tree/master/retrofit-adapters).

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

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

or using Apache Maven:

```
    <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.

JAVA

```
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.

JAVA

```
@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:

```
JAVA
@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.

```
JAVA
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

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

```
JAVA
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.

```
JAVA
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 (http://www.vogella.com/tutorials/Gradle/article.html) and Buildship for Eclipse (http://www.vogella.com/tutorials/EclipseGradle/article.html).

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.

```
XML
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.

```
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.

JAVA

```
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).

```
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 = 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();
```


```
        }
    }

    @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();
    }
}
```

JAVA

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 (http://www.vogella.com/tutorials/Gradle/article.html) and Buildship for Eclipse (http://www.vogella.com/tutorials/EclipseGradle/article.html).

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.

```
XML
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:

```
<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/article.html</lin
k>
    <author>lars.vogel@vogella.com (Lars Vogel)</author>

<guid>http://www.vogella.com/tutorials/AndroidTestingEspresso/article.html</gui
d>
</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.html</link>
    <author>lars.vogel@vogella.com (Lars Vogel)</author>
    <guid>http://www.vogella.com/tutorials/EclipseGradle/article.html</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</link>
    <author>lars.vogel@vogella.com (Lars Vogel)</author>
    <guid>http://www.vogella.com/tutorials/Mockito/article.html</guid>
</item>
```

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*.

```
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;
    }
}
```

```
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;
    }


}
```

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.

JAVA

```
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.

```
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.

JAVA

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

public class Application {

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

}
```

6. Exercise: Build an application for querying StackOverflow

StackOverflow (http://stackoverflow.com/) is a popular side for asking programming orientated questions. It also provides a REST API which is well documented on the Stackoverflow API side (https://api.stackexchange.com/docs/search).

In this exercise you use the Retrofit REST library. You will use it to query StackOverflow for tagged questions and their answers.

We use the following query URL in our example. Open this URL in the browser and have a look at the response.

```
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.stackoverflow*. Use *com.vogella.android.stackoverflow* as the top level package name.

Add the following dependency to your build.gradle file.

```
compile "com.android.support:recyclerview-v7:25.3.1"
compile 'com.google.code.gson:gson:2.8.1'
```

XML

6.2. Create data model

We are interested in the questions and answers from Stackoverflow. Create the following two data classes for that purpose.

```
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);
    }
}
```


```
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");
    }
}
```

6.3. Create activity and adjust layout

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

```
    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.stackoverflow.MainActivity">

    <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>
```

Add the a recycler view adapter class called `RecyclerViewAdapter` to your project.

One possible implementation could look like the following listing.

```
import android.support.v7.widget.RecyclerView;
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 List<Answer> data;


    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<Answer> data) {
        this.data = data;
    }

    @Override
    public RecyclerViewAdapter.ViewHolder onCreateViewHolder(ViewGroup parent,
int viewType) {
        View v;
        v =
LayoutInflater.from(parent.getContext()).inflate(android.R.layout.simple_select
able_list_item, parent, false);
        return new ViewHolder(v);
    }

    @Override
```

```
            holder.text.setText(answer.toString());
            holder.itemView.setTag(answer.answerId);
        }

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

    }
```

Change your *MainActivity* class to the following.

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

public class MainActivity extends Activity implements View.OnClickListener {

    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) {
                Toast.makeText(MainActivity.this, "Spinner item selected",
Toast.LENGTH_LONG).show();
```

```
            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));

    }

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


    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case android.R.id.text1:
                if (token != null) {
                    // TODO
                } else {
                    Toast.makeText(this, "You need to authenticate first",
Toast.LENGTH_LONG).show();
                }
                break;
            case R.id.authenticate_button:
                // TODO
                break;
```

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

6.4. Use a fake data provider

Create a fake data provider and fill the spinner with fake questions and the recyclerview of fake answers (once the spinner selection changes).

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


public class FakeDataProvider {
    public static List<Question> getQuestions(){
        List<Question> questions = new ArrayList<>();
        for (int i = 0; i<10; i++) {
            Question question = new Question();
            question.questionId = String.valueOf(i);
            question.title = String.valueOf(i);
            question.body = String.valueOf(i) + "Body";
            questions.add(question);
        }
        return questions;
    }
    public static List<Answer> getAnswers(){
        List<Answer> answers = new ArrayList<>();
        for (int i = 0; i<10; i++) {
            Answer answer = new Answer();
            answer.answerId = i;
            answer.accepted = false;
            answer.score = i;
            answers.add(answer);
        }
        return answers;
    }
}
```

Now configure the spinner and the recyclerview to use this fake data.

```
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.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.Toast;

import java.util.List;

public class MainActivity extends Activity implements View.OnClickListener {

    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) {
                Toast.makeText(MainActivity.this, "Spinner item selected",
Toast.LENGTH_LONG).show();
            }
```

```
            }
        });

        List<Question> questions = FakeDataProvider.getQuestions();
        ArrayAdapter<Question> arrayAdapter = new ArrayAdapter<Question>
(MainActivity.this, android.R.layout.simple_spinner_dropdown_item, questions);
        questionsSpinner.setAdapter(arrayAdapter);

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

        recyclerView = (RecyclerView) findViewById(R.id.list);
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new
LinearLayoutManager(MainActivity.this));
        List<Answer> answers = FakeDataProvider.getAnswers();
        RecyclerViewAdapter adapter = new RecyclerViewAdapter(answers);
        recyclerView.setAdapter(adapter);
    }

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


    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case android.R.id.text1:
                if (token != null) {
                    // TODO
                } else {
                    Toast.makeText(this, "You need to authenticate first",
```

```
            case R.id.authenticate_button:
                // TODO
                break;
        }
    }

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

}
```

6.5. Add Gradle dependencies and permissions

Add the following dependency to your build.gradle file.

```
XML
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.

```
    package="com.vogella.android.stackoverflow">
    <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>
```

6.6. Define the API and the Retrofit adapter

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 is needed to handled the Stackoverflow items wrapper. This class accepts a type parameter and simply wraps a list of objects of that type.

```
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.stackoverflow;

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);

}
```

6.7. Adjust activity

Change your *MainActivity* activity code to the following.

```
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) {
```

```
        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).enqueue(answersCall
back);
            }

            @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);
        }
```

```
        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) {
                    //TODO
                } else {
                    Toast.makeText(this, "You need to authenticate first",
Toast.LENGTH_LONG).show();
                }
                break;
            case R.id.authenticate_button:
                // TODO
                break;
        }
    }

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

```
        @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<Answer> data = new ArrayList<>();
                data.addAll(response.body().items);
                recyclerView.setAdapter(new RecyclerViewAdapter(data));
            } else {
                Log.d("QuestionsCallback", "Code: " + response.code() + "
Message: " + response.message());
            }
        }
```

```
        }
    };

    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();
        }
    };
}
```

6.8. Optional: Get the profile pictures from the user

Change your row layout in the recycler view to show also the profile image of the user. Extend your data model to also receive the profile picture of the user who answered the question. Add an `ImageView` to your row layout and use the Glide library to download the picture.

6.9. Optional: Use different layouts for even and odd rows

This requires that inflate different layouts based on the data type. Use `getItemViewType` in the adapter.

6.10. Optional: Handle network failure

If you have a network failure, show a retry button instead of the main user interface.

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

This exercise describes how to list all Github repositories for an user in an Android application using Retrofit. You can select a repository from a drop down field and list the issues that are assigned to the user for the selected repository. You can then select an issue from an additional drop down field and post a comment on it. A *DialogFragment* is used to enter the credentials for authentication. Make sure, that you have a Github account, as this tutorial doesn't work if you don't have one. As Retrofit is used in conjunction with RxJava2 during this tutorial, make sure that you also check the RxJava2 Tutorial (http://www.vogella.com/tutorials/RxJava/article.html).

7.1. Project setup

Create an Android application with the name *Retrofit Github*. Use *com.vogella.android.retrofitgithub* as the top level package name and use the empty template. Ensure to select the *Backwards Compatibility* flag in the wizard.

the following lines to your build.gradle file

```
GRADLE
compile 'com.squareup.retrofit2:retrofit:2.3.0'
compile 'com.squareup.retrofit2:converter-gson:2.3.0'
compile 'com.squareup.retrofit2:adapter-rxjava2:2.3.0'
compile 'io.reactivex.rxjava2:rxandroid:2.0.1'
```

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

```
XML
<?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:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
android:name="com.vogella.android.retrofitgithub.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

7.2. Define the API

Create the following two data classes called *GithubIssue* and *GithubRepo*.

JAVA

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

import com.google.gson.annotations.SerializedName;

public class GithubIssue {

    String id;
    String title;
    String comments_url;

    @SerializedName("body")
    String comment;

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

```
public class GithubRepo {
    String name;
    String owner;
    String url;

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

From the repositories, only the name and the url of the repository is displayed in the drop down field. Also add *owner* to the data class as the owner name is mandatory for requesting the issues later. We only show the id and the title of the issue in the drop down field, so create a field for each of them. Furthermore, the response from Github contains the URL to post the comment to, which is stored in the field *comments_url*. To later post a new comment to the Github API, add a field called *comment*. The Github API (https://developer.github.com/v3/issues/comments/#create-a-comment) 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.

Unfortunately the GithubRepo class is not enough to query all needed repository information. As you can see here (https://developer.github.com/v3/repos/), the owner of a repository is a separate JSON object within the repository response and therefore it normally needs a corresponding Java class for (de)serialization. Fortunately Retrofit

deserialized, this custom Deserializer is used.

To do so, add the following class called *GithubRepoDeserialzer*.

JAVA

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

import com.google.gson.JsonDeserializationContext;
import com.google.gson.JsonDeserializer;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParseException;

import java.lang.reflect.Type;

public class GithubRepoDeserializer implements JsonDeserializer<GithubRepo> {

    @Override
    public GithubRepo deserialize(JsonElement json, Type typeOfT,
JsonDeserializationContext context) throws JsonParseException {
        GithubRepo githubRepo = new GithubRepo();

        JsonObject repoJsonObject =  json.getAsJsonObject();
        githubRepo.name = repoJsonObject.get("name").getAsString();
        githubRepo.url = repoJsonObject.get("url").getAsString();

        JsonElement ownerJsonElement = repoJsonObject.get("owner");
        JsonObject ownerJsonObject = ownerJsonElement.getAsJsonObject();
        githubRepo.owner = ownerJsonObject.get("login").getAsString();

        return githubRepo;
    }
}
```

Define the REST API for Retrofit via the following interface:

```
import java.util.List;

import io.reactivex.Single;
import okhttp3.ResponseBody;
import retrofit2.http.Body;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.Path;
import retrofit2.http.Url;

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

    @GET("user/repos?per_page=100")
    Single<List<GithubRepo>> getRepos();

    @GET("/repos/{owner}/{repo}/issues")
    Single<List<GithubIssue>> getIssues(@Path("owner") String owner,
@Path("repo") String repository);

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

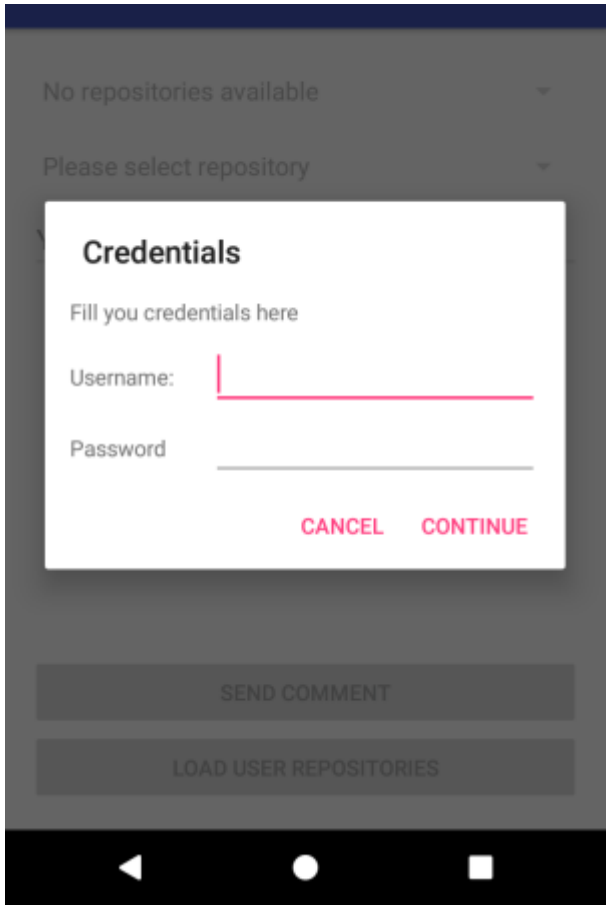
You might wonder about the `@Url` annotation. With the help of this annotation, we can provide the URL for this request. This allows us to change the URL for each request dynamically. We need this for the *comments_url* field of the GithubIssue class.

The `@Path` annotation binds the parameter value to the corresponding variable (curly brackets) in the request URL. This is needed to specify the owner and the repository name for which the issues should be requested

7.3. Create Credentials dialog

To give the user the opportunity to store their credentials in the application a *DialogFragment* is used. Therefore, create the following class named *CredentialsDialog* and also add a xml layout file named *dialog_credentials.xml* to your layout resources folder.

The result should look similar to the following screenshot.



```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginTop="16dp"
    android:text="Fill you credentials here" />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:orientation="horizontal">

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="0.3"
        android:text="Username:" />

    <EditText
        android:id="@+id/username_edittext"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="0.7" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
```

```
        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="0.3"
            android:text="Password" />

        <EditText
            android:id="@+id/password_edittext"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="0.7"
            android:inputType="textPassword" />
    </LinearLayout>
</LinearLayout>
```


```
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.DialogFragment;
import android.support.v7.app.AlertDialog;
import android.view.View;
import android.widget.EditText;

import okhttp3.Credentials;

public class CredentialsDialog extends DialogFragment {

    EditText usernameEditText;
    EditText passwordEditText;
    ICredentialsDialogListener listener;

    public interface ICredentialsDialogListener {
        void onDialogPositiveClick(String username, String password);
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        if (getActivity() instanceof ICredentialsDialogListener) {
            listener = (ICredentialsDialogListener) getActivity();
        }
    }

    @NonNull
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        View view =
getActivity().getLayoutInflater().inflate(R.layout.dialog_credentials, null);
        usernameEditText = (EditText)
view.findViewById(R.id.username_edittext);
```

```
        usernameEditText.setText(getArguments().getString("username"));
        passwordEditText.setText(getArguments().getString("password"));
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity())
                .setView(view)
                .setTitle("Credentials")
                .setNegativeButton("Cancel", null)
                .setPositiveButton("Continue", new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        if (listener != null) {

listener.onDialogPositiveClick(usernameEditText.getText().toString(),
passwordEditText.getText().toString());
                        }
                    }
                });
        return builder.create();
    }
}
```

7.4. Create Activity

Adjust the `activity_main.xml` layout to the following.

```
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context="com.vogella.android.retrofitgithub.MainActivity">

<android.support.v7.widget.Toolbar
    android:id="@+id/my_toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp" />

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">

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

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

    <EditText
        android:id="@+id/comment_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
            android:imeOptions="actionDone"
            android:inputType="text"
            android:maxLines="1" />

        <Button
            android:id="@+id/loadRepos_button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:enabled="false"
            android:gravity="center"
            android:onClick="onClick"
            android:text="Load user repositories" />

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

Two *Buttons* (to load the repositories and to send the comment), two *Spinners* (the drop down field to list the repositories and issues) and an *EditText* (to provide your comment) are added. To start the CredentialsDialog a menu in an Android *Toolbar* is used. To create one, add a xml menu file named *menu_main.xml* to your menu resources folder (Create the folder, if it does not exist).

```
    <item android:id="@+id/menu_credentials"
        android:title="Credentials"/>
</menu>
```

Because we are using the `Toolbar` widget, you have to turn off the default action bar. To do so, modify your xml style file to look like the one below.

XML

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Change your activity code to the following.

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
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 io.reactivex.android.schedulers.AndroidSchedulers;
import io.reactivex.disposables.CompositeDisposable;
import io.reactivex.observers.DisposableSingleObserver;
import io.reactivex.schedulers.Schedulers;
import okhttp3.Credentials;
import okhttp3.Interceptor;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.Retrofit;
import retrofit2.adapter.rxjava2.RxJava2CallAdapterFactory;
import retrofit2.converter.gson.GsonConverterFactory;

public class MainActivity extends AppCompatActivity implements
CredentialsDialog.ICredentialsDialogListener {

    GithubAPI githubAPI;
```

```
    Spinner repositoriesSpinner;
    Spinner issuesSpinner;
    EditText commentEditText;
    Button sendButton;
    Button loadReposButtons;

    private CompositeDisposable compositeDisposable = new
CompositeDisposable();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.my_toolbar);
        setSupportActionBar(toolbar);

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

        repositoriesSpinner = (Spinner)
findViewById(R.id.repositories_spinner);
        repositoriesSpinner.setEnabled(false);
        repositoriesSpinner.setAdapter(new ArrayAdapter<>(MainActivity.this,
android.R.layout.simple_spinner_dropdown_item, new String[]{"No repositories
available"}));
        repositoriesSpinner.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int
position, long id) {
                if (parent.getSelectedItem() instanceof GithubRepo) {
                    GithubRepo githubRepo = (GithubRepo)
parent.getSelectedItem();

compositeDisposable.add(githubAPI.getIssues(githubRepo.owner, githubRepo.name)
                            .subscribeOn(Schedulers.io())
                            .observeOn(AndroidSchedulers.mainThread())
```

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

            }
        });

        issuesSpinner = (Spinner) findViewById(R.id.issues_spinner);
        issuesSpinner.setEnabled(false);
        issuesSpinner.setAdapter(new ArrayAdapter<>(MainActivity.this,
android.R.layout.simple_spinner_dropdown_item, new String[]{"Please select
repository"}));

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

        loadReposButtons = (Button) findViewById(R.id.loadRepos_button);

        createGithubAPI();
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (compositeDisposable != null && !compositeDisposable.isDisposed()) {
            compositeDisposable.dispose();
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
```

```
                return true;
            }
            return super.onOptionsItemSelected(item);
        }

        private void showCredentialsDialog() {
            CredentialsDialog dialog = new CredentialsDialog();
            Bundle arguments = new Bundle();
            arguments.putString("username", username);
            arguments.putString("password", password);
            dialog.setArguments(arguments);

            dialog.show(getSupportFragmentManager(), "credentialsDialog");
        }

        private void createGithubAPI() {
            Gson gson = new GsonBuilder()
                    .setDateFormat("yyyy-MM-dd'T'HH:mm:ssZ")
                    .registerTypeAdapter(GithubRepo.class, new
GithubRepoDeserializer())
                    .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.basic(username, password));

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

```
                .baseUrl(GithubAPI.ENDPOINT)
                .client(okHttpClient)
                .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build();

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

    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.loadRepos_button:
                compositeDisposable.add(githubAPI.getRepos()
                        .subscribeOn(Schedulers.io())
                        .observeOn(AndroidSchedulers.mainThread())
                        .subscribeWith(getRepositoriesObserver()));
                break;
            case R.id.send_comment_button:
                String newComment = commentEditText.getText().toString();
                if (!newComment.isEmpty()) {
                    GithubIssue selectedItem = (GithubIssue)
issuesSpinner.getSelectedItem();
                    selectedItem.comment = newComment;

compositeDisposable.add(githubAPI.postComment(selectedItem.comments_url,
selectedItem)
                            .subscribeOn(Schedulers.io())
                            .observeOn(AndroidSchedulers.mainThread())
                            .subscribeWith(getCommentObserver()));
                } else {
                    Toast.makeText(MainActivity.this, "Please enter a comment",
Toast.LENGTH_LONG).show();
                }
                break;
        }
    }
```

```
        return new DisposableSingleObserver<List<GithubRepo>>() {
            @Override
            public void onSuccess(List<GithubRepo> value) {
                if (!value.isEmpty()) {
                    ArrayAdapter<GithubRepo> spinnerAdapter = new
ArrayAdapter<>(MainActivity.this,
android.R.layout.simple_spinner_dropdown_item, value);
                    repositoriesSpinner.setAdapter(spinnerAdapter);
                    repositoriesSpinner.setEnabled(true);
                } else {
                    ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<>
(MainActivity.this, android.R.layout.simple_spinner_dropdown_item, new String[]
{"User has no repositories"});
                    repositoriesSpinner.setAdapter(spinnerAdapter);
                    repositoriesSpinner.setEnabled(false);
                }
            }

            @Override
            public void onError(Throwable e) {
                e.printStackTrace();
                Toast.makeText(MainActivity.this, "Can not load repositories",
Toast.LENGTH_SHORT).show();

            }
        };
    }

    private DisposableSingleObserver<List<GithubIssue>> getIssuesObserver() {
        return new DisposableSingleObserver<List<GithubIssue>>() {
            @Override
            public void onSuccess(List<GithubIssue> value) {
                if (!value.isEmpty()) {
                    ArrayAdapter<GithubIssue> spinnerAdapter = new
ArrayAdapter<>(MainActivity.this,
android.R.layout.simple_spinner_dropdown_item, value);
```

```
                issuesSpinner.setAdapter(spinnerAdapter);
            } else {
                ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<>
(MainActivity.this, android.R.layout.simple_spinner_dropdown_item, new String[]
{"Repository has no issues"});
                issuesSpinner.setEnabled(false);
                commentEditText.setEnabled(false);
                sendButton.setEnabled(false);
                issuesSpinner.setAdapter(spinnerAdapter);
            }
        }

        @Override
        public void onError(Throwable e) {
            e.printStackTrace();
            Toast.makeText(MainActivity.this, "Can not load issues",
Toast.LENGTH_SHORT).show();
        }
    };
}

private DisposableSingleObserver<ResponseBody> getCommentObserver() {
    return new DisposableSingleObserver<ResponseBody>() {
        @Override
        public void onSuccess(ResponseBody value) {
            commentEditText.setText("");
            Toast.makeText(MainActivity.this, "Comment created",
Toast.LENGTH_LONG).show();
        }

        @Override
        public void onError(Throwable e) {
            e.printStackTrace();
            Toast.makeText(MainActivity.this, "Can not create comment",
Toast.LENGTH_SHORT).show();
        }
```

```
    @Override
    public void onDialogPositiveClick(String username, String password) {
        this.username = username;
        this.password = password;
        loadReposButtons.setEnabled(true);
    }
}
```

Here, add the previously created GithubRepoDeserializer as a TypeAdapter to the *GsonBuilder*. To handle the authentication for every call, add a *Interceptor* (http://www.vogella.com/tutorials/Retrofit/article.html#retrofit-authentication) to the *OkHttpClient*. To let the API interface methods return RxJava2 types, add the RxJava2 CallAdapter to your Retrofit client.

8. 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 (https://dev.twitter.com/oauth/application-only) 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 (https://apps.twitter.com/) and create a new app to get your consumer key and the corresponding secret. We need this later to request our OAuth token.

8.1. Project setup

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'
```

GRADLE

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>
```

XML

8.2. Define the API

JAVA

```java
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();
    }
}
```

```
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:

JAVA

```
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")
    Call<UserDetails> getUserDetails(@Query("screen_name") String name);
}
```

8.3. Create Activity

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

```
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: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_above="@id/request_user_details_button"
        android:layout_below="@id/username_container"
        android:gravity="center"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:minHeight="50dp">

            <TextView
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:gravity="center_vertical"
                android:text="Name:" />

            <TextView
                android:id="@+id/name_textview"
                android:layout_width="0dp"
                android:layout_height="match_parent"
```

```
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="50dp">

    <TextView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center_vertical"
        android:text="Location:" />

    <TextView
        android:id="@+id/location_textview"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center_vertical"
        android:text="---" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="50dp">

    <TextView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center_vertical"
        android:text="Url:" />

    <TextView
```

```
                android:layout_weight="1"
                android:gravity="center_vertical"
                android:text="---" />
        </LinearLayout>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:minHeight="50dp">

            <TextView
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:gravity="center_vertical"
                android:text="Description:" />

            <TextView
                android:id="@+id/description_textview"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:gravity="center_vertical"
                android:text="---" />
        </LinearLayout>
    </LinearLayout>
</RelativeLayout>
```

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

import java.io.IOException;

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

public class MainActivity extends AppCompatActivity {

    private String credentials = Credentials.basic("aConsumerKey", "aSecret");

    Button requestTokenButton;
    Button requestUserDetailsButton;
    EditText usernameEditText;
    TextView usernameTextView;

    TextView nameTextView;
    TextView locationTextView;
    TextView urlTextView;
    TextView descriptionTextView;

    TwitterApi twitterApi;
    OAuthToken token;
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        requestTokenButton = (Button) findViewById(R.id.request_token_button);
        requestUserDetailsButton = (Button)
findViewById(R.id.request_user_details_button);
        usernameEditText = (EditText) findViewById(R.id.username_edittext);
        usernameTextView = (TextView) findViewById(R.id.username_textview);

        nameTextView = (TextView) findViewById(R.id.name_textview);
        locationTextView = (TextView) findViewById(R.id.location_textview);
        urlTextView = (TextView) findViewById(R.id.url_textview);
        descriptionTextView = (TextView)
findViewById(R.id.description_textview);

        createTwitterApi();
    }

    private void createTwitterApi() {
        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",
                        token != null ? token.getAuthorization() :
credentials);

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

        Retrofit retrofit = new Retrofit.Builder()
```

```
                .build();

        twitterApi = retrofit.create(TwitterApi.class);
    }

    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.request_token_button:

twitterApi.postCredentials("client_credentials").enqueue(tokenCallback);
                break;
            case R.id.request_user_details_button:
                String editTextInput = usernameEditText.getText().toString();
                if (!editTextInput.isEmpty()) {

twitterApi.getUserDetails(editTextInput).enqueue(userDetailsCallback);
                } else {
                    Toast.makeText(this, "Please provide a username",
Toast.LENGTH_LONG).show();
                }
                break;
        }
    }

    Callback<OAuthToken> tokenCallback = new Callback<OAuthToken>() {
        @Override
        public void onResponse(Call<OAuthToken> call, Response<OAuthToken>
response) {
            if (response.isSuccessful()) {
                requestTokenButton.setEnabled(false);
                requestUserDetailsButton.setEnabled(true);
                usernameTextView.setEnabled(true);
                usernameEditText.setEnabled(true);
                token = response.body();
            } else {
                Toast.makeText(MainActivity.this, "Failure while requesting
```

```
            }
        }

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

    Callback<UserDetails> userDetailsCallback = new Callback<UserDetails>() {
        @Override
        public void onResponse(Call<UserDetails> call, Response<UserDetails>
response) {
            if (response.isSuccessful()) {
                UserDetails userDetails = response.body();

                nameTextView.setText(userDetails.getName() == null ? "no value"
: userDetails.getName());
                locationTextView.setText(userDetails.getLocation() == null ?
"no value" : userDetails.getLocation());
                urlTextView.setText(userDetails.getUrl() == null ? "no value" :
userDetails.getUrl());

descriptionTextView.setText(userDetails.getDescription().isEmpty() ? "no value"
: userDetails.getDescription());
            } else {
                Toast.makeText(MainActivity.this, "Failure while requesting
user details", Toast.LENGTH_LONG).show();
                Log.d("UserDetailsCallback", "Code: " + response.code() +
"Message: " + response.message());
            }
        }

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

Make sure, that you replace *aConsumerKey* and *aSecret* with the consumer key and secret provided by Twitter.

Also have a look at the interceptor we add to our Retrofit client. As we are using OAuth, our credentials are different for each call. The *postCredentials* method needs to post credentials in the *Basic* scheme to Twitter, which are composed from our consumer key and the secret. As a result, this call returns the bearer token Retrofit deserializes into our OAuthToken class, which we then store in the *token* field. Any further request can (and have to) now use this token as its credentials for authorization. So does the request to get the user details.

9. About this website

10. Retrofit resources

Consuming APIs with Retrofit tutorial (https://github.com/codepath/android_guides/wiki/Consuming-APIs-with-Retrofit)

In dept blog series about Retrofit (http://inthecheesefactory.com/blog/retrofit-2.0/en)

Consuming APIs with Retrofit (https://guides.codepath.com/android/Consuming-APIs-with-Retrofit)

10.1. vogella GmbH training and consulting support

TRAINING (http://www.vogella.com/training/)	SERVICE & SUPPORT (http://www.vogella.com/consulting/)
The vogella company provides comprehensive training and education services (http://www.vogella.com/training/) from experts in the areas of Eclipse RCP, Android, Git, Java, Gradle and Spring. We offer both public and inhouse training. Whichever course you decide to take, you are guaranteed to experience what many before you refer to as "The best IT class I have ever attended" (http://www.vogella.com/training/).	The vogella company offers expert consulting (http://www.vogella.com/consulting/) services, development support and coaching. Our customers range from Fortune 100 corporations to individual developers.

Appendix A: Copyright and License

Last updated 2018-02-13 12:00:13 +0100

Online Training
(https://learn.vogella.com)

QUICK LINKS

- 15 OCT - RCP Training (http://www.vogella.c
- vogella Training (http://www.vogella.c
- vogella Books (http://www.vogella.c

SHARE