



**POLITECNICO**  
**MILANO 1863**

**Software Engineering II project**

ACADEMIC YEAR: 2018/2019



**Implementation and Testing  
Document**

VERSION 1.0 - 13/01/2019

*Davide Rutigliano - 903616*

*Claudio Ferrante - 903417*

*Davide Matta - 920349*

# Contents

<b>1</b>	<b>Purpose and Scope</b>	<b>1</b>
<b>2</b>	<b>Requirements and Functions</b>	<b>2</b>
<b>3</b>	<b>Adopted Development Frameworks</b>	<b>3</b>
3.1	Programming Language . . . . .	3
3.2	Frameworks, Libraries and Other Software . . . . .	3
3.3	API . . . . .	4
<b>4</b>	<b>Structure of the Code</b>	<b>5</b>
4.1	Server Code Packages . . . . .	5
4.2	Client Code Packages . . . . .	5
4.3	External Device Emulator Package . . . . .	6
<b>5</b>	<b>Testing</b>	<b>7</b>
<b>6</b>	<b>End User Installation Instructions</b>	<b>7</b>
<b>7</b>	<b>Developer Install Instructions</b>	<b>8</b>
7.1	Server Build and Installation . . . . .	8
7.1.1	Install PostgreSQL Database . . . . .	8
7.1.2	Build with IDE . . . . .	9
7.1.3	Build with Maven . . . . .	9
7.2	Client Build and Installation . . . . .	11
7.2.1	Build with Android Studio . . . . .	11
7.2.2	Build with Gradle . . . . .	12
7.3	External Device App Build and Installation . . . . .	13
7.3.1	Device Emulator Creation . . . . .	13
7.3.2	Device Emulator Set Up . . . . .	16
<b>8</b>	<b>Additional Implementation Notes</b>	<b>17</b>
<b>9</b>	<b>Effort Spent</b>	<b>18</b>

# 1 Purpose and Scope

This document is intended for presenting how all the features presented and designed in previous documents have been implemented [1][2]. In addition, the document be points out rationale for design choices for the adopted development frameworks.

This paper also contains a detailed description of the whole implementation both of client and server code and further testing following the *I&T plan*.

Finally, it contains all the installation and build instruction both for client and server, needed to run the code.

This document is intended for developers or anyone interested in the implementation details of *TrackMe*.

## 2 Requirements and Functions

This chapter is focused on highlighting requirements presented in previous documents, assuming the reader has previously read both Requirement and Design Document. In particular requirements implemented are from Data4Help and Track4Run services; AutomatedSos functionality instead has been not implemented because not requested by the customer.

With respect to the Requirement Analysis document, non implemented requirement are the following:

- **(REQ-18)** *The system must allow the Third-Party to activate Automated-SOS service;*
- **(REQ-19)** *The system must be able to assign an ambulance dispatcher to the SOS service;*
- **(REQ-20)** *The system must allow the Individual to activate Automated-SOS service;*
- **(REQ-21)** *The system should allow the Individual to choose between third parties who has enabled the service;*
- **(REQ-23)** *The system must be able to send the position to the nearest ambulance.*

Moreover, some requirements has been implemented slightly different from the RASD proposal:

- **(REQ-17)** *Data4help must allow the Individual to connect an external device through BT or NFC*

For the implementation, as well our choice for client application was Android, we used Google Play services (Android Wear Os) by developing a wearable application.

Rationale for this choice is primarily the fact that we do not know a priori which version of the protocol the external device uses, and most importantly we can't know how the output is syntactically: so it's impossible to generalize.

To solve this we would find a list of sensors that would be supported by TrackMe, analyzing each sensor output and standardize it to work with our environment.

This wasn't done because the acceptance test would be impossible if you don't have a listed sensor, so we decided to implement an Android Wear Os application that anyone can run in the Android Studio emulator.

Furthermore, as the application is for demo purposes, we intentionally did not implemented *"the connection between the client and server is encrypted and sent over SSL to guarantee integrity of data"*, as it was unnecessary for demonstration.

## 3 Adopted Development Frameworks

### 3.1 Programming Language

For both server and client side application the adopted language is Java: version 8 server-side, version 7 client-side; version 7 for mobile application is in order to support older OS version such as *Android KitKat (4.0)*. The rationale for this choice is that Java is more suitable for large-scale application with respect to other programming languages. In addition it has a large number of libraries and frameworks and it is also supported by android, needed for the client application.

#### Pros:

- Object Oriented language;
- Cross-Platform support;
- Proprietary implementation of Persistence API;
- Proprietary implementation of Authentication and Access Control;
- Test-driven development;
- Scalability;
- High Performance.

#### Cons:

- Code Verbosity.

### 3.2 Frameworks, Libraries and Other Software

#### Server-Side Frameworks

- Spring Framework: this choice is an alternative to Java Enterprise Edition and Enterprise Java Bean based models. The main advantage of this framework is indeed that allows programmers to write less code and to focus on critical parts of the application instead of "*simple and standard*" things. For instance, Spring Boot allows developers to create application with embedded web server such as Tomcat or Jetty with few lines of configuration without installing the web server itself. In addition, this framework also supports dependency injection and inversion of control. These latter features also allows developer to speed up and facilitate unit testing.

**Client-Side Libraries** In the Design Document we proposed Google for Map libraries, but since June 2018 Google Maps is not free anymore, we implemented our custom version of the map<sup>1</sup>, using following libraries:

- Osmdroid;
- Osmbonuspack;
- GraphView.

In addition, other client libraries are:

- Android Wearable;
- StompClient<sup>2</sup>.

#### **Other Software**

- PostgreSQL Database as database system.

### **3.3 API**

- Wear OS (Android Wear) API in order to make the mobile application communicate with an external devices such as a smartwatches.

---

<sup>1</sup>In a production system it may be better to use Google Maps API to manage maps as their libraries are widely used and tested with respect to a custom map implementation.

<sup>2</sup>For STOMP protocol, we implemented our set of libraries in order to make it work because public libraries available online works bad or does not work at all.

## 4 Structure of the Code

The application has been implemented following the design document, then structure of the code is almost the same presented into component and class diagrams, with little modifications skipped in the design document in order to not overload them.

Furthermore, the overall structure of both client and server application is organized in packages in order to improve reusability and maintainability of the code.

### 4.1 Server Code Packages

- **config:** spring configuration classes for security and web sockets
- **constant**
- **model:** contains two sub-packages, first spring entities used by repositories and second DTO classes related to entities
- **repository:** spring crud repository classes
- **service:** spring services, the business logic of the application
- **controller:** spring controllers, handles communications between server logic and client presentation layer
- **token:** contains spring security token authentication filter and token utilities

### 4.2 Client Code Packages

- **model:** models in the MVP pattern. Contains DTO classes (i.e java beans), that will contain all the data shared between server and client.
- **view:** views in the MVP pattern. Contains interfaces that provide the UI methods that will be implemented in every activity.
- **presenter:** presenters in the MVP pattern. Contains logic classes that handles the communication between server and client, and other business logic.
- **activity:** contains android activity classes that implements a view. Activities represent a single screen in the android environment.
- **fragment:** contains android fragment classes. They are similar to activities, but they can't exist on their own. They must be instantiated inside an activity.
- **service:** contains android service classes. Services are like background threads. They run even if the application is closed. They are used to send the position and the health data of all individuals to TrackMe.
- **httprequest:** contains all http logic to send and receive http messages.

- **task:** contains android asynchronous task classes. An asynchronous task is like a new thread, that will send a callback when completed. They are used by presenters to do the business and communication logic.
- **websocket:** contains all the websockets logic and the STOMP protocol implementation.
- **session:** contains classes that handles the application session.
- **util:** contains utility classes, for example a factory that builds a loading screen.

### 4.3 External Device Emulator Package

This package contains a small wearable application for *Google Wear OS* that allows the end user to connect and synchronize to, and acquire data from, his smart-watch.



## 5 Testing

Testing code is available within the project directory in "test" package. Server-side code has been tested using the tools provided by Java with some additional libraries.

In particular were used the JUnit to test the part that works in local without using external resources such as DBMS or network and Mockito, a popular mock framework which can be used in conjunction with JUnit.

The integration testing was done following the strategy described on the Design Document.

All the entry and exit criteria for each phase described on the Design Document were followed.

## 6 End User Installation Instructions

The client application is available in a pre-assembled version for Android devices (apk tested against Oreo 8.0 and Nougat 7.0).

This version of the application communicates with the last active deployment of the server application on Heroku.

Although it has not been tested against an hardware device (but working on built-in Android Studio emulator), we provided a pre-assembled apk also for the wearable application also for this part of the client package.

## 7 Developer Install Instructions

First of all you should install git and then clone *TrackMe* repository:

```
git clone https://github.com/DavideRutigliano/  
FerranteMattaRutigliano
```

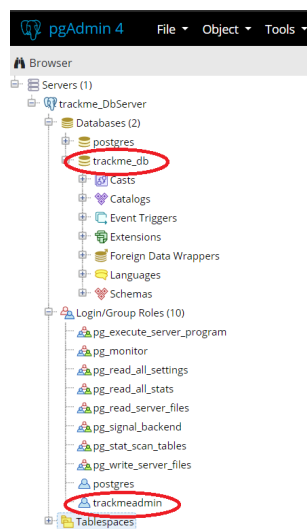
Then, to build and install the application, independently server or client, you can proceed in two different ways: build the application within your IDE or with Maven/Gradle wrapper.

### 7.1 Server Build and Installation

#### 7.1.1 Install PostgreSQL Database

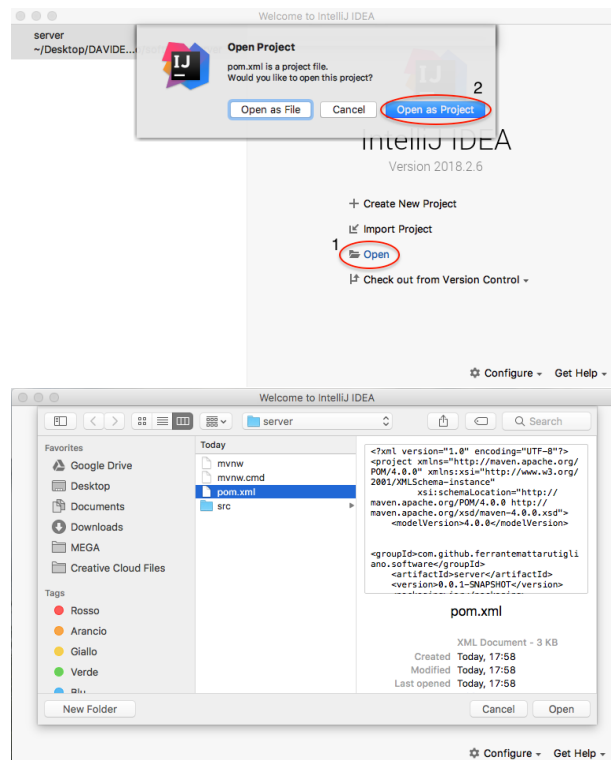
1. Download and install PostgreSQL. **Remember to write down/remember your password, you will need it later.**
2. Launch pgAdmin and create a new Server. Under the tab *Connection*, set Host name/address to: *localhost*. Press Save. **Don't edit the username** (it should be *postgres*) but add the password you used during the installation.
3. Right click on Login/Group Roles and create a new user: *trackmeadmin* identified by password *password*.
4. Alternatively you can use an existing user changing *spring.datasource.username* and *spring.datasource.password* in *application.properties* in the server Java application.
5. In the tab *Privileges* check *Superuser* and *Can login?*. Press Save.
6. Right click on Databases and create a new database called *trackme\_db*.

You should have the following configuration at the end:



### 7.1.2 Build with IDE

1. Open as a project: "*ferrantemattarutigliano/software/server/src/pom.xml*" and wait until import is complete;
2. Setup project sdk: file -> project structure -> project sdk (you need at least project language level 11);
3. Add configuration -> application -> main class: *ServerApplication*.



### 7.1.3 Build with Maven

With Maven wrapper you can build, install and run the application without installing any other software component, as Maven will download it for you from Maven central repository.

**Unix:** in order to build *TrackMe* server with Maven you do:

```
cd FerranteMattaRutigliano/software/server
sudo chmod +x mvnw
./mvnw package
```

**Windows:** in order to build *TrackMe* server with Maven you do:

```
cd FerranteMattaRutigliano\software\server
mvnw.cmd package
```

The script should return something like:

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 8.658 s  
[INFO] Finished at: 2018-12-28T20:16:17+01:00  
[INFO] -----
```

Then, you can run the application:

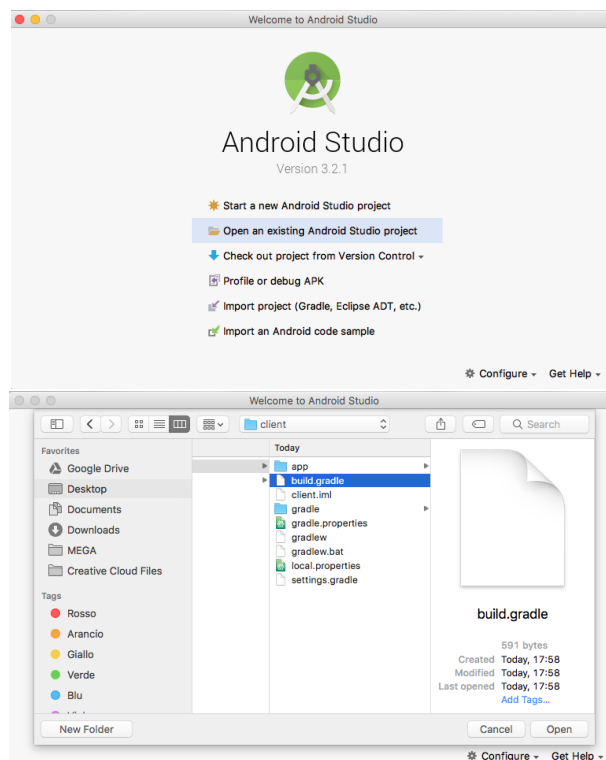
```
java -jar ./target/server/server-0.0.1-SNAPSHOT.jar
```

## 7.2 Client Build and Installation

In order to build client mobile application within an IDE, you should use Android Studio.

### 7.2.1 Build with Android Studio

1. Download [Android Studio](#);
2. Install android studio and Android Virtual Device;
3. Open an existing android studio project:  
"ferrantemattarutigliano/software/client/build.gradle" and wait until import is complete;
4. Select file -> project structure. Make sure that Android NDK is installed;
5. Run the application and create a new virtual device (you can choose the default one, Nexus 6). Select an API with version > 15;
6. If you want to use a local server, uncomment the appropriate line in *app/java/com/github/ferrantemattarutigliano/software/client/httprequest/HttpConstant.java*.



**Known Error** Emulator: ERROR: x86 emulation currently requires hardware acceleration!

**Solution:** tools –> SDK Manager –> SDK Tools (tab) and make sure that Intel x86 Emulator Accelerator is installed. If not, install it; if already installed, uninstall and re-install it.

### Important Notes

- When you create the android SDK directory, don't use white space or '-' characters. This causes problem with the Android framework;
- If you prefer to launch the application on your own android device you can do that: enable USB debug on your android phone and connect it to your PC with an USB cable.

### 7.2.2 Build with Gradle

With Gradle wrapper you can build, install and run the application without installing any other software component, as Gradle will download it for you from Maven central repository.

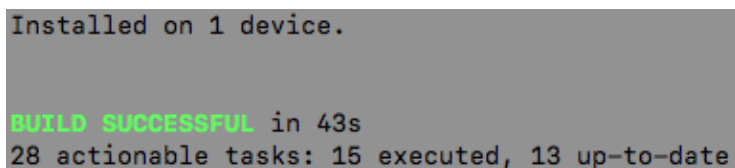
**Unix:** in order to build *TrackMe* server with Gradle you do:

```
cd FerranteMattaRutigliano/software/client
sudo chmod +x gradlew
./gradlew installDebug
```

**Windows:** in order to build *TrackMe* server with Gradle you do:

```
cd FerranteMattaRutigliano\software\client
gradlew.bat installDebug
```

The script should return something like:



```
Installed on 1 device.

BUILD SUCCESSFUL in 43s
28 actionable tasks: 15 executed, 13 up-to-date
```

## 7.3 External Device App Build and Installation

In order to run the external device application you should have a physical smart-watch or an emulator (available within *Android Studio*'s virtual device manager).

Build and Installation instruction for this code package will not be listed as they are exactly the same steps for building the client application (i.e. using gradlew or Android Studio), obviously changing build files and packages.

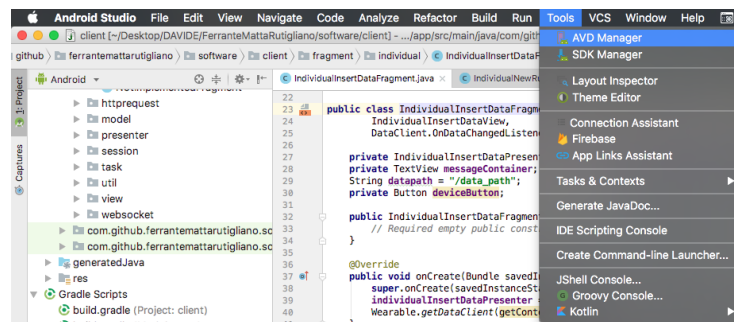
If you have an external device such as a *Gear-Fit* or similar android devices following steps are not needed.

**Important Note:** The source code included in this package has been tested only against a Square Wear OS smart-watch emulator with Android Oreo OS and API version 26. Any other run of the application on other device could lead to unexpected behaviour as it has not been tested.

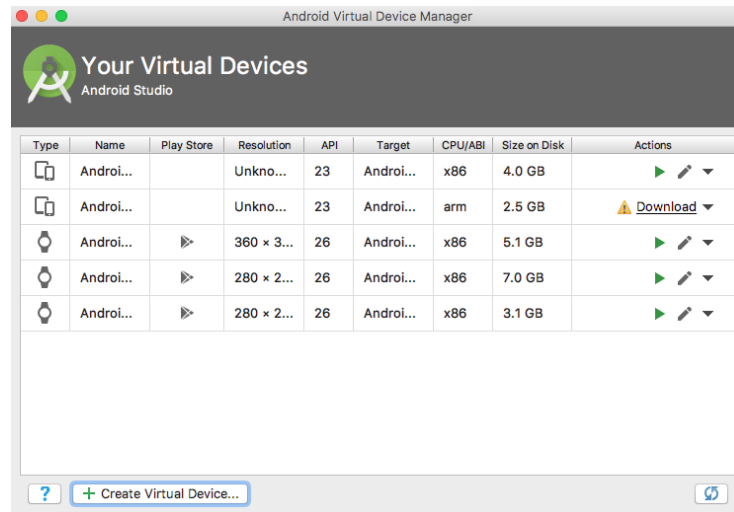
### 7.3.1 Device Emulator Creation

In order to run the application on an emulator, you should first create it from android studio.

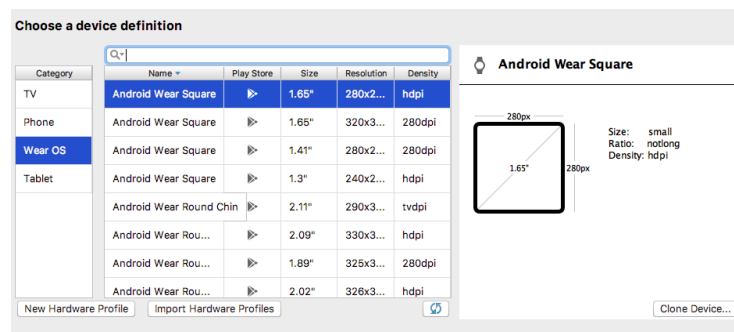
- Click on Tools, then AVD Manager;



- Click *Create Virtual Device*;

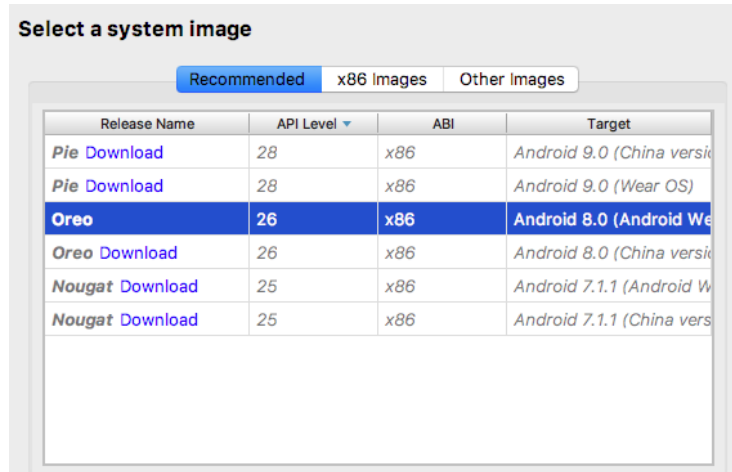


- Select *Wear OS* from category menu, then *Android Wear Square*;



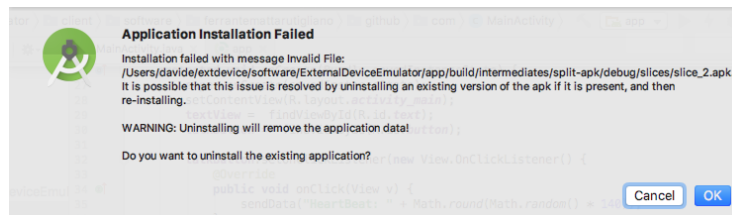


- Select Android OS version;

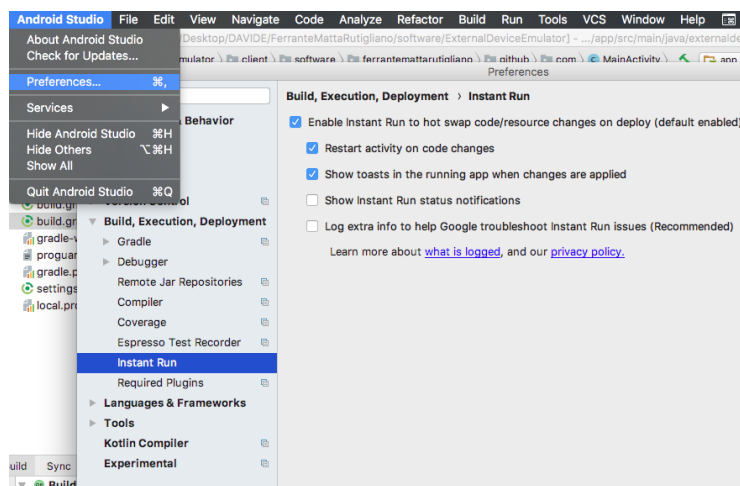


Then you can run your *Wear OS* virtual device from AVD manager clicking on play button.

If during run of the application you get the following error:



Then you should uncheck *Enable Instant Run* feature from Android Studio preferences (Build, Execution, Deployment):



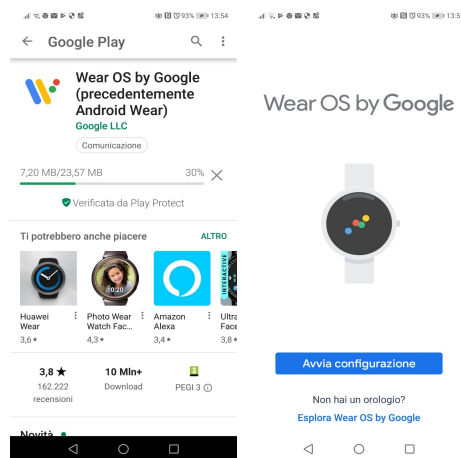
### 7.3.2 Device Emulator Set Up

In order to allow the emulator communicate with the smartphone, you should connect via USB (with debug enabled) the phone and tell the *Android Debugger* to forward requests from the phone to the emulator and vice-versa using the following command:

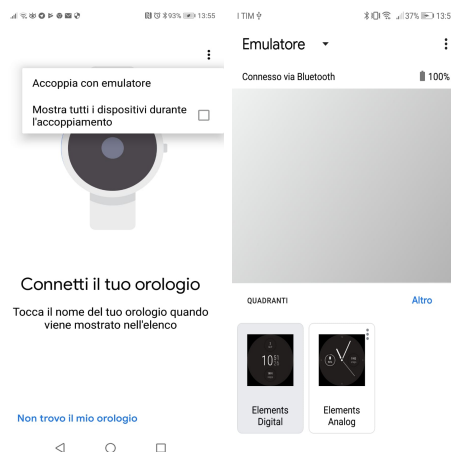
```
adb -d forward tcp:5601 tcp:5601
```

On your mobile phone, you should:

- Download and Install Google Wear OS Application, then start configuration;



- Once configuration has finished, pair the mobile phone with the smart-watch emulator;



## 8 Additional Implementation Notes

- TrackMe is deployed on Heroku, a free cloud application platform. Using the TrackMe end-user version (i.e using the provided apk version or if the ip server address is not changed in the developer version), the client application could give a timeout warning or an infinite loading screen. To solve this simply redo the request.
- Further implication of using a smart-watch emulator is that health data are merely random numbers due to the lack of sensors for Android Studio emulator; in a real word application this mechanism should work listening to smart-watch's sensors using Android Wear services.
- When you create or view a map on Track4Run, sometimes you can see that the route between markers is not calculated and it's simply a straight line between the markers. This is caused by our (free) map library osmdroid. It simply cannot reply to too many requests, so sometimes the route will not be calculated.

**NOTE:** Due to the demonstration purpose of this version of the application only free and open source libraries were used; in a production environment these kind of problem should not arise by using well working and tested commercial API.

## 9 Effort Spent

- Davide Rutigliano: 120h
- Davide Matta: 120h
- Claudio Ferrante: 80h