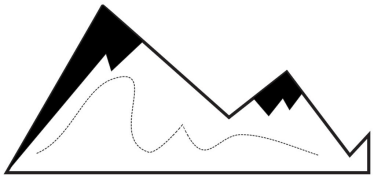
Mobile application for ski touring
LogGPX

Otba ZERAMDINI - Ali EL MUFTI - Robin DELBOS



Projet 2019/2020

Mr. Didier DONSEZ

Summary

Overview

On our 4th year in Polytechnic in the INFO major, we had the opportunity to choose a project that would last for next 3 months.
That project that we ended with was about the creation of a mobile application for Android around Ski-Hiking that could record our GPS trace and later on exploit it, analyse it, thanks to the website VisuGPX.com.

You'll be able to find the entirety of our code and of our documentation here :
https://gricad-gitlab.univ-grenoble-alpes.fr/Projets-INFO4/19-20/17

Introduction

Presentation

The VisuGPX company, and more particularly Mr. Jeroen Zijp who entrusted us with this project, is responsible for managing GPS tracks, the main functions of which are as follows:

- Advanced GPS track analysis
- Creation / modification of GPX traces
- GPX trace tracking with card caching

But they needed an application, "LogGPX", to allow recording of GPS tracks in the field. Current browsers do not allow this functionality because geolocation tracking is disabled in the background. It was therefore mandatory to go through a native application.

The main features that we had to implement were divided into 3 parts:

- Recording of traces in the field outside network coverage, with a map display if the network is available as well as related statistics (duration, distance traveled, elevation).
- Synchronization of the traces recorded with his account on VisuGPX (requires being logged in) or their direct download.
- Sharing the continuous recording as well as sending a message to the targeted contacts telling them the URL of the tracking page.

So we divided up the roles and each one worked on his personal branch at first:

Otba handled the development of the mapping and the generation of GPX and JSON files.

Ali took care of the creation of the login interface and connectivity with the API, as well as receiving, processing and sending decoded Json Objects to other activities.

Robin took care of the home page and the compatible display on most devices of the different elements that compose it.

Technologies used

When it comes to this project we mainly used Android Studio as our main IDE for the project, because we really wanted to code in native since we've been taught how to do it for the past two years.
But we had a lot of issues with this IDE because it was quite heavy on our computers and it could crash 2 times or even 3 times every 2 hours, so it became really frustrating over the time.
We also used XML in order to design our interfaces, adding buttons, slide bars and other widgets to trigger our Java codes.
We also used GPX for all the GPS coordinates and Json in order to store and reuse them.
We also interacted a lot with an API designed by Mr. Jeroen Zijp that provided us with all what we needed in order to connect this application to his website.
We also used to Http to get and post our requests from and to the website.

Development

The homepage

The homepage is presented as such :

It was implemented in these two files : ***Item_view.java*** and ***activity_item_view.xml.***

First of all, as we can see, we do have a menu button on the top right side of the screen that will allow us to connect or disconnect.

On the top of the page we also do have a button that allows us to start to record a new trace, by sending us to the MainActivity.

Next, we have a list of the different recordings that we have saved with different information for each one: the name of the recording, the date, the duration, the distance travelled and the height difference.

We also have several buttons for each record:

- A button to delete the record.
- A button to synchronize the recording with the website VisuGPX.com.
- A button to download the recording directly to your device.

The list view

The list view allows you to manage the different records and display them one below the other. To do this, we used the ***ItemAdapter.java*** class and the ***adapter_item.xml*** file which allow to manage one item of the list and then to adapt it to all the others.
The tricky part of this part was to make the display compatible on most devices (phones and tablets). For this, we used ***LinearLayout*** and the ***android:orientation*** attribute to distribute

the elements horizontally or vertically. We also used ***android:layout_weight*** attribute to distribute the space between the different elements to be displayed and ***android:layout_gravity*** and ***android:gravity attributes*** to position them in the right places.

The MainActivity (the map) :

The main activity is the core activity of our application. this activity implements to the **MainActivity.java** and **activity_main.xml.**
In this activity we used a Mapbox map as specified by the client. The activity allows us to visualize and to show specific informations such as the altitude, the time spent recording ect..

The main activity works as the following :
Once the user arrives here, he will see this ;

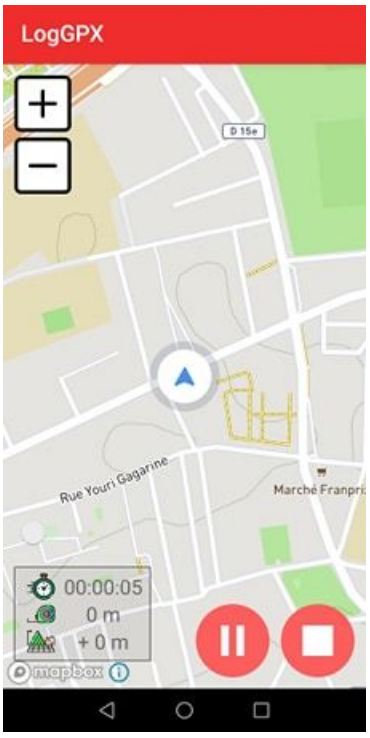
This activity is made out of a mapbox, zoom buttons written on the **activity_main.xml** and implemented on **MainActivity.java.**
We can also see that on the bottom left corner of the interface three components :

- A timer icon allowing us to know for how long the recording has been going
- A distance icon allowing us to know the distance travelled by the user while recording
- An elevation icon allowing the user to know in real time the elevation of his current place

Finally, we can also see in this MainActivity 2 red buttons :

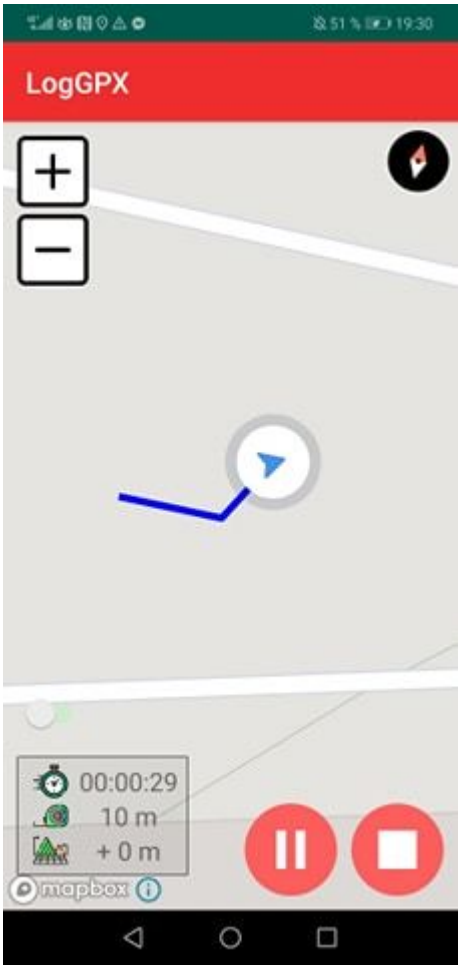
- Le bouton Play/Pause, qui permet de commencer un enregistrement ou de l'arrêter momentanément, c'est-à-dire le mettre en pause.
- A play button that starts a recording of a trace if you press it one time and if you press it another time it'll momentarily stop the recording.

- A stop button allows the user to stop the current recording. Once the play button pressed by the user, the main_activity looks like this :



The user is now located and a maker appeared on the map, showing its current location but also its orientation.
The map is self-centered on the current position.
We also have a chronometer implemented in order to know how long his the recording had been going.
In order to implement that we added the class :
Chronometer.java a class that implements Runnable.
The Pause button stops the recording but also the timer.

When the user moves, the MainActivity looks like this:



Between each user location report, a blue path is drawn, allowing the user to have a representation of the path traveled during the current recording. For example in the example above: the user has to walk 10 meters in 29 seconds and has not encountered any particular elevation.

The user can then turn off their phone and move around as they wish. At any time he will be able to visualize: the path he passed by (blue lines), but also in the square at the bottom right his journey time or recording time, without forgetting the distance and the difference in altitude he will have traveled.

When the phone has no network the MainActvity looks like this:

A yellow icon with a black exclamation mark appears. It lets the user know that he does not have a network and therefore that this data cannot be sent to his VisuGPX.com account. To do this, we used a thread that we launched when we started the application. This is the Network class which inherits from the Thread class. The purpose of this thread is to continuously check whether the network on the mobile phone is available or not.
If the network is not available on the phone, through a UiThread, the Network thread changes the view of MainActivity, that is to say that when the phone does not have access to the network (both wifi and 4G, 3G…), an icon appears indicating to the user that he does not have access to the network. If he has access to the network this icon disappears.

Once the user has finished recording his tracks (hiking, biking, running, etc.), or has decided to take a break, he has several options:

- Pausing the recording for example if he wishes to take a break by clicking on the pause button:

A window is then displayed indicating to the user that the recording is paused and so if he wishes to stop or continue the recording.
Note that when the recording pauses the stopwatch stops. I pause the thread. On the other hand, the MainActivity unsubscribes from the location data providers, so it no longer receives any location data. The distance traveled and the elevation level are then paused. Until the user makes a choice the application is somewhat frozen.

- If the user answers no, he is automatically sent to the same interface as if he had pressed the stop button:

- When the user presses the stop button or responds no to the last window, the MainActivity will look like this:

A window appears asking the user to enter the name of his **GPX** file (file format allowing the exchange of GPS coordinates) of his current recording. We will come back to this later. As proposed by the person in charge of the project, we pre-filled the file name with the date of the recording. For example in the image above: this recording took place on 30/04/2020 at 19h and 30 minutes.

The user then has two choices: save this file or cancel the saving.

In both cases the user is returned to Item_view, the project homepage activity. The data will be sent to this activity and will be necessary for Robin to its display.

What's going on is pretty simple in **MainActivity.java :**

We subscribe to a provider of location data, so when the provider provides us the location data we realize certain operations that allow us to obtain the distance traveled and the elevation. Everything happens in the function:

```
public void onSuccess(LocationEngineResult result) ;
```

This function will be called every 2 seconds. We are offering new location data.
The **LocationEngineResult**, a result returned by the location data provider, allows us to retrieve the following data: longitudes, latitudes, altitudes :

```
activity.longitude=result.getLastLocation().getLongitude();


activity.latitude=result.getLastLocation().getLatitude();


activity.altitude=result.getLastLocation().getAltitude();
```

Distance calculation:

```
activity.distance+=Tools.distance(activity.latitude,activity.lastPoint.getLatitude(),activity.longitude,activity.lastPoint.getLongitude(),activity.altitude,activity.lastPoint.getAltitude());
```

Elevation calculation :

```
activity.elevation+=Tools.distance(0,0,0,0,activity.altitude,activity.lastPoint.getAltitude());
```

All this data will be sent to the ***FileRegisterDialog.java*** class. It is this class that will display the window that asks the user to enter the file name. Once the user has chosen to save, a date will be issued and the name entered will be retrieved.
All this data :

- Filename
- Registration date
- Duration of the recording
- Distance travelled
- Height difference

will be sent to Robin in his **Item_view** so he can display them and create a new record in his ListView.

Fichier JSON et GPX :

On the other hand, the **MainActivity,** not only stops to display the data relating to the location of the user, but it allows to generate files of type JSON and GPX containing the location data. The expected JSON file is of the type:

```
[
[45.21406,5.74749,"2019-11-05T15:29:45Z",497.8],
[45.21401,5.74752,"2019-11-05T15:29:46Z",496.8],
[45.21397,5.74763,"2019-11-05T15:29:47Z",494.7],
[45.21393,5.74789,"2019-11-05T15:29:48Z",495.8],
[45.21389,5.74849,"2019-11-05T15:29:49Z",493.1]
]
```

JSON files must be sent to visuGPX, to the user's account if the user is logged in and presses the synchronize button in the Item_view activity.

The Json data are written in JSON arrays : **private** JSONArray **array**;
in the **MainActivity.java.** Everything is working on the onSuccess method:

public void onSuccess(LocationEngineResult result) ;

Each time this function is gonna get called, every two seconds :
A line like this [“latitude”, “longitude”, “date”, “altitude”] will be added to the array :

```
array.put(activity.latitude);
array.put(activity.longitude);
array.put(activity.dateFormated);
array.put(activity.altitude);
//add json array to have an array of array.
activity.array.put(array);
```

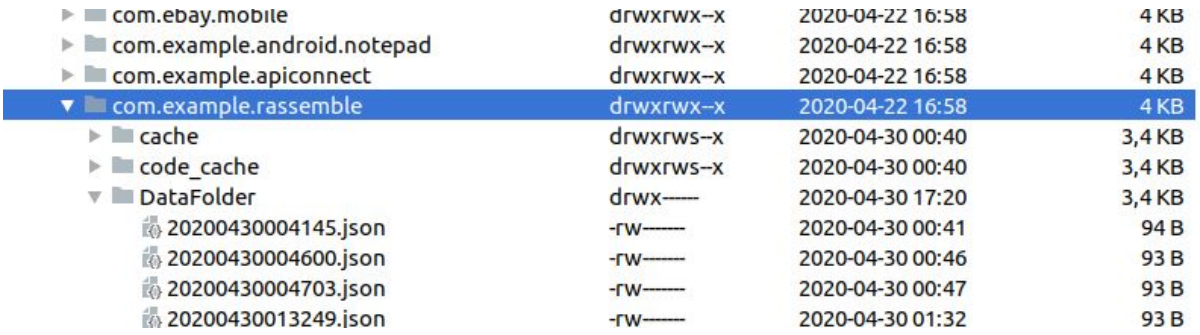
This table will then be sent to the **FileRegisterDialog.java class**, which when the user presses the save button after naming his file will once again send this data to the activity: **Item_view.** So in this activity the user can send this data to visuGPX by pressing the synchronization button. If the user presses the synchronization button. The static writeJSONFile method of the WriteJSON class, will be called with the jsonBody array as arguments, ie the array created in the **MainActivity class.**

absolutePathJson=WriteJSON.*writeJSONFile*(**context**, dateSavingJson,**jsonBody**);

This method will take care of saving this file in the internal memory of the application, i.e. you will not be able to see this file when browsing the phone. The Json file will be created in a folder: DataFolder.

```
//Creating the directory localy (in the internal storage od the app).
File checkFile = new File( pathname: context.getApplicationInfo().dataDir + "/DataFolder/");
if (!checkFile.exists()) {
    checkFile.mkdir();
}
//getting the absolute path
absolutePath=checkFile.getAbsolutePath() + "/"+dateSavingJson+".json";
//wriring the file in DataFolder in the Donwload folder.
FileWriter file = new FileWriter( fileName: checkFile.getAbsolutePath() + "/"+dateSavingJson+".json");
file.write(jsonBody);
file.flush();
file.close();
```

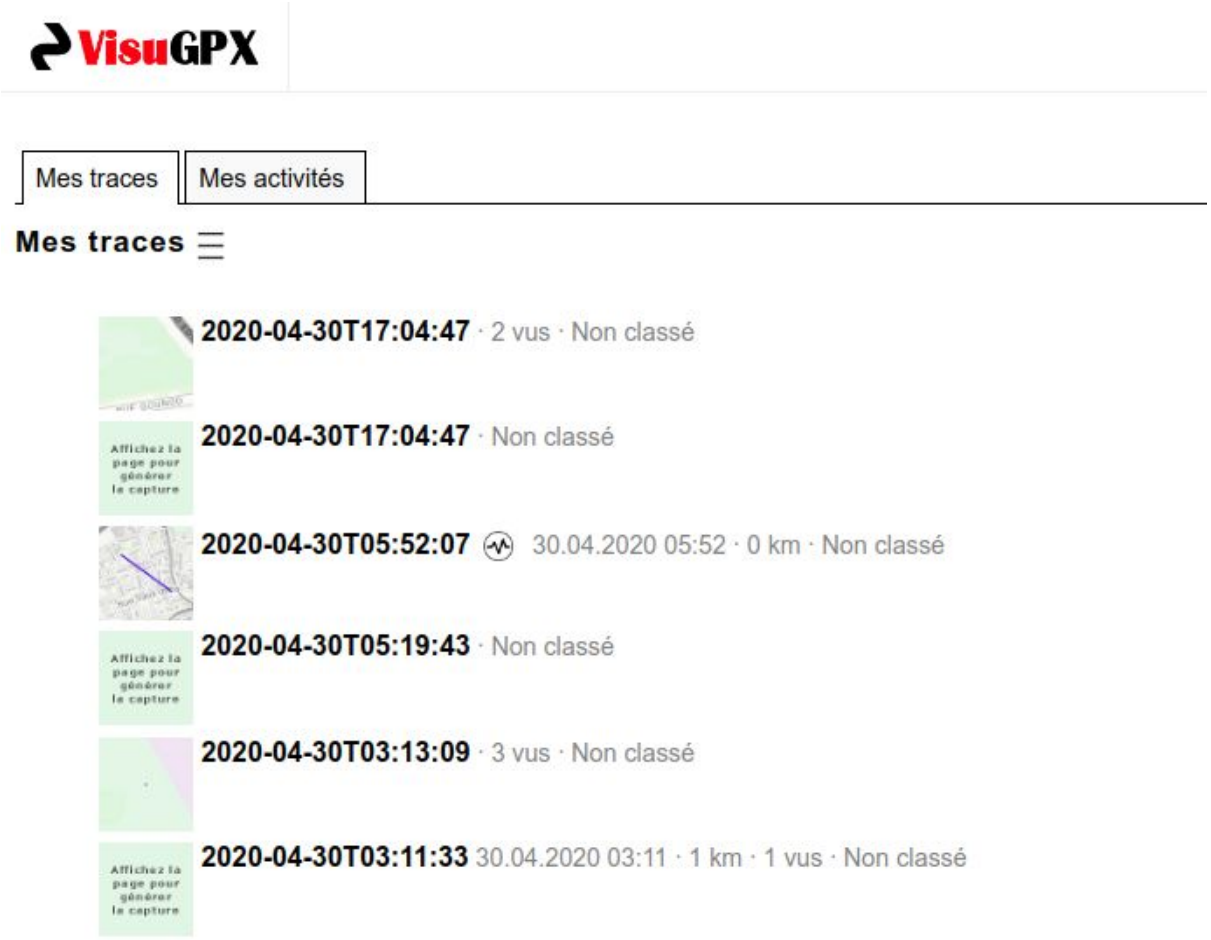
For example thanks to android studio we can visualize the internal memory and JSON files :



Then this data will be sent to the GPU account visu, of the user if this one on a thanks to the ApiPost.java class, which will send an Http request, with JSON file as request body:

```
OkHttpClient client= new OkHttpClient();

//Building the body of the httpRequest
RequestBody formBody = new  MultipartBody.Builder()
      .setType(MultipartBody.FORM)
      .addFormDataPart("user_id", useId)
      .addFormDataPart("json", absoluteJsonPath)
      .addFormDataPart("json",
nameJsonFile+".json",RequestBody.create(MediaType.parse("application/json"), new
File(absoluteJsonPath)))
      .build();
//  formBody = RequestBody.create(JSON, bodyJson);

//Building the header and adding the body of the HttpRequest
final Request request = new Request.Builder()
      .url("https://www.visugpx.com/api/pushtrack")
      .addHeader("cle", cle)  // add request headers
      .post(formBody)
      .build();
```

From where we get the recordings on VisuGPX which we can view later : **Item_view.**



For GPX files, these are also created thanks to the MainActivty and again thanks to the function:

```
public void onSuccess(LocationEngineResult result) ;
```

The body of the file and thus create in the following way:

```
activity.bodyGPX += "<trkpt lat=\"" + activity.latitude+ "\" lon=\"" + activity.longitude + "\">\n<ele>" + activity.altitude + "</ele>\n<time>" + df.format(new Date()) + "</time>\n</trkpt>\n";
```

This body of the method will then be sent as the coprs of the JSON file to the activity developed by Robin :

If the user decides to download his GPX file to his phone to access it later, read it, share it… .., he must press the download button.

La méthode static ***writeGpxFile*** dans **WriteGPX** :

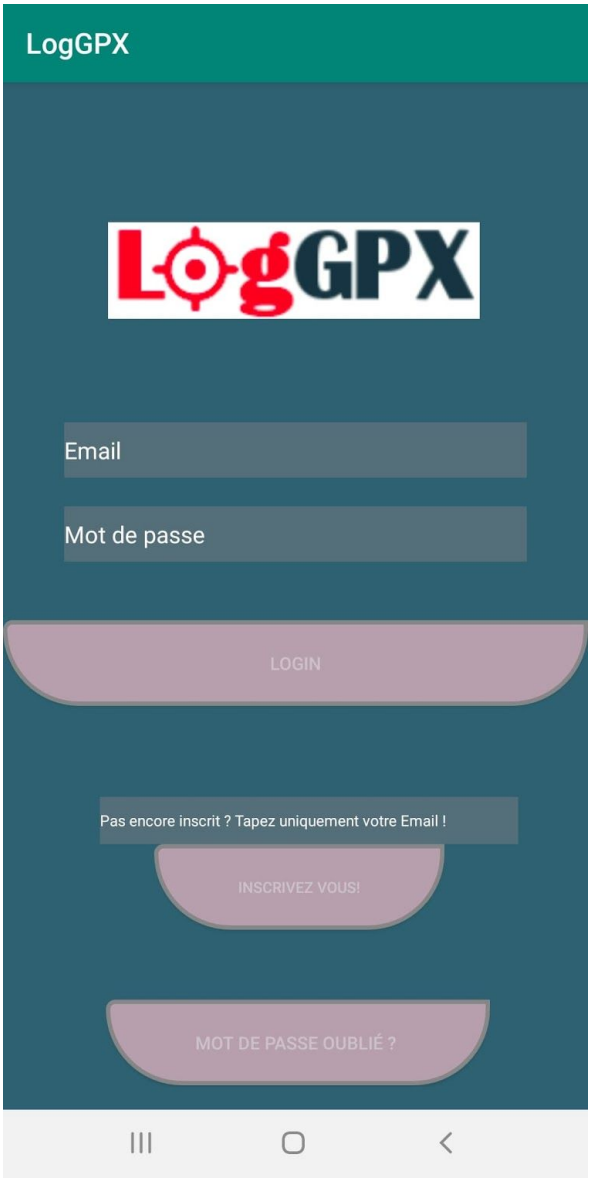
WriteGPX.*writeGpxFile*(**bodyGPX**,**nameFile**,**context**);

writeGpxFile then takes care of creating the file and creating the “logGPX” folder in Phone Downloads. This is called an external backup since the file is not saved in the application memory but outside in the internal memory of the phone.

```
//the header of GPX file.
String header = "<?xml version=\"1.0\" ?>\n<gpx
xmlns=\"http://www.topografix.com/GPX/1/1\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:schemaLocation=\"http://www.topografix.com/GPX/1/1
http://www.topografix.com/GPX/1/1/gpx.xsd\" version=\"1.1\"
creator=\"VisuGPX\">\n<trk>\n";

//The name of GPX file.
String name = "<name>" + fileName + "</name>\n<trkseg>\n";

//the footer of GPX file.
String footer = "</trkseg>\n</trk>\n</gpx>";

//The string witch represent the GPX file.
String gpxString = header + name + bodyGPX + footer;
```

The user can then recover his GPX file, share it, view it on a platform other than visuGPX ...

The Log in Activity



The log in activity is made out of two main text view that will allow us to gather all the information that we need for the use of the other three buttons which are :

- LOGIN BUTTON
- SUBSCRIBE BUTTON
- FORGOTTEN PASSWORD

For the last two options, only an email is required.

So let's dive in for a more precise explanation.
First of all, to handle that much buttons we had to create a switch case based on the the ID of the pressed button, then we pretty much execute the same pattern of commands for each and every one of the commands depending essentially on the arguments that we put on there.

So to begin we created an OkHTTPClient request because we figured out quite later on on this project that this particular request allowed us to get the desired header response from the API that can be changed to a Json object afterwards :

```
client = new OkHttpClient();
```

Then we proceed to create the body of the request. This is where we put all of our arguments, depending on the request that we want (Log in Subscribe Password forgotten).

For example :
```
formBody = new FormBody.Builder()
        .add("action", "login")
      .add("email", email)
       .add("pass", password)
        .build();
```

here we can see that for the Login request we have to use the password and email address and the action = login.

Finally we create the header

```
request = new Request.Builder()
        .url("https://www.visugpx.com/api/account")
        .addHeader("cle", "rKGjt2lkBQrhRV4nJIt7bP3ql")  //
add request headers
        .post(formBody)
        .build();
```

it will include the baseURL as well as the api_key that.
this part is identical for every request. Finally we lunch the client that we created earlier

```
client.newCall(request).enqueue(new Callback() {
```

and we end up trying to interpret the incoming response.
For example, if the login failed, it'll show us a message with the problem, but if the log in was successful, then, we proceed to transform the response to a Json Object in order to extract two essential informations :

```
JSONObject json= new JSONObject(myResponse);
JSONObject jsonObj1=json.getJSONObject("SUCCESS");
usr_id=jsonObj1.getString("id");
key=jsonObj1.getString("key");
```

The key and the id of a user.
These two variables will be really important for the other parts of the project.
Since we weren't able to create a database in order to store the data and reuse it in other activities, we decided to store these informations in global variables and just share them with the other classes. The other classes will just pass on these informations to the upcoming task before closing and so on.
These informations are really important because they allow Otba and Robin to send the current traces to the website as we can see it in the API :

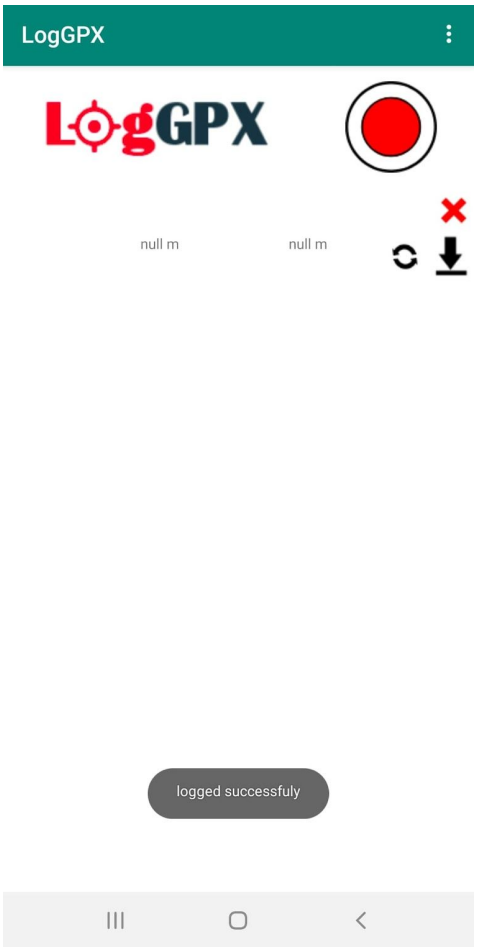
Exemple **: API POST https://www.visugpx.com/api/pushtrack**

```
Parameters (POST fields)
user_id : the id of the user
json  :  the JSON file with lat, lng, date, ele. Example :
[
[45.21406,5.74749,"2019-11-05T15:29:45Z",497.8],
```

```
[45.21401,5.74752,"2019-11-05T15:29:46Z",496.8],
[45.21397,5.74763,"2019-11-05T15:29:47Z",494.7],
[45.21393,5.74789,"2019-11-05T15:29:48Z",495.8],
[45.21389,5.74849,"2019-11-05T15:29:49Z",493.1]
]
```

Exemple : **API GET https://www.visugpx.com/api/marked**

```
Parameters (GET)
user_id : the id of the user
```



Once the login or the subscription done, we are redirected to the Item_View activity.
The subscribe and newpass code are the same, but for a single detail, instead of using the action subscribe, we use the action newpass and we only take as a parameter the email address and not the password.

The reason that this class took us so long to implement is because we used 2 past models that worked initially but failed to provide us the Json response that we all needed.
We first implemented this as we did in one of our classes because for once, we weren't sleepwalking and we could reuse a code that we understood.
Turned out that the protocols that we used weren't effective and didn't provide us the adequate responses. So we had to start all over again 2 times in a row before finding this solution and that can be seen in the past commits on Ali's branch

Conclusion

Progression

The entire group found this project very interesting. Indeed this project allowed us to put into practice our knowledge in Java, in a field that we did not know yet: the development of mobile applications. During this project we were fortunate to learn a lot of new things. We started at the starting point, asking ourselves what is an activity? So we have, it must be recognized had a lot of difficulty at the start since we knew nothing in the world of Android. So learning the basics of this new technology took us quite a while. On the other hand, the difficulty in getting started could also be explained by the fact that we had no source code at our disposal, to be inspired by it for example. We had to build an application from A to Z. Once the first month passed, the automations were put in place, the work was rather well shared and distributed. Our help throughout this project allowed us to overcome the difficulties that presented themselves to us. We really enjoyed working on this project and we found it very interesting. Perhaps the only regret we have is that we did not have enough time to further improve this project. Thus, this project in addition to strengthening our knowledge in JAVA allowed us to manipulate many new technologies, and to familiarize ourselves with, such as android application programming XML, JSON, GPX, HTTP….

Future prospects

As we have said before, our only regret is probably not having the time to go even further and improve the application.
Thus, if we had more time we would have improved the following points:

- We would have used SQL LITE databases on the phone to be able to keep all of the user's records even if the user leaves the application
- Allow users subscribed to visuGPX to post their location data live or continuously, to share it with other members of the visuGPX community, or to share it with friends on social networks (Facebook, Twitter, etc.).

- Choose a more precise localization provider, than that of Mapbox.
- Add a button in the MapActivity to change the type of map.
- Provide an interface that allows the user when logged into visuGPX how often he wants these location coordinates to be sent to VisuGPX.
- Allow visuGPX subscribers to send a message to targeted contacts to let them know the URL of the tracking page, so that they can follow their movement live.