Politecnico di Milano
AA 2018-2019

Computer Science and Engineering
Software Engineering II

Implementation and Testing Document

Stefano Bagarin
Alessandra Pasini

`https://github.com/stepolimi/BagarinPasini/tree/master/ITD/ITDProject`

Document version: 1.0
January 20, 2019

Contents

Section 1

Introduction

1.1 Scope

The aim of this document is to provide an overall description of the two prototyped applications that has been developed by our group.
This document contains information related to all implemented functions, some of the limitations present in them and the way a stackholder must behave to make the apps correctly run; some more details of a few APIs that have been used and that were not already considered in the DD; the code structure, testing details and the exact procedure that must be applyed to correctly open and make the software work.

1.2 References

This document is strictly based on the specification concerning the Implementation and Testing assignment for the Software Engineering II project, part of the course held by professor Matteo Rossi and Elisabetta Di Nitto at the Politecnico di Milano, A.Y 2018/2019. It refers to the implemented code which refers to the RASD and DD previously produced.
More details about RASD and DD can be found in the in the GitHub repository, the code can be found in the same repository and at the link in the front page.

Section 2

Software Functions

2.1 Implemented functions in both Users and Third Parties' app

2.1.1 Registration

The registration function has been implemented for both stackholders. This function has been considered foundamental not just to develop a personal area for each user or third party, but also to obtain all users' personal data necessary to evalue them in relation to data's requests of groups of users.
This function is also convenient to obtain an authentication code which will be necessary to all future http request sent from clients to the server.
NB. Bacause of the fact that this is just a prototype, in the user registration the only country which will be accepted is *Italy* and, for this reason, the only accepted regions will be the italian ones. The app speaks in english so everything will need to bee written in english in order to work.

2.1.2 Login

Also the login function has been implemented for both stackholders as a collateral effect of the registration one. Because of the precence of a registration step and a personal area the login has been developed for a..
Every time a user or a third party does the login, the authentication code is sent from the server.

2.2 Implemented functions in Users' app

2.2.1 Add height and weight

In the settings area which can be found in the menu it is possible to add height and weight, but not a wearable device. This function has been only half completed because none of the group members have a wearable device. A user must add his/her measures in order to start sending his/her personal health parameters to the server.This function not only follows the working flow of a well known health app called *Health*, but it is also a good base point for the future development of the ASOS service. Given age, sex, weight and height parameters it is possible to implement a good algorithm to evaluate the threshold below which the user can be consider in life danger.

2.2.2 Send health parameters to the server

Although it is not possible to add a wearable device, it has been implemented a thread in which random values are sent to the server as the user's health parameters. This function has been implemented to simulate the DB's filling and in order to be a good starting point for the future possibility to connect via bluetooth a wearable device to the app. All sent datas are randomly chosen between normal real health parameters which don't represent a dangerous status for an adult.

2.2.3 See personal health parameters

Every user, in his/her personal area, can have access to his/her personal data. This is the users' app main function and, for this reason, it has been implemented in the better possible way by givin the possibility to see not only the daily health parameters but also the weekly, monthly and year ones. The shown health parameters are the minimum, average and maximum heart beat; the minimum and maximum vein pressure and the minimum and maximum temperature detected till that moment. In this activity are also reported weight and height in order to give to the user a general idea of his/her health behaviour.

2.2.4 Accept or deny Third Parties data's requests

This is a collateral function strictly related to one of the most important functions in the other stackholder's app. Because of the third parties' possibility to require single users' data , it has been found necessary to give to the

user the possibility to choose to accept or deny the request. By clicking on the get Notification Button in the apposite fragment, the user will obtain a pop up strictely related to a request. His/her answer will be sent and stored in the DB without any consequence for the user app.

2.3 Implemented functions in Third Parties' app

2.3.1 Require data of group of users

This is one of the main functions of the third parties stackholder and, because of this reason, it has been found necessary to implement it. The third party can choose how much to restrict the request by adding a few constraints such as the minimum and maximum age; the minimum and maximum weight; the minimum and maximum height; the address (state, region, city, cap, street, number) and the sex (male, female or both).

2.3.2 Require data of single user

Require data of single users is the other fundamental function with the one of requiring data of groups of users. The third party can ask for receiving data of a single user by giving the fiscal code or the id number (depending on the country in which the user live) and by adding the reason why it wants to get the health parameters.

2.3.3 See data of single users

This function has been implemented in order to give the possibility to the third party to see the health paramters related to an accepted single user request. The third party has the possibility to see the minimum, average and maximum heart beat; the minimum and maximum vein pressure and the minimum and maximum temperature detected in that particular day and in the 6 days before.This function has been realized in this way in order to prototype a future behaviour in which the third party will have access to all health parameters present in the calendar.
This function has two different behaviour depending on the third party choice to subscribe or to not subscribe for receiving more user's data:

- If the third party has chosen to subscribe to a particular user, once data will be show they will also be periodically reloaded and it will be

possibile to see the updated health parameters. It will also be possibile in this case to find a button in the *Subscribed Data* fragment that will let the third party to get again the user data any time the stackholder will need to access to the user data.

- If the third party has chosen to do not subscribe to a particular user it will just be able to see the user data at that precise moment and then won't have anymore the possibility to acced to them.

2.3.4 See data of group of users

This function has been implemented in order to give the possibility to the third party to see the health paramters related to an accepted request of a group of users. The third party has the possibility to see the minimum, average and maximum heart beat; the minimum and maximum vein pressure and the minimum and maximum temperature detected in that particular day and in the 6 days before.This function has been realized in this way in order to prototype a future behaviour in which the third party will have access to all health parameters present in the calendar associated to a particular group of users.
This function has two different behaviour depending on the third party choice to subscribe or to not subscribe for receiving more data:

- If the third party has chosen to subscribe to a particular group of users, once data will be show they will also be periodically reloaded and it will be possibile to see the updated health parameters. It will also be possibile in this case to find a button in the *Subscribed Data* fragment that will let the third party to get again the group data any time the stackholder will need to access to them.

- If the third party has chosen to do not subscribe to a particular group it will just be able to see the data at that precise moment and then won't have anymore the possibility to acced to them.

2.3.5 Subscribe or not for receiving new data of single users

In order to develop the subscribe function it has been given to the third party the possibility to choose to subscribe or not to the single user request. Because of the time that can pass between the sending of a request to a user and the user answer, it has been implemented a fragment *Waiting Data* in which all users' positive answers are saved in buttons. Once the third party

clicks on one of those button, a subscribe pop up will be shown.
There are two possibile behaviours:

- if the third party clicks on the *Refuse Button* the health parameters will be shown in the not subscribed modality.
- if the third party clicks on the *Accept Button* the pupup will be delated and the stackholder will have to go the *Subscribe Data* fragment and click on the associated button.

2.3.6 Subscribe or not for receiving new data of group of users

In order to develop the subscribe function it has been given to the third party the possibility to choose to subscribe or not to the group request as soon as the server has stated that there are enough users which respected the constraints.
There are two possibile behaviours:

- if the third party clicks on the *Refuse Button* the health parameters will be shown in the not subscribed modality.
- if the third party clicks on the *Accept Button* the pupup will be delated and the stackholder will have to go the *Subscribe Data* fragment and click on the associated button.

Section 3

Adopted Development Frameworks

Java has been used for programming both the server and the client, in particular we have used Android Studio for developing the client and Ultimate IntelliJ for the server.

As already said in the DD JPA and JAX RS APIs has been used; to better know the reason why they have been chosen please read section two of the DD. In this document it is important to highlight that JPA and JAX RS provided by the chosen server Apache TomEE have been used; this means that we worked with OpenJPA and Apache CXF.
Jackson API has been used to transform in a simple way Json objects send from the Client via HTTP in Java classes. To use it, it is necessary to map every json value to one of the attributes of the related Java class.
Volley API has been used in the client to create in simple way HTTP request containing Json Objects and to easily receive the server response by overriding the ParseNetworkResponse method.

Section 4

Code Structure And Testing

4.1 Code Structure

The whole project has been devided in three different subproject:

- **Server:** the server has been implemented in a Maven project with JavaEE specification. It is formed by:
  - model classes, which are all classes mapped to percistence elements by the OpenJpa API. Those classes contains only getter and setter methods;
  - DAO classes have the objective to interact with the DB by requiring, adding, delating and updating data;
  - service classes which contains all the REST methods and the agorithms related to internal operations;
  - support classes to do all stuffs in a well written way.
- **Client:** both clients have been written in Android studio creating two different gradle projects. Both are formed by:
  - activity classes which have been used has major containers for fragment and which have the aim to do all basics operation required;
  - fragment classes implement all functions required by fragments, evere class layed down on an activity class and is usually called by a menu class or a pagerView class;
  - dialog fragment classes implement dialog fragments which are shown when some particular occasions occurs;

– support classes to do all stuffs in a well written way.

For more details it is possible to read the Design Document in which class diagram, sequence diagrams and more information related to the overall structure can be found.Those may be a little different in names but the structure has been respected has much as possible.

4.2 Testing

The testing has been done mostly as written in the DD. To be more precise the server has been tested by unit and integrity tests, which has been all succesfully passed by th code.
Once the client has been developed the interaction via HTTP has been tested by trying all possible functonalities; this has made us all be able to test not only the correct requests and behaviours but also most of the limit cases.

Section 5

Installation Instruction

To install and run our software it is necessary to download Ultimate IntellJ Idea and docker. By using docker we have been able to cancel the whole process of downloading, installing and setting both server and database; both are simulated in dockers defined in the dockerfile.

5.1 Linux - Fedora

We have worked on Fedora 29, the following instructions, especially for what is related to Docker, are strictly related to this Linux platform.
The following steps are necessary to correctely run our software on Fedora:

- Download the **Ultimate IntellJ** version which can be found at the following link:
  `https://www.jetbrains.com/idea/download/#section=windows`
- Download **Android Studio** for simulate the two clients, the latest version can be found at the following link:
  `https://developer.android.com/studio/` To correctly install this software follow the steps at the following link:
  `https://developer.android.com/studio/intro/`
- For docker follow the prompt command explained in the next link:
  `https://developer.fedoraproject.org/tools/docker/docker-installation.html`
- To run the server code the following prompt commands must be executed:

- **selinux** and then **sudo setenforce 0**: to disactivate the security and let docker work;
- **sudo dnf install docker docker-compose**: to install the docker compose package;
- **sudo groupadd docker && sudo gpasswd -a $USER docker**
- **newgrp docker**
- **mvn clean install -DskipTests** : to correctly build the maven project;
- **sudo systemctl start docker** : because of the sudo command it will be necessary to confirm the command by inserting the password. This command is fundamental to start up the docker deamon,
- **docker-compose up - -build** : to build the docker which will contain the container for both server and DB.
  N:B The space between - - is incorrect, here it has been added jsut to show the presence of two - -.

- Before starting the client it is necessary to create the DataSource in server by folowing the next fw steps:

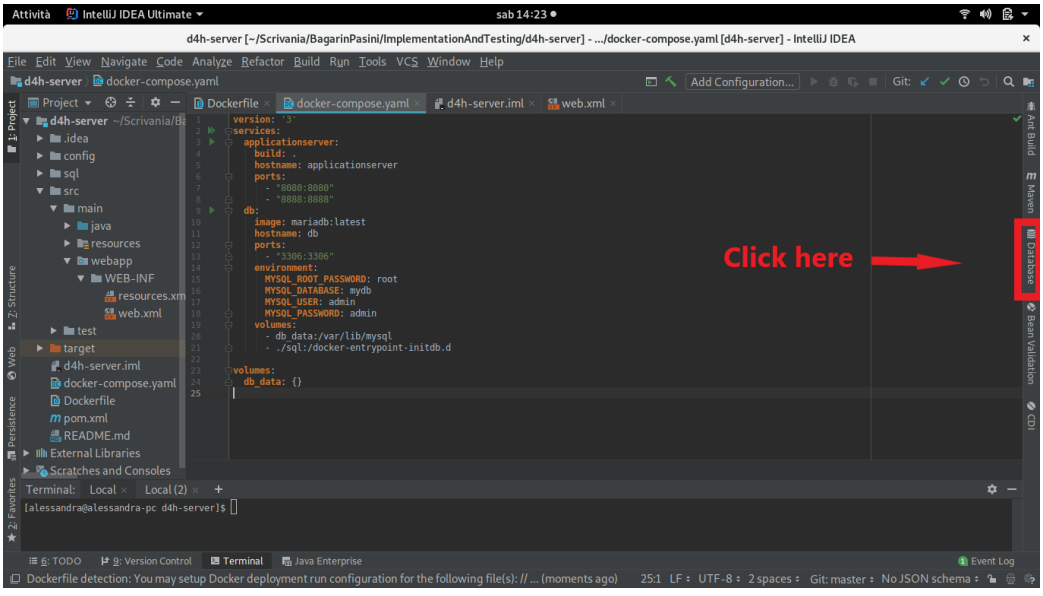

Figure 5.1: Click on the database button in the right side of IntellJ Ultimate

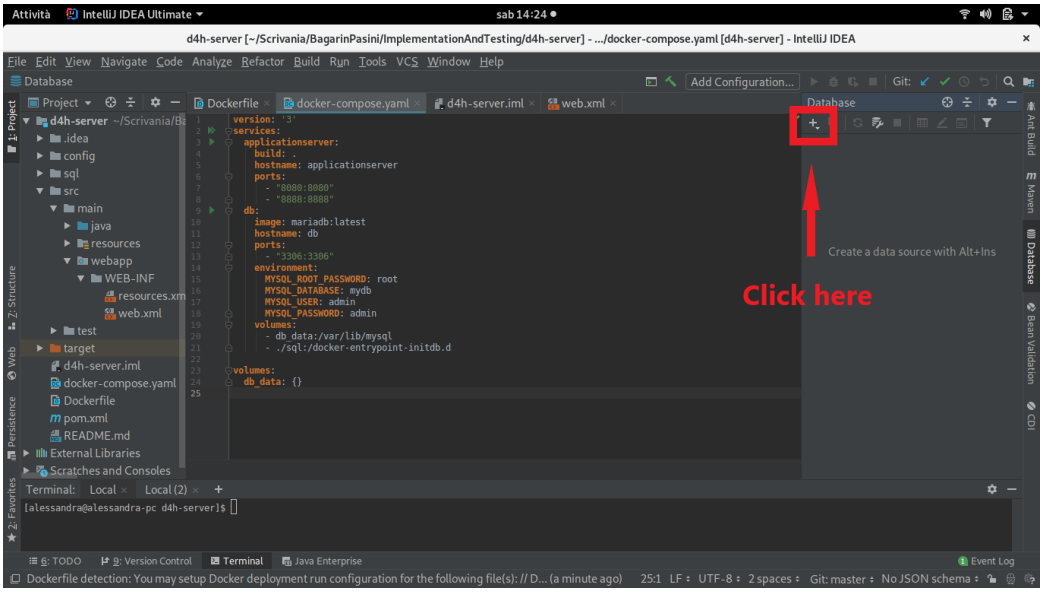

Figure 5.2: Click on the plus button to add a new DataSource. Select DataSource, click on MariaDB.

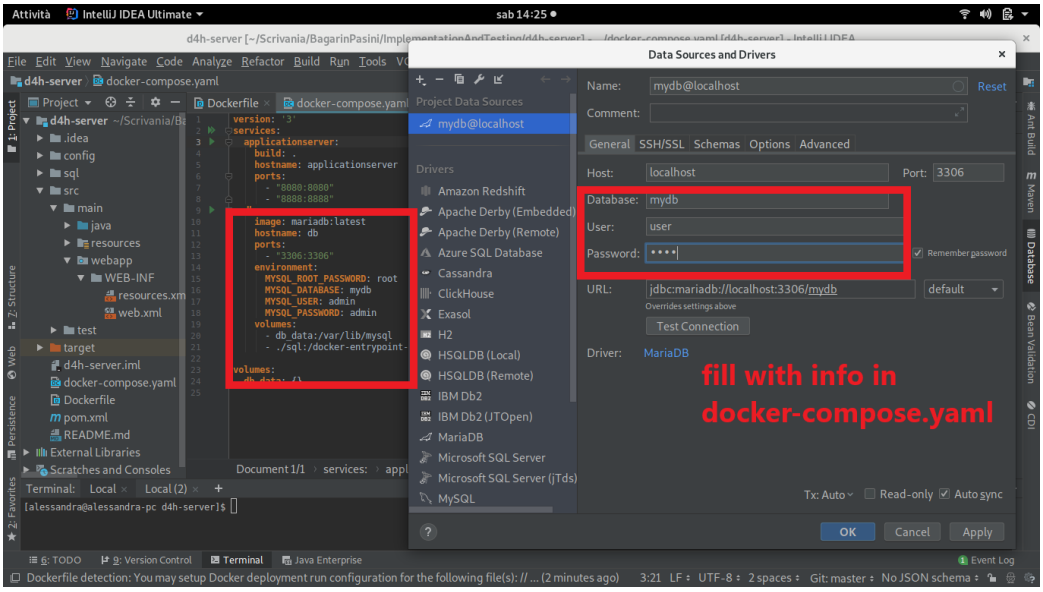

Figure 5.3: Fill the properties.

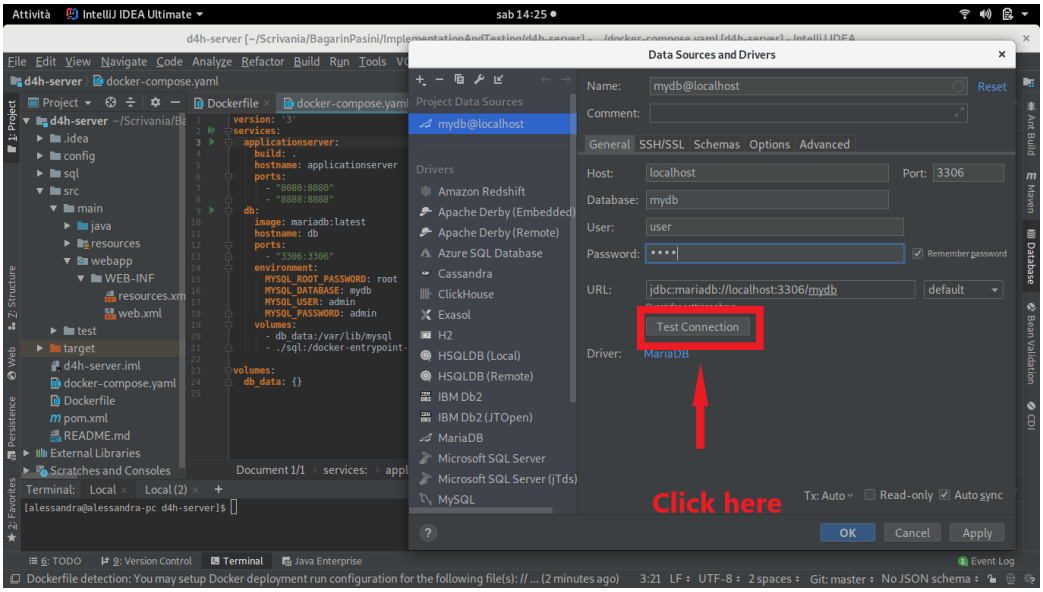

Figure 5.4: Once all properties are added verify if the connection has been established in a correct way. The test should be successful because the docker has already been built up.

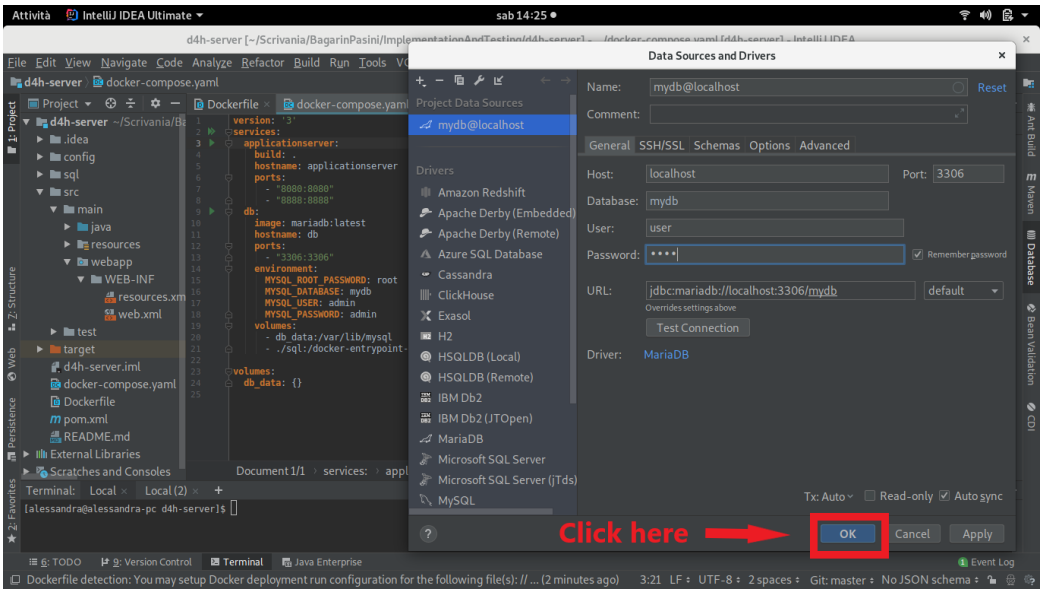

Figure 5.5: Click okey to build the DataSource.

- To run the client code it is necessary to:
  - go in *config.class* and modify the domain in URL string (of course

it will have to be the domain on which the server works);

- go in the *networksecurityconfig.xml* to modify the URL string as in config.class.
- run the app on an emulator phone.

More details can be found in the server's ReadMe which includes some more docker commands that might be useful in case of problems, such as having some dockers that already work on the prefixed doors 8080 and 3306.

5.2 Windows

The following steps are necessary to correctely run our software on Fedora:

- Download the **Ultimate IntellJ** version which can be found at the following link:
  `https://www.jetbrains.com/idea/download/#section=windows`
- Download **Android Studio** for simulate the two clients, the latest version can be found at the following link:
  `https://developer.android.com/studio/`

  To correctly install this software follow the steps at the following link:
  `https://developer.android.com/studio/intro/`
- To correctly download and install docker follow the instruction explained in the folowing link:

  - if you have Windows 10 Pro, Education, Enterprise download it at: `https://docs.docker.com/docker-for-windows/install/`
  - if you have lower versions download it at `https://docs.docker.com/toolbox/toolbox_install_windows/`

  When installing the docker remember that you will need to install also the docker-compose which is give for free in Windows and OS docker installation.
- To run the server code the following prompt commands must be executed:

  - To run the Maven project by using the Maven Docker image directly, pass a Maven command to docker run:
    $ docker run -it –rm –name my-maven-project -v "$(pwd)":/usr/src/mymaven -w /usr/src/mymaven maven:3.3-jdk-8 mvn clean install -DskipTests;
  - **start docker**this command is fundamental to start up the docker deamon;
  - **docker-compose up - -build** : to build the docker which will contain the container for both server and DB.
    N:B The space between - - is incorrect, here it has been added jsut to show the presence of two - -.

- To run the client code it is necessary to:
  - go in *config.class* and modify the domain in URL string (of course it will have to be the domain on which the server works);
  - go in the *networksecurityconfig.xml* to modify the URL string as in config.class.
  - run the app on an emulator phone.

5.2.1 Mac OS

The following steps are necessary to correctely run our software on Fedora:

- Download the **Ultimate IntellJ** version which can be found at the following link:
  `https://www.jetbrains.com/idea/download/#section=windows`
- Download **Android Studio** for simulate the two clients, the latest version can be found at the following link:
  `https://developer.android.com/studio/`

  To correctly install this software follow the steps at the following link:
  `https://developer.android.com/studio/intro/`
- To run the server code the following prompt commands must be executed:

  - To run the Maven project by using the Maven Docker image directly, pass a Maven command to docker run:
    $ docker run -it –rm –name my-maven-project -v "$(pwd)":/usr/src/mymaven -w /usr/src/mymaven maven:3.3-jdk-8 mvn clean install -DskipTests;

  - **sudo systemctl start docker** : because of the sudo command it will be necessary to confirm the command by inserting the password. This command is fundamental to start up the docker deamon,
  - **docker-compose up - -build** : to build the docker which will contain the container for both server and DB.
    N:B The space between - - is incorrect, here it has been added jsut to show the presence of two - -.

- To run the client code it is necessary to:
  - go in *config.class* and modify the domain in URL string (of course it will have to be the domain on which the server works);
  - go in the *networksecurityconfig.xml* to modify the URL string as in config.class.
  - run the app on an emulator phone.

For more details on Docker and Docker compose look at this link:
`https://docs.docker.com/`

Appendix A

Effort Spend

A.1 Efford Spent

The major part of the document has been produced working togheder and that's the reason way there is not a precise division of hours per sections and per group component.
The following is an approximate extimation of the number of hours of work for each group member:

- Alessandra Pasini: ~1 month of work * 5/7 (hours day);
- Stefano Bagarin: ~1 month of work * 5/7(hours day);

A.2 Bibliography

- AA 2018/2019 Software Engineering 2 - *Requirements Analysis and Specification Document* - Stefano Bagarin, Alessandra Pasini
- AA 2018/2019 Software Engineering 2 - *Design Document* - Stefano Bagarin,Alessandra Pasini