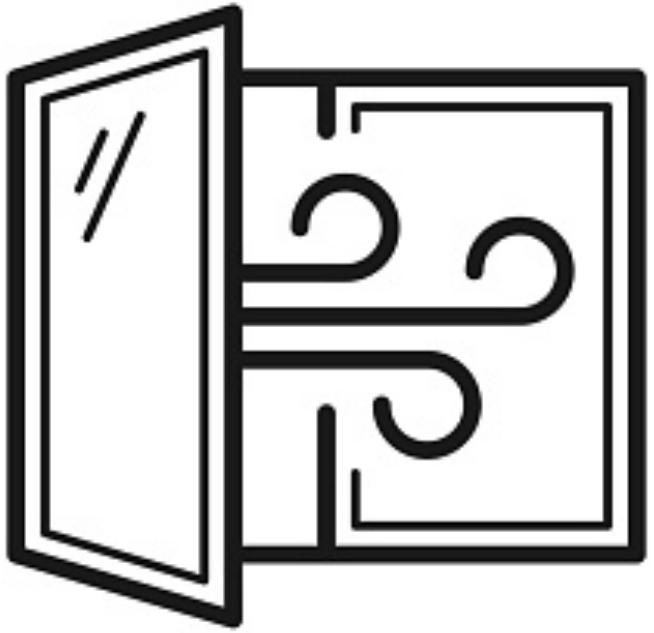
# Android App Programming

Lecture 3: Using web services
Oct 19, 2021

Anastasia Bezerianos
Université Paris-Saclay

course based in part on that of Thomas Nowak

# in class



# Lecture Format

- Videos of lectures online
  - (this week's is online, next will appear in a few days)
  - from now on lecture videos online ~week in advance
- Goal:
  - you watch the lectures at your own pace (before or in class) and you do the exercises, pause / rewatch
  - Anastasia walks around and answers questions

# Last lecture recap

# Intents

- Mechanism to launch activities from other activities
- To change activity: need to create object of type `Intent`, then call the method `startActivity` with that object
- ex :

```
Intent i = new Intent(this,
                      OtherActivity.class);
startActivity(i);
```

# Bundles

- Mechanism to pass data to activities: **Bundles**
- Can put data in and retrieve it
- ex:

```
bundle.putFloat("pi", 3.14);

bundle.getFloat("pi");
```

- An Intent comes by default with the bundle Extra `intent.putExtras("pi", 3.14)`

# Using web services

# Web services

- web service = set of  responses from a web server that aren’t HTML
- a web API (application programming interface) is a set of commands we can use to communicate with a server
- web APIs are defined on the server side

# Web services

- web service = set of  responses from a web server that aren't HTML
- return formats: typically JSON or XML
- query formats: often HTTP GET request w/ query strings
  - ex: https://www.google.com/search?q=hello
- find web APIs: www.programmableweb.com

# XML

- XML is eXtensible Markup Language
- used already for our user interface layout
- places keys inside < > brackets, followed by their values

```
<client>
<name>Jeff</name>
<age>32</age>
</client>
```

# JSON

- JSON “Javascript Object Notation”
- value/attribute pairs

```
[{client: {“name”:”Jeff”, “age”: 32}}]
```

# JSON example

```
{
  "type": "success",
  "value": [
    {
      "id": 166,
      "joke": "Chuck Norris doesn't play god.
               Playing is for children.",
      "categories": []
    },
    {
      "id": 557,
      "joke": "Chuck Norris can read from an input stream.",
      "categories": [
        "nerdy"
      ]
    }
  ]
}
```

# Parsing JSON

- Two main libraries in Java:
  - org.json
  - GSON
- both have more or less the same functionality (url, get, …)
- ex :
  `json.getAsJsonObject(“a”).get(“b”).getAsInt()`
- `{“a”: {“b”: 42}, “c”: {“d”: 22}}`

# Coding demo

# choose a REST API

- We will use a REST API, i.e., that follows the REpresentational State Transfer architecture [*]
- they are stateless and separate the client and server (their implementation can be done independently)
- server / client only know the format of msg
- REST-compliant systems are called RESTful systems

[*] for our first example we will use {JSON} Placeholder, a fake API for testing and prototyping.

# communicate

1. Allow your app to go online (in `AndroidManifest.xml`)

2. Use a library or client to do the HTTP request towards the API (e.g., Volley, Retrofit 2, Ion, …)

3. Read JSON data and turn them into a Java object using a library (e.g., org.json, GSON) [*]
   - You should have a class to store these objects, often called POJOs (Plain Old Java Objects)

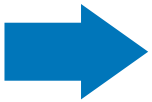
4. Setup & Make the Http request.
   - Use a callback function to wait for response

5. … do stuff with the data we read

[*] these libraries do de-serialization

1) Allow your app to go online (in `AndroidManifest.xml`)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.ups.lab3_firstcontact">

    <application
        …
        <activity
        …
        </activity>
    </application>

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

</manifest>
```



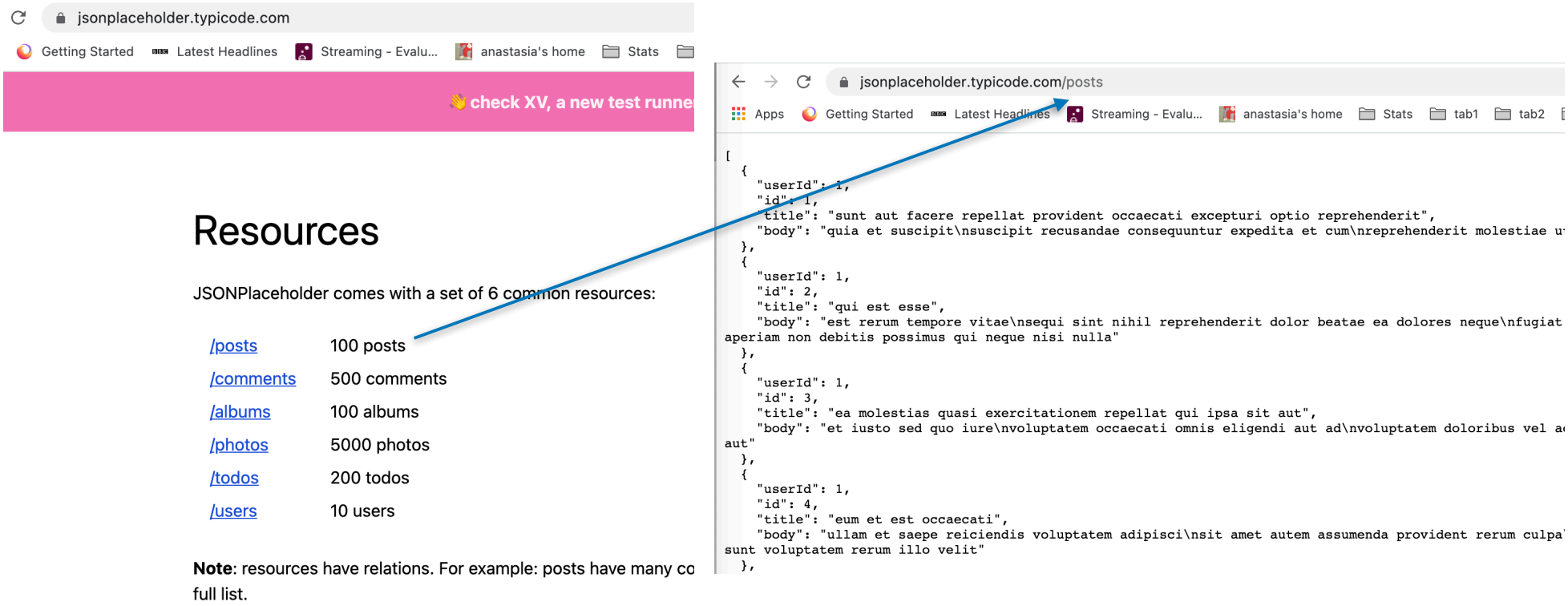
2) Add the needed libraries in Gradle

(Retrofit 2 for Http requests, Gson for reading JSON files)

Check the warning and do a Sync Now (to download them)

```
plugins {
    id 'com.android.application'
}

android {
    compileSdk 30

   …
}

dependencies {
   implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    implementation 'com.squareup.retrofit2:retrofit:2.4.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.4.0'
}
```

# 3a) Look at the fake API {JSON} Placeholder




**Link for placeholder server:** https://jsonplaceholder.typicode.com/

3b) Create a class to store objects from the JSON (deserialized data). Its variables should correspond to the data we read from the API.

```
package fr.ups.lab3_firstcontact;

public class PlaceholderPost {
    private int userID;
    private int id;
    private String title;
    private String body;

    public int getUserId() {
        return userID;
    }

    public int getId() {
        return id;
    }

    public String getTitle() {
        return title;
    }

    public String getBody() {
        return body;
    }
}
```

Names same as the ones used in the API

4a) Create an Interface (a pattern, similar to class)

Make sure you import the files for retrofit2

```
package fr.ups.lab3_firstcontact_test;

import java.util.List;
import retrofit2.Call;
import retrofit2.http.GET;

public interface PlaceholderAPI {
    @GET("posts")
    Call<List<PlaceholderPost>> getPosts();
}
```

needed imports

part of the url that “GET” will use

define how to store the entities it reads

# 4b) In your MainActivity.java, in onCreate setup the http connection

```
package fr.ups.lab3_firstcontact_test;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
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 AppCompatActivity {

    private TextView textView;
    private PlaceholderAPI myPlaceholderApi;

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

        textView = findViewById(R.id.textView);

        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("https://jsonplaceholder.typicode.com/")
                .addConverterFactory(GsonConverterFactory.create())
                .build();

        myPlaceholderApi = retrofit.create(PlaceholderAPI.class);
        get_all_posts();
    }
…
}
```

Setup the Retrofit reading
define the url
and setup the JSON conversion

read data
using the format
you've defined

a new method to treat data
(see next page)

# 4c) In your MainActivity.java, in your new method, treat data

```
…

public class MainActivity extends AppCompatActivity {
    …

    void get_all_posts(){

        Call<List<PlaceholderPost>> call = myPlaceholderApi.getPosts();

        call.enqueue(new Callback<List<PlaceholderPost>>() {
            @Override
            public void onResponse(Call<List<PlaceholderPost>> call,
                                   Response<List<PlaceholderPost>> response) {

                if (response.isSuccessful()) {
                    Log.d("Success", "Got something");
                    List<PlaceholderPost> posts = response.body();
                    for (PlaceholderPost p: posts) {
                        Log.d("Success", p.getTitle());

                        String content = p.getUserId() + "\n";
                        content += p.getId() + "\n";
                        content += p.getTitle() + "\n";
                        content += p.getBody() + "\n\n";

                        textView.append(content);
                    }
                } else {
                    textView.setText("Code " + response.code());
                    return;
                }
            }

            @Override
            public void onFailure(Call<List<PlaceholderPost>> call, Throwable t) {
                textView.setText(t.getMessage());
            }

        });
    }

}
```

our new method to treat data

setup the list that waits for data

a callback method that waits for data to be added in the list

onResponse here we add our code for what to do when we get data

onFailure what happens when we do not

!!! the parameters match the ones in my Interface ( `List<PlaceholderPost>`) in 4a

# troubleshooting (1)

- make sure your Emulator / phone is connected to the network
  - If your emulator does not connect, see this post
  - https://blog.tda-corp.co.jp/blog-tieng-viet/android-emulator-wifi-connected-with-no-internet/

# troubleshooting (2)

- (known problems)
  - Error "Installed Build Tools Revision 31.0.0 is corrupted", go into the file `Grardle Scripts > Build.gradle (module)` and change 31 and replace it by 30 in the following 3 places: `compileSdkVersion, buildToolsVersion, targetSdkVersion`.
  - For downloads, make sure your proxy is setup (in `Appearance & Behaviour > System Setup > HTTP Proxy`. The proxy's address is `proxy.iut-orsay.fr` and its port number is 3128.

# Lab 3

1. Use the ICNDB (Internet Chuck Norris Database)
2. Mix in some images from memegen.link

# Advance reading

- Next week's topic:
- Services
- Read https://developer.android.com/guide/components/services

# Questions