Politecnico di Milano
AA 2018/2019

POLITECNICO
MILANO 1863

Software Engineering 2

TrackMe

Implementation and Test Document

Stefano Pecchia
921201

Edoardo Peretti
921286

Deliverable:	ITD
Title:	Implementation and Testing Document
Authors:	Stefano Pecchia and Edoardo Peretti
Version:	1.0
Date:	January 13, 2019
Download:	https://github.com/Speck1996/PecchiaPeretti
Copyright:	Copyright © 2019, S. Pecchia and E. Peretti – All rights reserved

Contents

1 Introduction

1.1 Introduction

In Data4Help implementation and testing document the process of the system creation from the back end to the front end is analyzed. Before continuing the reading of this document it is highly suggested to read the RASD and the DD to better acknowledge the development decisions. Data4Help is a service strongly based on the server side, whereas the apps for the users are used to access the services provided by the system. This will be a key point to keep in mind in order to better understand the ordering of feature implementations and so on.

1.2 Scope

The scope of this document is to provide a general overview of the structure of the code, while referring mainly to the DD for the code design and testing decisions, and the RASD for the features implemented. Regarding the features implemented, the document will go deep in details in explaining what are the features implemented and why some features where dropped during the development of Data4Help. Moreover a guide to successfully install the app, and starting the server will be provided with insights on their user interfaces and functionalities.

1.3 Acronyms

- **API**: Application programming interface
- **DD**: Design Document
- **DB**: Database
- **EJB**: Enterprise JavaBeans
- **RASD**: Requirements Analysis and Specifications Document
- **REST**:Representational State Transfer
- **MVC**: Model View Controller pattern

1.4 Abbreviations

Rn : n-th functional requirement

1.5 Document Structure

The ITD is structured in the following way: it starts with a brief of description of the document and an introduction in Chapter 1.

Chapter 2 is focused on the requirements implemented: in this chapter there is a general description of the decision made during development and how and where the requirements are realized in the source code.

Chapter 3 presents the frameworks used during development.

Chapter 4 has a general overview of the source code structure.

Chapter 5 describes the testing process used for Data4Help components.

Chapter 6 presents the installation instructions for both the Server and the Android App.

Chapter 7 contains the references cited in the document.

Chapter 8 show the effort spent by each group member in order to realize the ITD document and Data4Help system.

2 Requirements Implemented

The developing of the system started with the design and the implementation of the DB. As stated in the DD, the server is considered as the main component in whole system. This is because it handles both the controller and the model in the MVC pattern used to build Data4Help. This also is coherent with what was stated in the RASD, where Data4Help is described as a central hub for eHealth data.

After finishing the DB, the development of the RequestManager component started. This component handles all the necessary operations to distribute and perform data requests to the DB. In particular in addition to the data retrieving it checks for privacy constraints for both individual and group requests, and it is also in charge of transforming location string filters in coordinates, using an external service: Nominatim.

Then the next item to be implemented was the DataCollector: this component takes individual's data and inserts it into the DB.

After having the eHealth data handling components ready the development of the login service started. The login service handles the login process for the WebApp and the Android App by using tokens. Tokens are generated by the server and stored by the clients, that have to provide the token each time they want to access a resource that requires authentication.

Once the login service was ready the android app implementation started, from the login itself. Some of the app features provided in the RASD were not implemented: for example there is no way to simulate sensors eHealth data from emulated smartwatches and neither one of the two team member has an android wearOS smartwatch. During the development there was an attempt to gather the data from google fit servers using Google Fit History API, but since Google itself states in the documentation that the data gathered is not reliable, since there is a mismatch between the Google Fit App and the google fit services, the feature was dropped for impossibility of testing it, because the data visualized in the app was different from the one sent by Google servers. This lead to the decision of not implementing the data history visualization process for AndroidApp, as it was mainly focused on keeping real data coming from smartwatches on Data4Help DB, so that individuals could have a backup of their eHealth Data independently from the owned smartwatch. The data gathered indeed is just some random data distributed with a Gaussian, with its parameter settable from a proper screen. Then the requests are fully functioning with the user that can accept or refuse requests.

In the end the WebApp was developed in all his functionalities allowing third parties to make and visualize their requested data.

The following list presents implemented requirements from the RASD with a brief description on where to find the source code that realizes them:

- **[R1] Users are able to login with the username and the password associated to their account**: The code that implement this feature can be found in the login package.
- **[R4] Individuals can upload their data through Data4Help app**: The code of this feature can be found in MyDataActivity Class of the AndroidApp. Since the smartwatch pairing was not implemented, the data is randomly generated.

- **[R4.1] Data4Help is able to store the data provided by individuals**: This is one of the most important features of Data4Help, and it is implemented in the collector package by the DataCollecor, with the help of a MySql DB.
- **[R4.2] Data4Help is able to organize data provided by individuals**: This feature is implemented in the model package where each kind of data has its own class, that corresponds to a table in the DB.
- **[R5] Third parties can formulate requests to access anonymized data of groups**: This feature is implemented by the WebApp with the help of EJB that can be found in the manager package.
- **[R6] Third parties can apply filters on data while formulating their request for data of groups**: Feature implemented in the WebApp.
- **[R7] The system checks if the number of individuals in the group detected by the third party request is higher than 1000**: This requirement is implemented by the GroupRequestManager class, in the manager package.
- **[R7.1] If the groups has 1000 or less individuals the system denies the request**: Feature implemented in the GroupRequestManager.
- **[R8] The system is able to distribute the requested data to the third party**: This functionality is provided by the WebApp with the help of EJB.
- **[R9] Third parties can formulate requests to access specific individual data, indicating the individual's TC**: This feature is implemented by the WebApp and the IndividualRequestManager, in the manager package.
- **[R10] Third parties can apply filters on the type of data while formulating their request for individual data**: This functionality is implemented by the WebApp.
- **[R11] Data4Help is able to forward the request to the individual specified by the third party**: The code of this feature can be found in the RequestFragment in the AndroidApp and in the individual request manager for what regards the server part.
- **[R12] The individual to whom the request will be forwarded is able to reply**: Implemented in the RequestFragment of the AndroidApp
- **[R12.1] The system notifies the third party about the individual reply: Feature implemented by the WebApp**: the WebApp doesn't have a real notification system, but just displays new data when it is available.
- **[R13] The system can check if the individual has accepted the data access request by the third party**: Feature implemented in the IndividualRequestManager through the monitoring table in the DB.
- **[R14] Third parties can specify to be updated whenever new data of groups detected by their data requests, or observed individuals is gathered. The time interval between an update and the next one can be set by the third party, in a limit that goes from one hour to 1 year**: Feature implemented by the WebApp, but with most frequent update that is one week.

- **[R15] The system is able to update third parties with new data, respecting their preferences on the interval timing**: Feature implemented by the WebApp, where the third party can choose the update frequency
- **[R21] The system doesn't allow third parties to access specific individuals data without first asking for their permission**: Functionality implemented by the IndividualRequestManager
- **[R22] The system stops to update third parties about new data whenever observed individuals decide to remove data access permissions, or whenever the number of individuals of an observed groups goes belove 1001**: Feature implemented by the IndividualRequestManager and by the GroupRequestManager.
- **[R23] Only users that know their username and password can access to their respective accounts**: Functionality implemented in the login package.

The following list instead contains the functionalities that were not implemented, or that were dropped during the development phase:

- **[R2] Data4Help is able to pair with the user wearable**: This requirement was not realized, because emulated smartwatches can't emulate some sensors like the heart rate sensor, making the pairing useless since no eHealth data can't be retrieved from emulated smartwatches.
- **[R3] Data4Help is able to download eHealth data from the individual's wearable**: As for the pairing requirement, this feature can't be implemented with emulated smartwatches.

The other requirements that are not present in these lists are part of the AutomatedSOS service, whose implementation was not required.

3 Adopted frameworks

The project has been developed with the programming language Java, using several specifications of Java EE 8. In particular the following specification have been adopted:

- Servlet 4.0
- EJB 3.2
- JPA 2.2
- JTA 1.2
- JAX-RS 2.1
- JSF 2.3
- Managed Bean 1.0

The Java EE 8 specifications are implemented by the application server Glassfish 5.

For the geocoding service provided by the manager an external API was used: Nominatim API. The choice of this API was based on the fact that is the only one entirely open source, and that doesn't require any subscription or payment even if it has a strong limitation on the requests that can be made in a certain amount of time and a documentation only based on Javadoc. For more information please consider reading Nominatim Policies linked in the references section.

Moreover, an Android application has been developed, again by means of the Java programming language. The frameworks used by the Android App are all part of the android native package with the exception of Retrofit 2.5. This APIs makes the handling of HTTP communication with the server faster, with its integrated API parsing response. For more information consider visiting Retrofit home page, linked in the references section.

4 Structure of the source code

The source code is clearly divided in two parts: the part that implements the TrackMe server and the part that implements the Android application.

4.1 Web server and Web App

The source code of the server side is subdivided in the following packages: `collector`, `login`, `manager`, `model`, `webapp`.

Collector In the `collector` package is located the fundamental `DataCollector` class whose task is, basically, to receive the data from the Android application and persist it in the database.

Login In the login package are located all the classes that implements functionality related to login, signup and authorization.

Manager In the `manager` package is located the logic that manages the request, forwarded by third parties by means of the web app, and the visualization of data. It also contains a sub package, called `geocoding`, in which are present interfaces and classes responsible for the maps external API.

Model The `model` package contains all the entities used in the Object-Relation Mapping, exploited by means of JPA specification. It contains also the definition of the enumeration and constant used in the database. In the sub package called `xml` are located some classes useful for marshalling data into xml.

Webapp In the `webapp` package are located the managed beans that constitute the back-end of the webapp (excluded those regarding login and signup that are located in the login package).

Besides the Java source code, the server also contains, in the `web` folder, all the front-end of the webapp developed using JSF alongside to some CSS.

4.2 Android

The source code of the AndroidApp is divided in the following packages `activities`, `model`, `retrofitclient`, `adapter`, `helpers`.

Activities In this package all the classes that handles view operations and rendering can be found. In particular to each activity or fragment corresponds a screen UI of the Android App.

Model In the `model` package there are the classes needed to communicate with Data4Help server.

Retrofitclient In the package `retrofitclient` there are the interface used to communicate with the server, and the class in charge of building the RetrofitClient.

adapter In the `adapter` package there is the class in charge of rendering the requests view, that are then used in the RequestFragment that can be found in the `activities`.

helpers In the `helpers` package there are utilities needed by the activities: an encryptor used to encrypt passwords in the login and signup activities and a DatePicker used to have a better view experience in the signup for what regards date input.

5 Testing

5.1 System test cases

Once the development of the server and of the webapp have been overall completed, several manual test cases have been performed. In particular, the following use cases have been successfully tested:

1. Third party login.
2. Third party signup.
3. Submission of a new individual data request.
4. Submission of a new group data request.
5. Visualization of all the submitted requests (either individual and group)
6. Visualization of the data of the group request (when sufficiently numerous to be anonymized) and individual request (if accepted by the individual).
7. Subscription and deletion of requests (either individual and group).

In the development phase of the REST API layer needed for the communication with the Android App for what regards the login process and the data sending/downloading from the server, an external tool was used, Postman, in order to make http client requests without developing a full client.

For what regards the Android App the testing phase used mainly black box texting. This is due to the fact that the app only handles the view parts and requires a persistent connection in all its functionalities. Indeed the unit tests that were developed initially were only assertion on the input fields or click listening with a mocked server, and due limited time they were dropped in favor of black box testing integrated with the whole system.

6 Installation instructions

The following software are needed

- MySql Community Server 5.7.24 or MySql Community Server 8.0.13
- MySql Connector/J 5.1.47
- MySql Workbench 8.0.13 (Optional)
- JDK 8 Update 144
- Glassfish 5.0

6.1 Database initialization

After having installed MySql Server, you have to create the database for trackme. In the repository you can find 3 sql file to do this:

1. `create.sql`
2. `sample.sql`
3. `bigsample.sql`

As their name suggest, the first one create the database and the tables, the second populated the tables with some small data, while the last one insert a larger, randomly generated, amount of data. Notice that running any of these files will drop the existing data currently stored in the database.

In order to execute those sql you can type in the terminal

```
mysql -u your_username -p < you_file.sql
```

and enter your password. Alternatively, you can use MySql Workbench with *File/Run sql script*.

Please note the individual's password of the samples are not hashed, so when trying to login on the Android App be sure that the encryption is disabled, as explained later in Section 6.6

6.2 Connection pool configuration

Assuming *domain1* is in use on glassfish, copy `mysql-connector-java-5.1.47-bin.jar` in the folder

glassfish5/glassfish/domains/domain1/lib

Now follow the next steps.

1. Start glassfish server.
2. Open the administration panel (usually on port 4848).
3. Select *Resources/JDBC/JDBC Connection Pools* on the tree on the left.

4. Click *New...*
5. In the *Pool Name* field enter `TrackmePool`.
6. As *Resource Type* select `javax.sql.DataSource`.
7. As *Database Driver Vendor* select `MySql`.
8. Click *Next*.
9. Make sure that *Data Source Classname* is `com.mysql.jdbc.jdbc2.optional.MysqlDataSource`.
10. In *Additional Properties*, you can delete all the default ones and set the following:
    - `portNumber`: the port of your mysql server, usually `3306`;
    - `user`: your username for the mysql server;
    - `password`: your password for the mysql server;
    - `databaseName`: enter the value `trackme` as defined in the SQL DD.
11. Click `Finish`.

Now your database connection pool should be configured, you can check selecting the connection pool and clicking *Ping* (be sure that the MySql server is running). A *Ping Succeeded* message should show up.

In order to allow the application to use this connection pool, a new JNDI resource must be created. From the glassfish administration panel:

1. Select *Resources/JDBC/JDBC Resources* on the tree on the left.
2. Click *New...*
3. As *JNDI Name* enter `jdbc/trackme` (note that this name is important, if different the application will not be able to locate the resource).
4. As *Pool Name* select the `TrackmePool` just created.
5. Click *OK*.

For more details or if you encountered other problems, you can refer to the administration guide of Glassfish linked in the references section.

6.3 Certificate issue for Nominatim API

It can happen that the call the to Nominatim API fails due to certification issue. This problem can be noted in the Glassfish log file when trying to perform a call for a location coordinates. If the problem is not fixed, the location constraints for group data request are simply ignored.

A possible fix is to use the certificates distributed along with the JDK 8. From the installation folder of the JDK, go to Contents/Home/jre/lib/security and copy the `cacerts` file. Go to the Glassfish installation folder and then, assuming domain1 is in use, to glassfish/domains/domain1/config. Here there should be an already exiting `cacerts.jks`, replace this file with the previously copied from the JDK (adding the extension could be necessary).

6.4 Web App Deployment

After downloaded the `trackme.war` from the repository, follow the next steps in order to deploy the application. Be sure that the mysql server is running and the connection pool properly configured.

1. Start glassfish server.
2. Open the administration panel (usually on port 4848).
3. Select *Applications* on the tree on the left.
4. If not already selected, select *Packaged File to Be Uploaded to the Server* and choose the `trackme.war` previously downloaded.
5. In the *Context Root* field enter `trackme`.
6. Click *OK*.

Now the trackme server should have been correctly deployed, to verify this, go to

http://localhost:8080/trackme/login.xhtml

the login page (or the homepage if the browser sent a valid authentication cookie) should appear.

6.5 Android App deployment

To install the app on your android phone first enable the installation from unknown sources. Then download the app from the delivery folder in your phone, find it with a file manager and tap on it. Allow the installation and wait for the process to finish. The app should now have been installed.

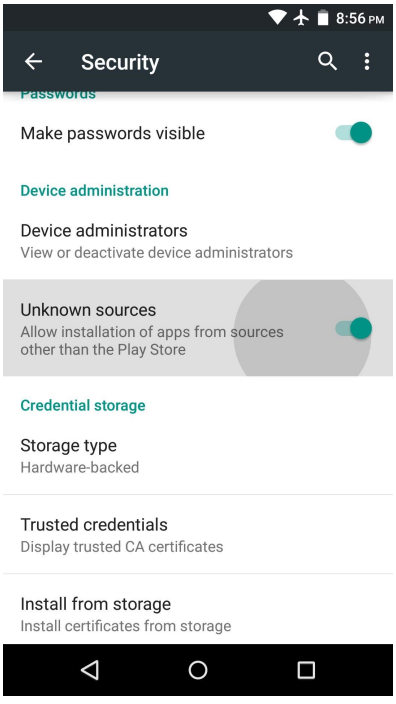

Figure 1: Unknown sources screen

Alternatively the app can be installed on an android studio emulator, that comes bundled with android studio. First thing that is necessary to deploy the app on your emulator is to have ADB installed in your system. Mind that each OS has its own version of ADB, so make sure to install the correct one. Next download the app and move it to android\sdk\tools or platform-tools folder. The path to these folders changes based on the OS. Open the terminal and go to the folder containing the apk then execute the following command: adb install apkname.apk. The apk should now have been installed.

Note: In case of multiple emulator error, run the command adb devices, pick the id of your emulator (it is usually emulator-5554) and then run the command adb -s <your emulator id> install <apk name> from the folder where the apk was moved before.

6.6 Setting up the App

By tapping more than 10 times on the App logo in the main screen the user can have access to the developer screen.

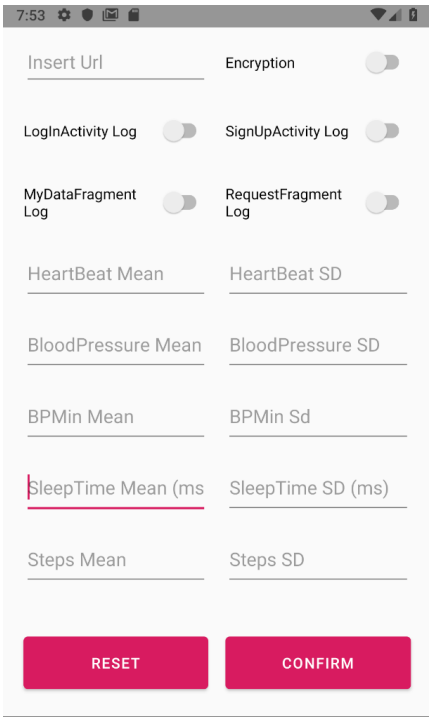

Figure 2: Developer Screen screenshot

As can be seen from the image various settings can be specified, but the most important is the serverl url: by default the url provided bundled with the apk is the one that lets the emulator connect to the localhost. Of course this cannot be used if the App has been launched on the phone, so in the Url text field it should be specified the host url of the server. This corresponds to http://ipv4netip:8080/trackme/rest/ in the case that both the smartphone and the device running the server are under the same network, where the ipv4netip is the network ip of the machine that runs the server. Moreover, to use the preloaded individual tuples that come with the DB, the encryption should be disabled (it is disabled by default) since that the password contained in the DB are not hashed. In the end some other parameters like activating various log for the activities, or setting up the parameters for the Gaussian distribution for each data can be set. To apply the changes

tap on the confirm button. In the end remember that this choices are kept as long as they are not cleared by tapping on the reset button in the developer screen.

7 References

- JDK 8 update 144 https://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html
- MySql Community Server https://dev.mysql.com/downloads/mysql/
- MySql Connector/J https://dev.mysql.com/downloads/connector/j/
- MySql Workbench https://dev.mysql.com/downloads/workbench/
- Glassfish 5.0 https://javaee.github.io/glassfish/download
- Glassfish Administration Guide https://javaee.github.io/glassfish/doc/5.0/administration-guide.pdf
- Google Fit History Api synchronization problem https://developers.google.com/fit/faq
- Retrofit https://square.github.io/retrofit/
- Nominatim policy https://operations.osmfoundation.org/policies/nominatim/
- ADB install https://www.xda-developers.com/install-adb-windows-macos-linux/
- Android Studio https://developer.android.com/studio/
- Postman https://www.getpostman.com/

8 Effort spent

In the following tables the time spent for each section of the project is presented. For the efforts spent on user interfaces, please refer to the RASD.

Stefano Pecchia

Section	Hours
Geocoding	6
Login Token Authentication and Signup	12
AndroidApp with server integration	60
BlackBox testing for AndroidApp	15
IDT document	6

Edoardo Peretti

Task	Hours
Database and JPA	25
Server	20
Web App	35
BlackBox testing	15
IDT document	6

