Gauthier Denglos / 2999490
Benjamin Goré / 2999581

The goal of this application is to work on new features seen in class, like files or gps points, and work on other basis like xml files, activities connection, buttons… We will describe here who did what on this application and describe as much as possible everything that we code in the android studio project.

Tasks:	Gauthier	Benjamin
Shell consisting of 2 activities	X	
GPS points	X	
GPX files		X
Statistic display (second activity)	X	
Graph / Custom view		X
Speed / Distance	X	X
Custom features	X	X

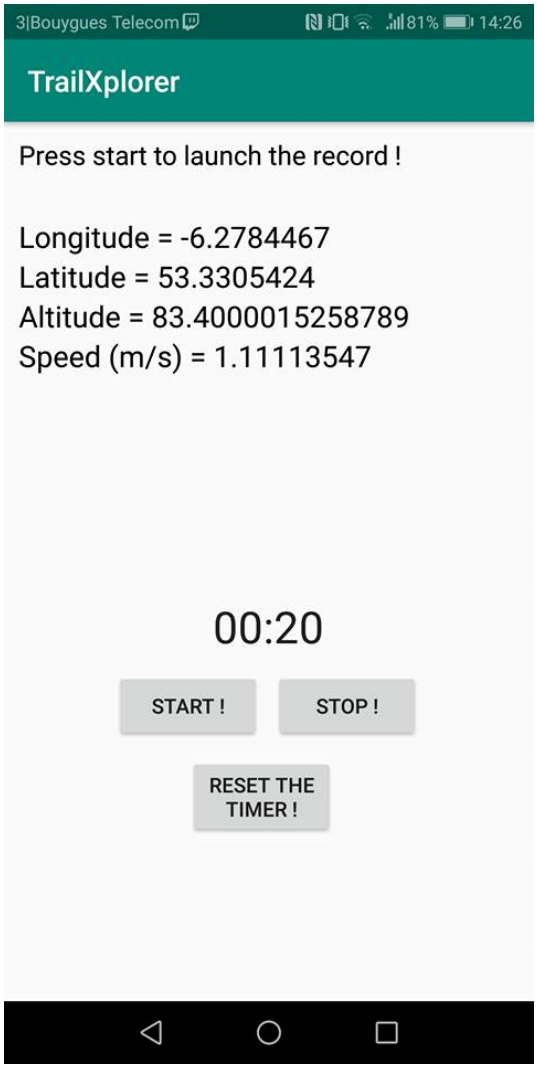

Figure 1: Main activity, after the start button has been pushed

This main activity is the first screen the user will see. The main goal of this activity is to display the different data while the user is using the app and started to record. Even if the location will start as soon as the app is used, the recording only starts when the start button is pushed by the user. At this precise moment, a gpx file will be created in the storage of the smartphone and the stopwatch will start increasing. The reset button allows the user to restart the chronometer. These buttons are all using basic button method that we already saw in the different previous project that we did, the new things we saw are how to get localisation data, display them and update them regularly. The different data will be the altitude, the longitude, the latitude, the speed and the time.

- The time will be obtained thanks to 2 main methods *"resetChronometer"* and *"startChronometer"* which both take view as parameter and return nothing. The goal of the first method is just to make the chronometer start back to 0, while the second method will continuously increase the stopwatch and set a Boolean to true, if the time is running. We use the chronometer library to get all the information.

- The location data will also use two main methods to operate correctly: the *"onLocationChanged"* and the *"run"* one. The first function takes a location in arguments, and will get the altitude, latitude and longitude of this location simply by calling javas method. It also allows to have the speed of the user by calculating the speed with V = D / T (where D is the distance between the current location and the previous location, and T is the difference between the time associated to the previous location and the current location). There is three other checks in this function: to know the minimum and maximum altitude during the journey, and if the speed is correct or not (Not a number and infinite bug). The run method is a little bit different. It uses a handler (allows to handle running object, here the different data we want to update) to update every 5 seconds the text Views and their data. The gpx part of this method will be explain later!

- Finally, the last thing interesting in this activity is the getter part. We created different getter to link this activity with the other one. There is 7 different getters, for the maximum and minimum altitude, the chronometer time, a table of the different speeds, the distance, the average speed (calculated in the *"onCreate"* method when the stop button is pushed, like the distance) and the time calculated by hand.

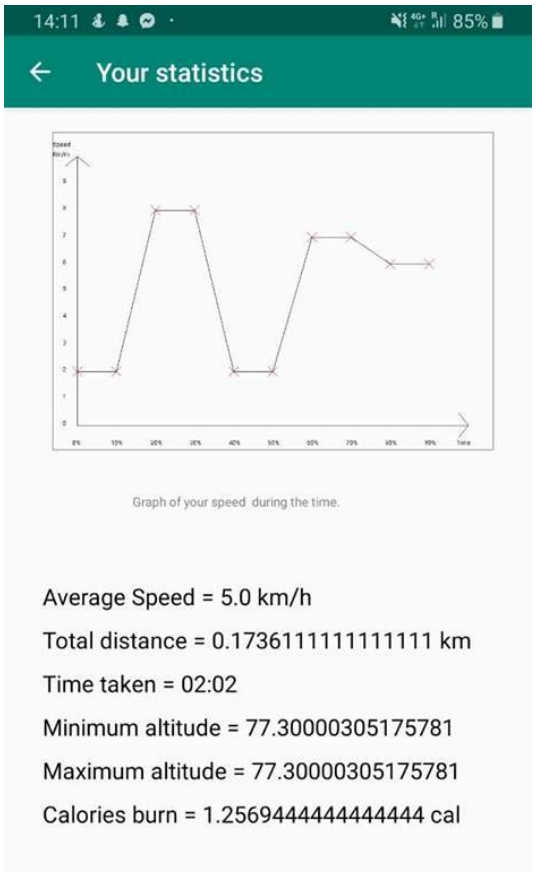

Figure 2: Second activity, after the stop button has been pushed

This activity is just composed of 2 main things: The custom view and a series of text views concatenate with the data we get in the first activity thanks to the different getter mentioned earlier. One of this text views will display the other feature, the calories burned calculation. For this, we analysed on the Internet the different parameter we needed to have, and we found that it depends mainly of the average speed and the duration. We choose to add this feature because when we are walking or running, it is always satisfying to know how many calories we burned during the effort.

The graphic is very important to see what you achieve and allows to clearly see the summary of our journey. To draw the graphic, we created a custom view and a java file linked to this custom view. In this java file we have one major method: *"onDraw"*. This method will allow the app to draw the graph when all the data will be retrieved from the first activity. The first mission is to create all the visual effect. With the *"DrawLine"* method, we make the first rectangle. Then we use the same method to create the (km/h, % of time) axe. After with the method "*drawText"*, we add the different text, specially the legend of both axes.

Then come the most important part of the graphic: Calculate where the different points will have to be. For this we choose to put 10 main points. We choose 10 points because we think that it shows a good evolution of the speed during the time. It is like ten highlights. To calculate the position, we use the size of the axe (which is 650dp), the size of the separation between each speed (55dp) and the different speed that has been stocked in a table during the journey. The formula is = 650 – speedTab[x] * 55. After that we have all the localisation of the 10 points, we draw some red cross (The highlights points) and after this we linked the different points with different lines.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<gpx xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
   <trk>
       <trkseg>
          <trkpt lat= 53.3304384 lon=-6.2786146>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304384 lon=-6.2786146>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304354 lon=-6.2786293>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304354 lon=-6.2786293>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304354 lon=-6.2786293>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304354 lon=-6.2786293>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304354 lon=-6.2786293>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304354 lon=-6.2786293>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304354 lon=-6.2786293>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304354 lon=-6.2786293>
               <ele>78.5999984741211</ele>
           </trkpt>
          <trkpt lat= 53.3304297 lon=-6.2786162>
               <ele>78.5999984741211</ele>
           </trkpt>
```

Figure 3: Example of a GPX file, stocked in the phone after the start button has been pushed

One of the main tasks of this project is the gpx file, which will contain information about the journey, like the latitude, the elevation and longitude. Firstly, to make a gpx file, we must ask for the permission. To do this, we must put the permission in the manifest file, and then with a dialogue message that pops up in front of the screen, we ask to the user if he allows the app to create and write files. To make this we simply use: *"ActivtyCompat.requestPermissions"*.

Then, on the *"onCreate"* method and on the *"startBtn.setOnClickListener"*, we firstly initiate the date (which is the title of the gpx file = "year, month, day hours: minutes: seconds") and secondly initiate the GPX File itself with the following path : "DOCUMENTS/GPStracks". After, using printWriter (variable call writer) we initiate basic stuff of the GPX file. This class allows to print different things in objects, here, in the gpx file.

When the gpx file has been initiated, we must write each 5 seconds the longitude, the latitude and the elevation. We implemented the *"Run"* method for that to. In this method, every data will be updated every 5 seconds, so using writers, we also update the gpx File.

The GPX File will be closed as soon as the button stop will be pushed. This operation is available thanks to the *"onCreate"* in which the stop button contains calls to function that will close the file (*"writer.close()"*).

In conclusion, this project taught us lots of new different things, especially using different class that we didn't know they existed. It was very interesting to create a gps application even if it a just a base. A more sophisticated application would have a map in background, and more precise data, but it was

not the main goal. The two different features are here to make the app more realistic and give more information to the user, which is always a good thing.