Politecnico di Milano
AA 2018-2019

Computer Science and Engineering

Software Engineering II

Design Document

Stefano Bagarin
Alessandra Pasini

Document version: 2.0
January 13, 2019

Contents

Section 1

Introduction

1.1 Purpose

The goal of the Design Document (DD) is to provide to the development team in charge of the project making, an overall description of the architecture it will need to have and some technical aspects. This is necessary in order to help to coordinate each team member and make them work with an unified vision of the software.
To reach the presented goal, in this document are described the architecture's design with some UML diagrams, the design of the user interfaces already described in the RASD document, the requirements traceability and the plan made for implementing, integrating and testing the software.

1.2 Scope

The aim of the TrackMe's system can be diveded in two parts: the first one is to gather data from users and to make them usable by third parties, the second is to guarantee a tempestive help to users who need it.
The software will be developed in three layers: presentation layer, server layer and data layer; the presentation layer will be strongly divided in two parts: the users part which will be developed as a mobile application and the third parties part which will be developed as a mobile application as well as a browser application. Each layer's characteristics will be described in the following chapters.
It is expected the collaboration with other companies which will provide some essential external services such as the SOS service.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions

- **TrackMe's system:** the whole software, it's a compact way to refer to all different services that must be developed.
- **Data:** location and health parameters collected by the user's device.
- **Location:** real time GPS position.
- **Health parameters:** data about the user's health status collected by his/her device, such as heart beat, vein pressure, body temperature ..
- **User:** an individual user of the application who provides data.
- **Third party:** somebody that uses the application because interested in obtaining data of users or groups of users.

1.3.2 Acronyms

- **ACID:** Atomicity, Consistency, Isolation and Durability. These are all the logic properties that transactions must have to guarantee correct operations on the stored data.
- **DBMS:** the database management system is a software system built to do operations on database in an efficient and correct way.
- **RDBM:** the relational database management system is a particular type of DB based on the relational model introduced by Edgar F. Codd.
- **HTTP:** HyperText Transfer Protocol

1.3.3 Abbreviation

- **RASD:** Requirement Analysis and Specification Document.
- **DD:** Design Document.
- **D4H:** Data4Help.
- **ASOS:** AutomatedSOS.
- **DB:** it stands for *database.*
- **App:** it stands for *application.*

- **[Gn]:** n-th goal
- **[Rn]:** n-th requiremnt

1.4 References

This document is strictly based on the specification concerning the DD 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 also refers to the RASD previously produced, more details about it can be found in the bibliography or in the GitHub repository.

1.5 Document Structure

This document consist in five sections:

- **Section 1- Introduction:** In this chapter is given an overall view of the document, its goal and the informations to understand it's contents.
- **Section 2 - Architectural Design:** The aim of this section is to give all necessary information about the system's structure: how components are done and their interactions, the runtime flow of events, the used hardaware and all necessary details to obtain what has been previously expressed in the RASD.
- **Section 3 - User Interface Design:** This section contains two flowchart diagrams in order to highlight the passage from a scene to an other one and gives further information with respect to those contained in the RASD.
- **Section 4 - Requirements Traceability:** It provides a mapping between goals and requirements expressed in the RASD and the designed elements defined in this document.
- **Section 5 - Implementation, Integration and Test Plan:** The aim of this section is to provide a work plan for the development team.
- **Appendix A:** contains the software and tools used, the bibliography and all efforts spent by each group component.

Section 2

Architectural Design

2.1 Overview

This section gives a detailed view of the physical and logical infrastructure.

High level component

The app has been designed by applying the three layers architecture; this pattern, also kown as the n-tier architecture, is the de facto standard for most Java EE applications and devide in a well defined way the application's parts by assigning a specific role to each layer.

- **Presentation Layer:** It is the front end layer, it consists in the external interfaces and it is composed by three sublayers: two mobile application dedicated to mobile devices, one for users and one for third parties, and a web browser for third parties.
  Both mobile apps includes only the presentation login and all sublayers directely interface with the app layer.

- **Application Layer:** It encases the app's core capability, this means that containts the whole logic included the main algorithms.

- **Data Layer:** It includes the data storage system and data access. Data are accessed by the application layer via API call.

In the following Figure the external services have been included to underline the fact that they interface with both presentation and application tier and it also underlines the fact that layers are closed. A closed layer means that a request moves from layer to layer and it must pass through all layers right below before reaching the destination one.

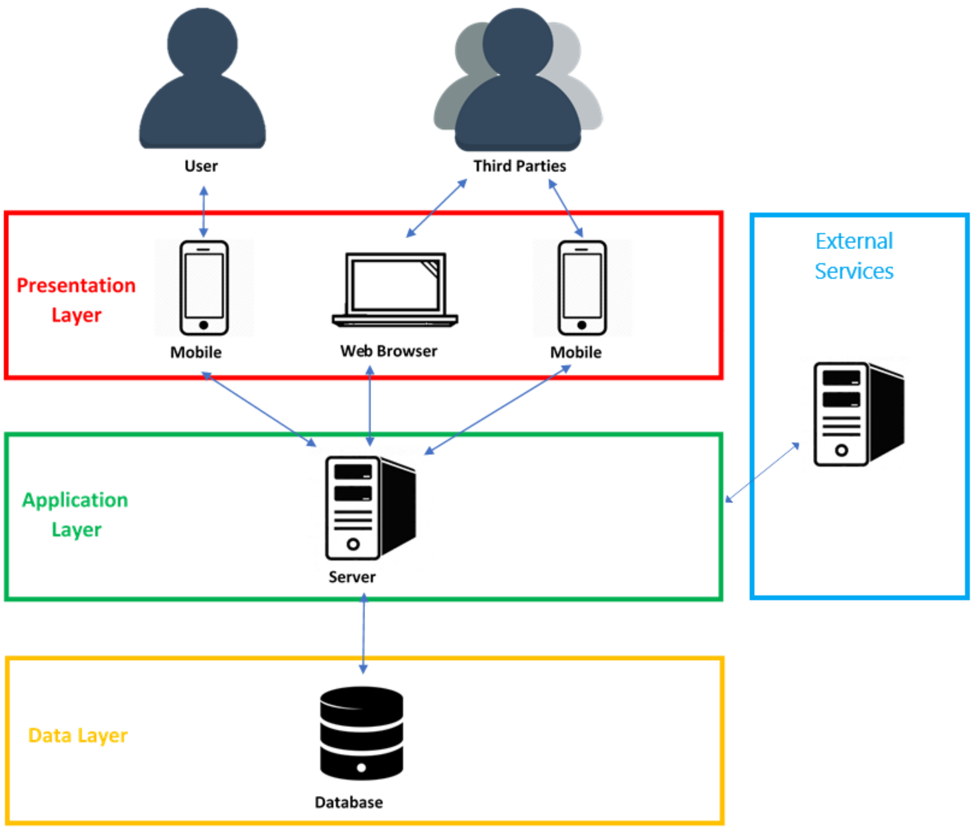


Figure 2.1: Three-tiers architecture

The next diagram shows interactions between all main components of the system. Because of the presence of two different stakeholders with different needs the client has been splitted in two according to the division in 2 different apps explained in the RASD. Althought this client's division the server has been maintained unique to avoid possible code redundancy and because most of the logic is related to the Third Party's app.

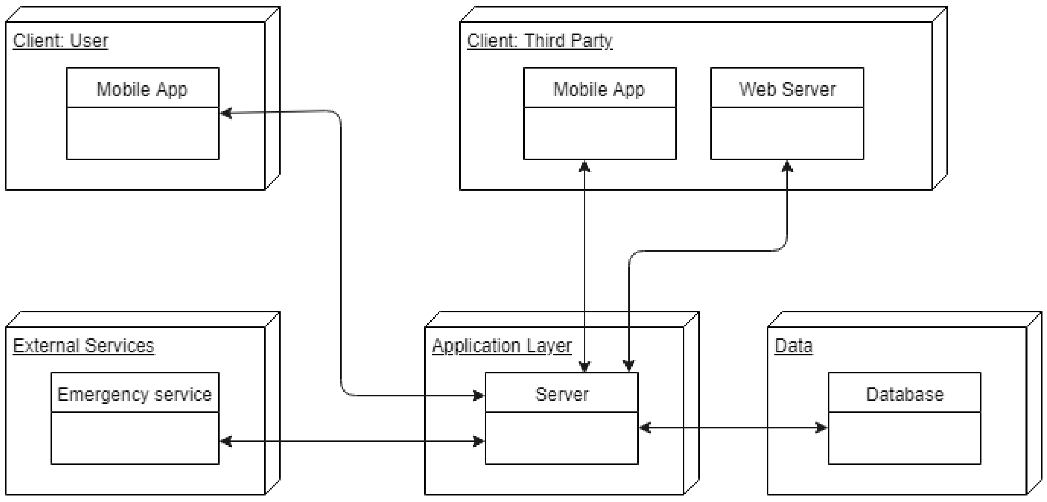


Figure 2.2: High Level Components

2.2 Component View

2.2.1 Components' Structure

The presented Component Diagram has the aim to show in a more precise and detailed way which components are in each layer and how they interact. To be more precise in the left it is possible to see the presentation layer devided in three components:

- **User's Mobile App:** it is the user's application via device; that is the only possible interface for a user and it is formed by a *Data Collector* and a *Data Processor* components which have the aim to collect data from the external device containing all the necessary sensors to gather health parameters of every user ; a *GPS Manager* which interfaces directly with the device's GPS needed in case of health parameter under the threshold (see RASD for more details); a *GUI Manager* which is foundational to handle all actions made by users and a *Controller* which is the bridge between the client's components and the application layer.

- **Third party's Mobile App:** it is the third party's application via device. This particular stakeholder doesn't need any GPS or Data manager because the aim of its app is to obtain data of a single user or of a group of users, that's the reason why it is just formed by a *GUI Manager* and a *Controller*. The aim of those two elements is the same presented for the user's mobile app.

- **Third Party's Web App:** the different aim of the app related to this stakeholder implies the necessity of a desktop application which is dial by the same components of the Mobile one.

The application layer is also formed by three different components:

- **Application Logic:** it contains the whole logic and algorithms which stay under the application and make it runs in a proper way. This component will be better analyzed in a dedicayìted subsection in which all its services will be presented into a class diagram.

- **ASOS Manager:** this component is in charge of interfacing with the external emergency service in case the related algorithms finds out that the user is an unhealthy one.

- **Data Persistence Unit:** it has a constant connection to the data layer and it has the aim to handle all possible dynamic behaviors related to operations on data.

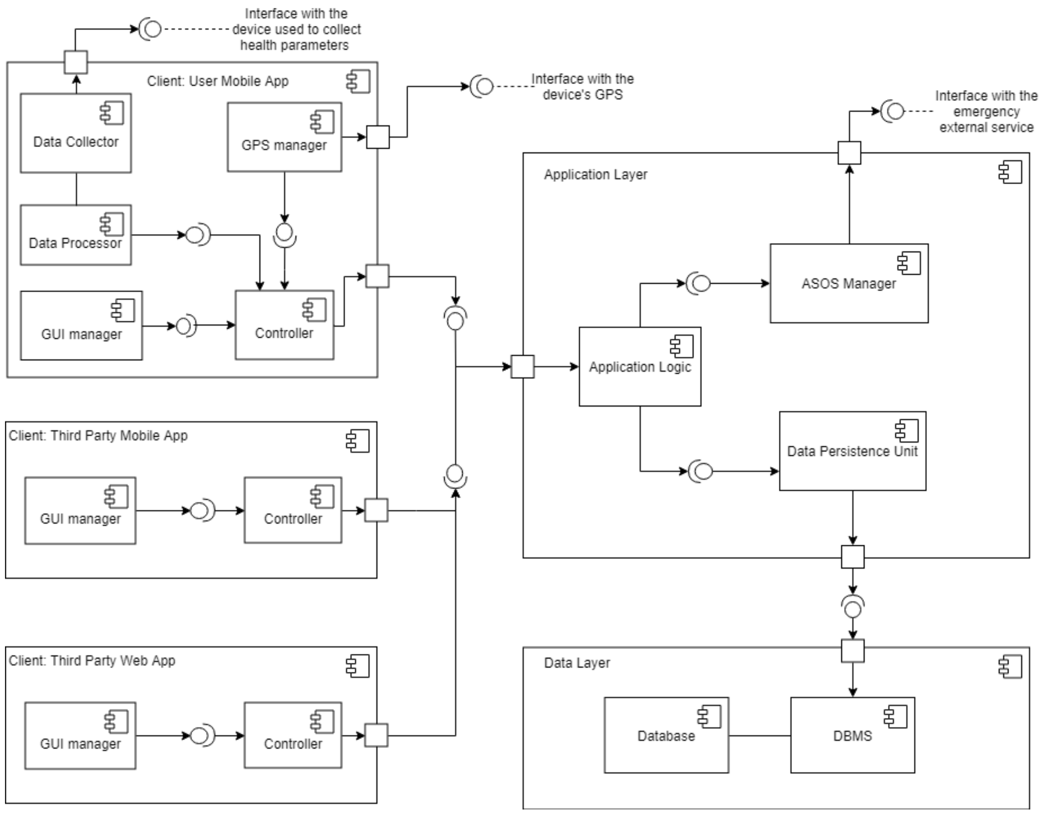


Figure 2.3: Component diagram

2.2.2 Application Logic

To better describe the model's and controller's structure, is presented an UML class diagram more precise than the one in the RASD document.

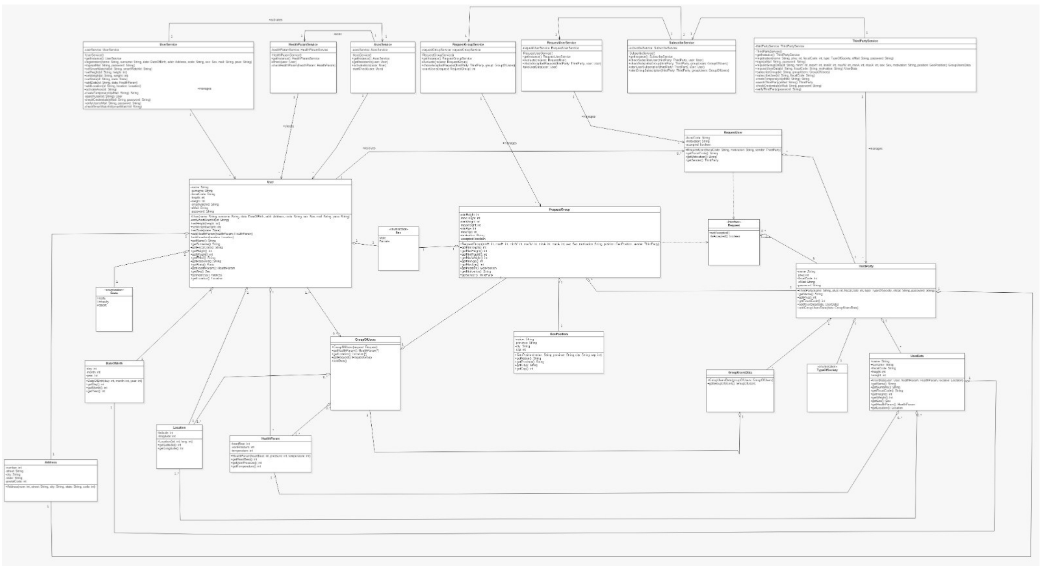


Figure 2.4: Class diagram showing the model of the appliation, it also contains all manager classes which have the aim to handle all possible dynamic actions.

Note that the presented class diagram is intended as a guide for the implementation and integration of the model's part of the software.

2.2.3 Database

The data layer must contain the DB and also a DBMS component necessary to manage all possible operation, such as insertion, deletion, update, that can be done on data inside the storage unit. The chosen DBMS must guarantee the correct working of transaction according to the ACID properties.
A RDBMS is the best choice for this app because it adheres to the ACID properties; guarantees high performances by using indexis to sort data and by supporting both desktop and web apps and also guarantees data security providing a layer of protection for sensitive data. Using a RDBMS it is also possible to build the DB structure and clearly define the relations between data.
As said in the previous section, the chosen architecture is composed by closed

layers and for this reason the data layer, the one which contains the stored data, can only be accessed by the Application's one, via a dedicated interface. Bacause of this it is necessary to provide a persistence unit which can handle all possible behaviors. Credentials and sensible data as personal informations must be encripted to avoid any possible privacy violation.
The Entity Relational diagram in figure describe a first relational model of the Database. The rectangual shapes represent the entities; the circles all possible attributes related to an entity in particular all cirles filled in black are the entities key and the relationships are represented by diamonds.

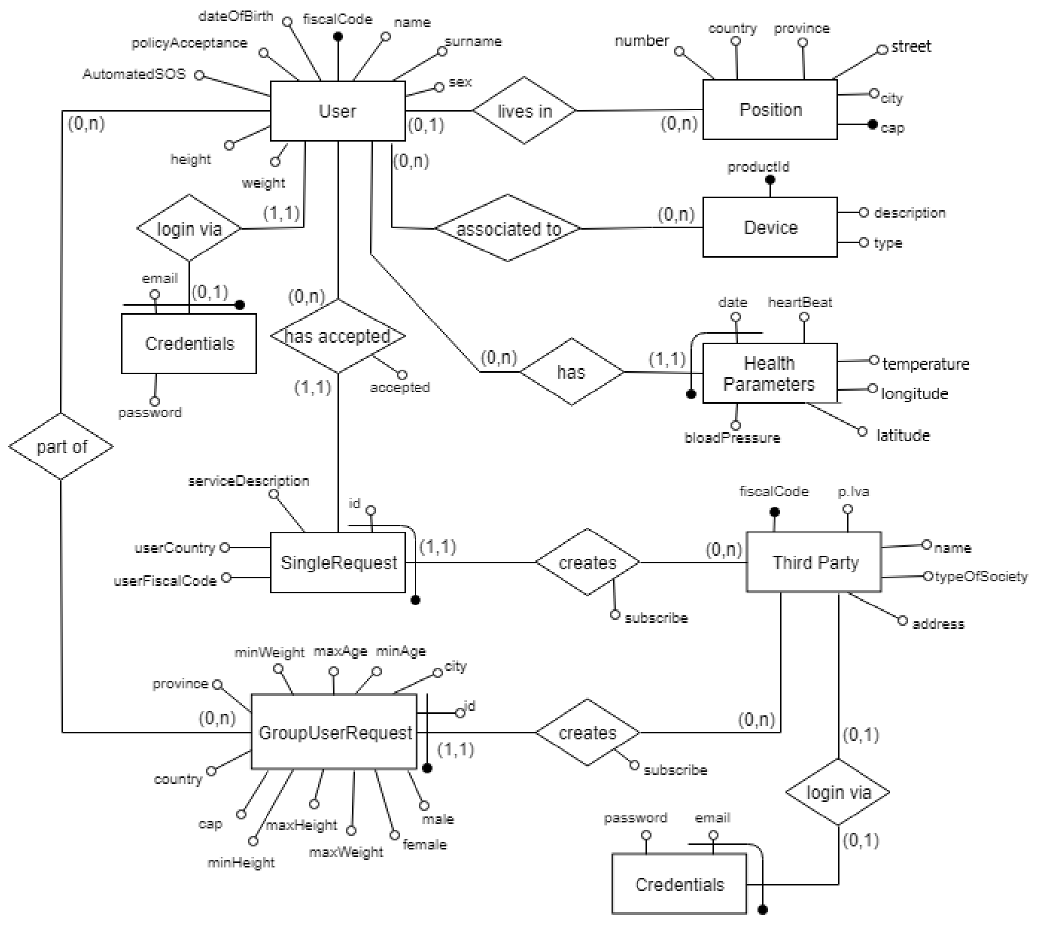


Figure 2.5: Entity Relationship diagram

2.3 Deployment view

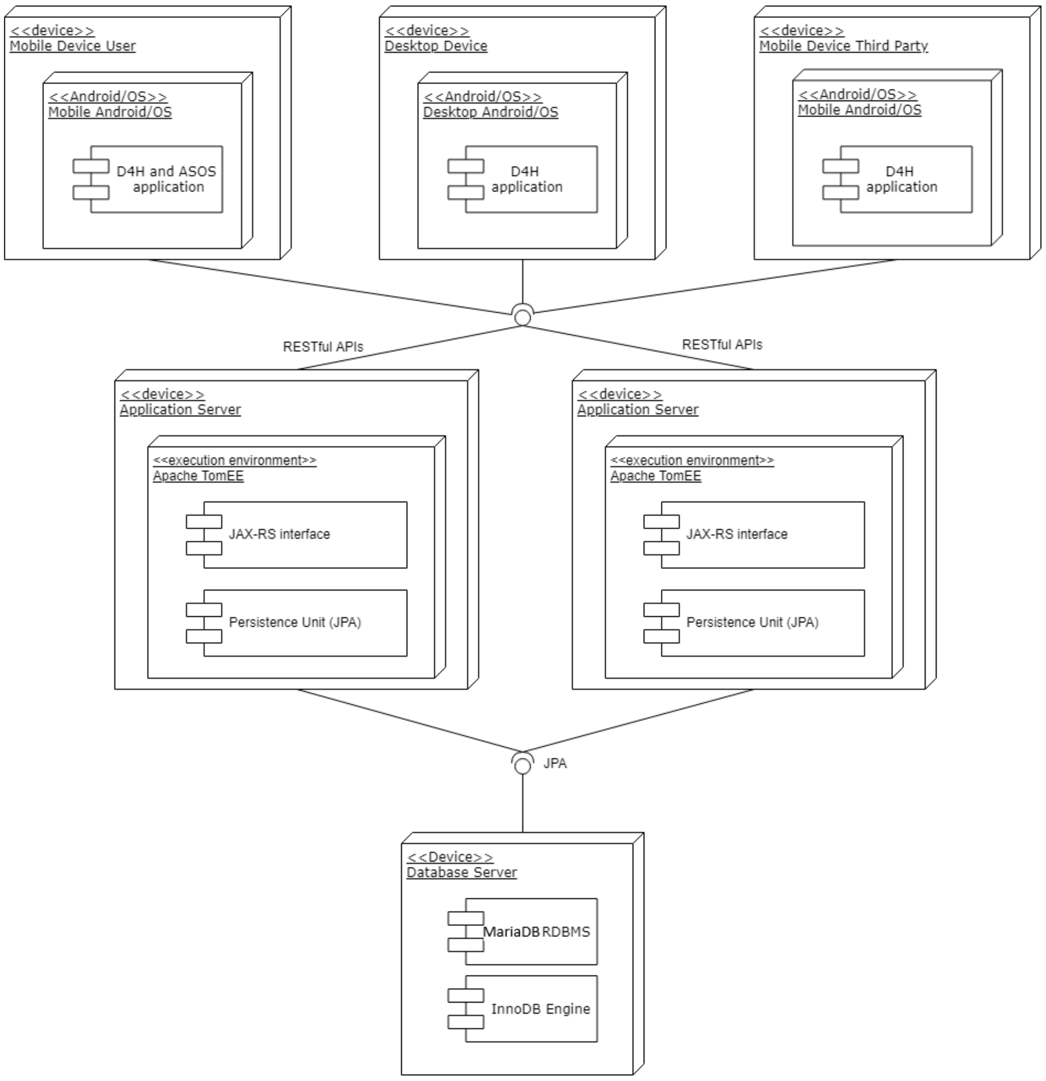


Figure 2.6: Deployment diagram

The Mobile App of users and third parties must be available on both IOS and Android systems; in order to obtain it the presentation layer must be code in Java for Android and in Swift for IOS.
More than one application servers is represented in the deployment diagram in order to guarantee high availability and reliability, as said in RASD's section 3.6, and to reduce the payload on each machine by distributing it.
It has been choice to use Java Enterprise Edition 8.2 (JEE) for the app layer because the aim of the final product is to become a large scope application with a great number of simultaneous clients and because, thanks to the ammount of APIs and tools the developers can focus on the main logic. Here

there are some choices made for the App Server implementation:

- TomEE has been chosen as server because it is an incredibly lightweight app, in fact it offers only the most basic functionality necessary to run a serverand it allows to use other API to solve all non basic ones. Thanks to its lightweight it is quite flexible and stable. TomEE has been choice also because it has born as a Web Server and than it became something close to an application server; for this reason it is totally compatible also with the 3 layer architecture that has been proposed in this document.

- The RESTful APIs architecture is applied because it makes the efficiet use of the bandwidth and uses standard HTTP. It also has a great scalability and it allows to carry out basic operations;

- JAX-RS to implement proper REStful APIs to interface with clients;

- Java Persistence API (JPA) as persistence Unit to acces the database. It has been picked out because it is an Object/ Relational Mapping (ORM) framework applicable to RDBMS without implement direct SQL queries and because of its high performances , scalability, stability and quality.

The DB's implementation choices are the following:

- JPA within the App Server as the interface to the DB;

- MariaDB as the RDBMS;

- InnoDB as the data engine; this is already the default engine applied to every table generated in MariaDB but it has been choice because it associates high performances to high data consictency and integrity. To let this happens, this particular engine has developed some particular functionalities such as the introduction of foreign keys; locks at row level; Multiversion Concurrency Control (MVCC) to increment the concurrency in queries and support in tansactions to guarantee correctness in data.

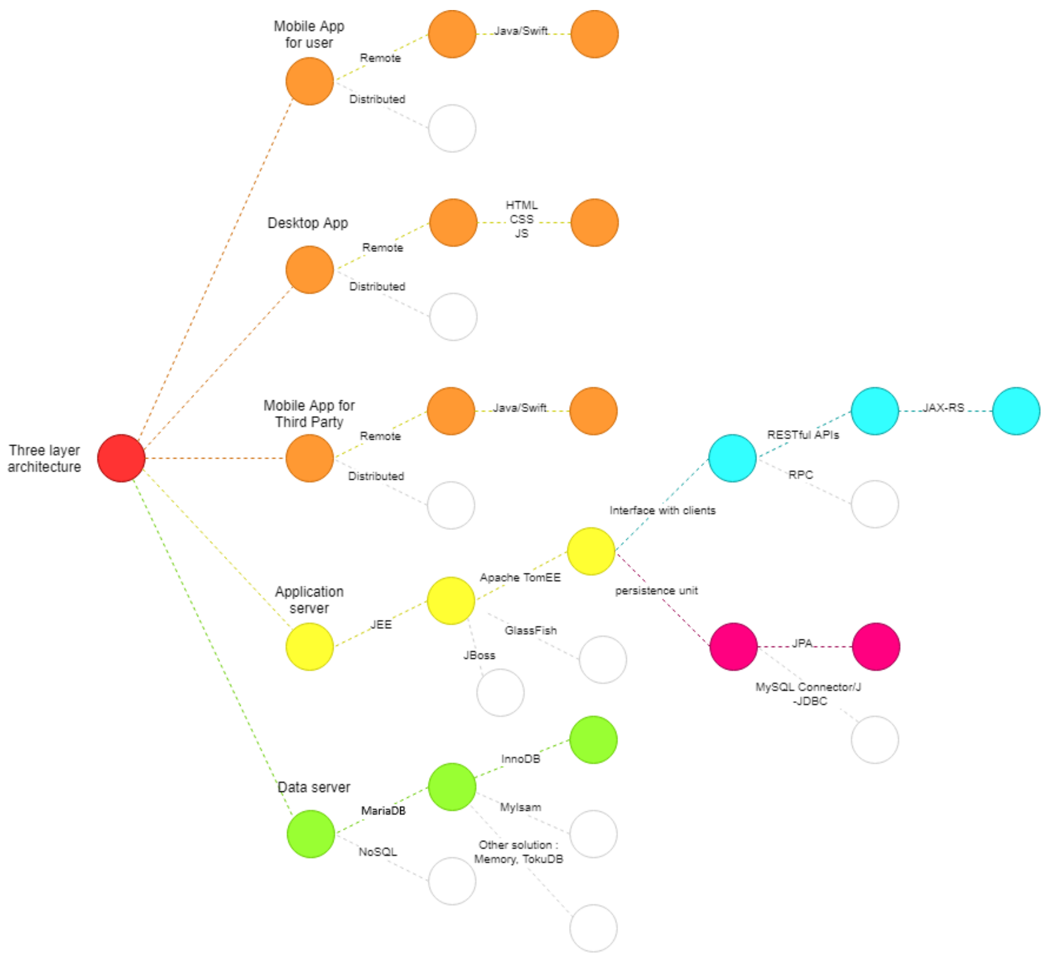
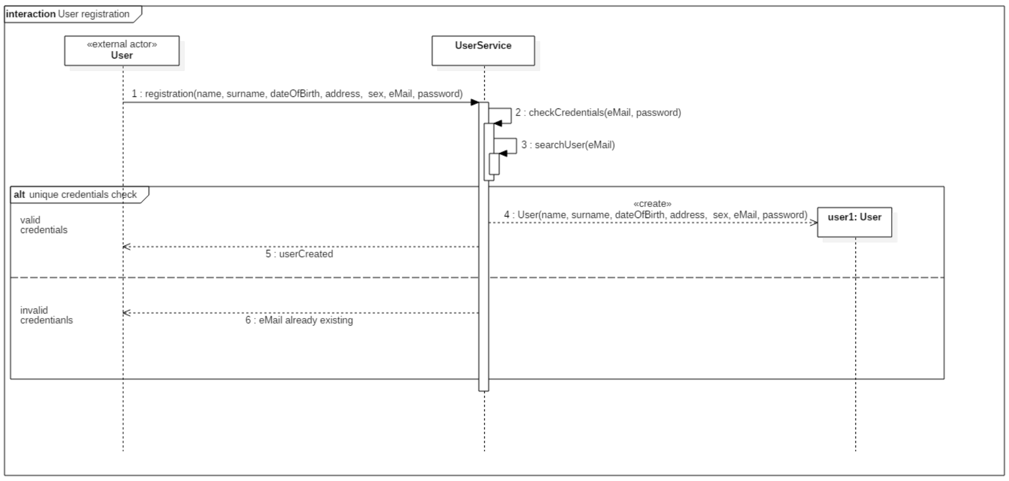


Figure 2.7: The aim of this figure is to show which implementation possibilities have been discarded.It represents the decision tree for the starting architectural design.

2.4 Runtime view

The runtime view of the software is described by a set of UML sequence diagrams that shows how the software will behave.
Note that the processes of registration and login are analyzed only for users but they are very similar for third parties.



interaction User registration
«external actor» User
UserService
1 : registration(name, surname, dateOfBirth, address, sex, eMail, password)
2 : checkCredentials(eMail, password)
3 : searchUser(eMail)
alt unique credentials check
valid credentials
«create»
4 : User(name, surname, dateOfBirth, address, sex, eMail, password)
user1: User
5 : userCreated
invalid credentianls
6 : eMail already existing

Figure 2.8: Sequence diagram about the registration of a user.

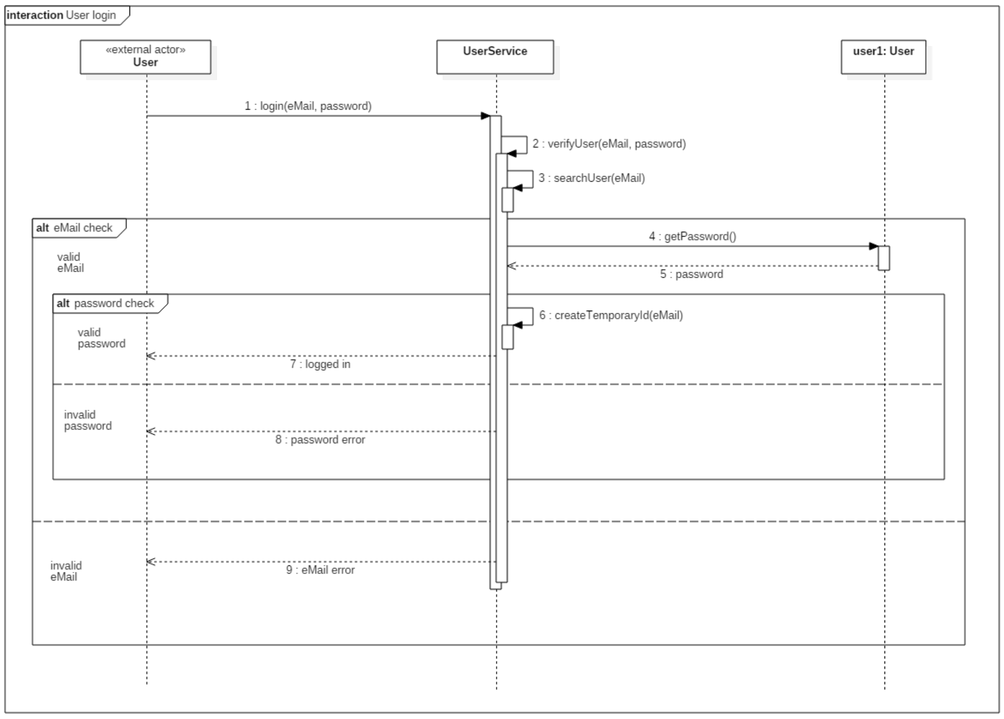


Figure 2.9: Sequence diagram about the login of a user.

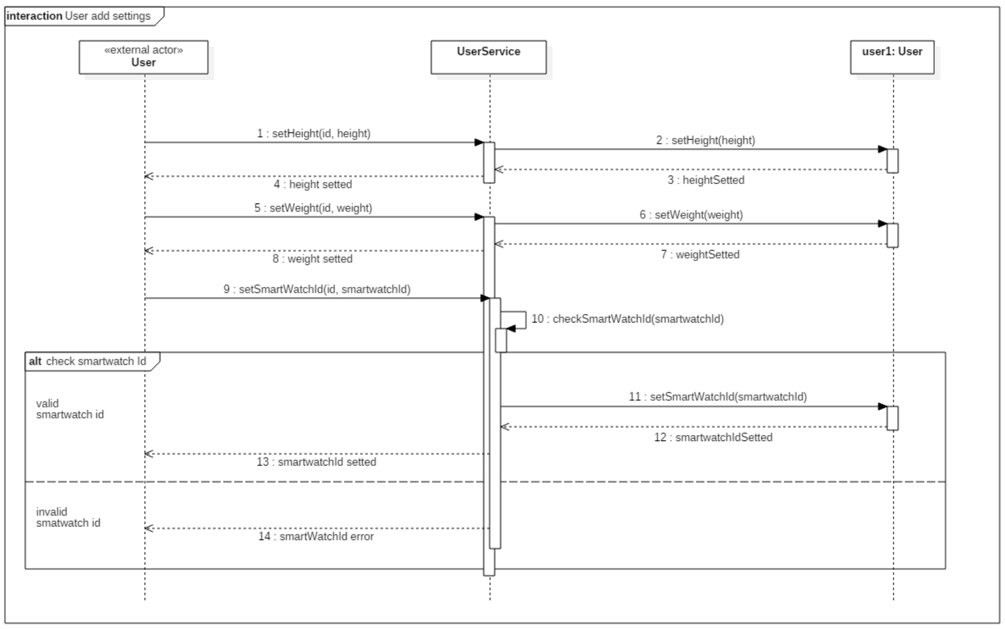


Figure 2.10: Sequence diagram about adding a user's settings.

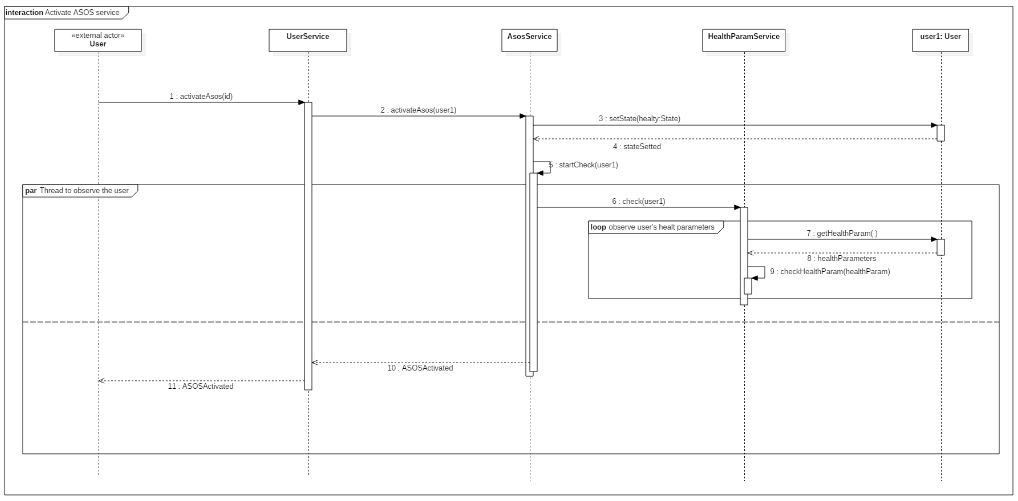


Figure 2.11: Sequence diagram about the activation of ASOS.

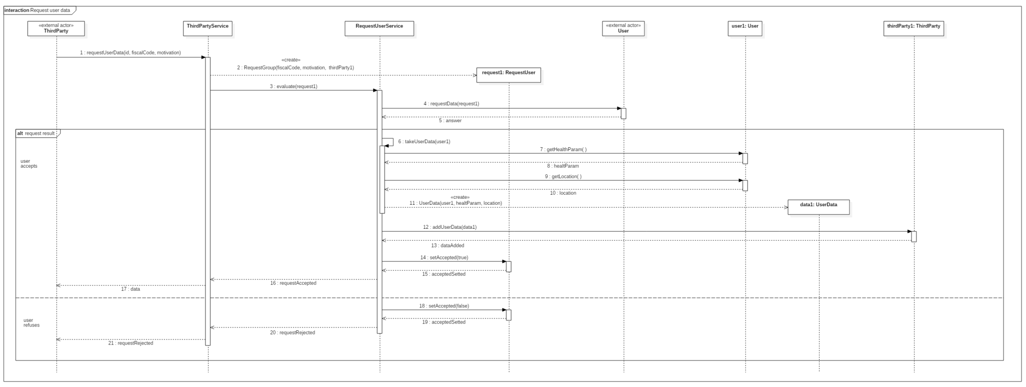


Figure 2.12: Sequence diagram about requesting data about a user.

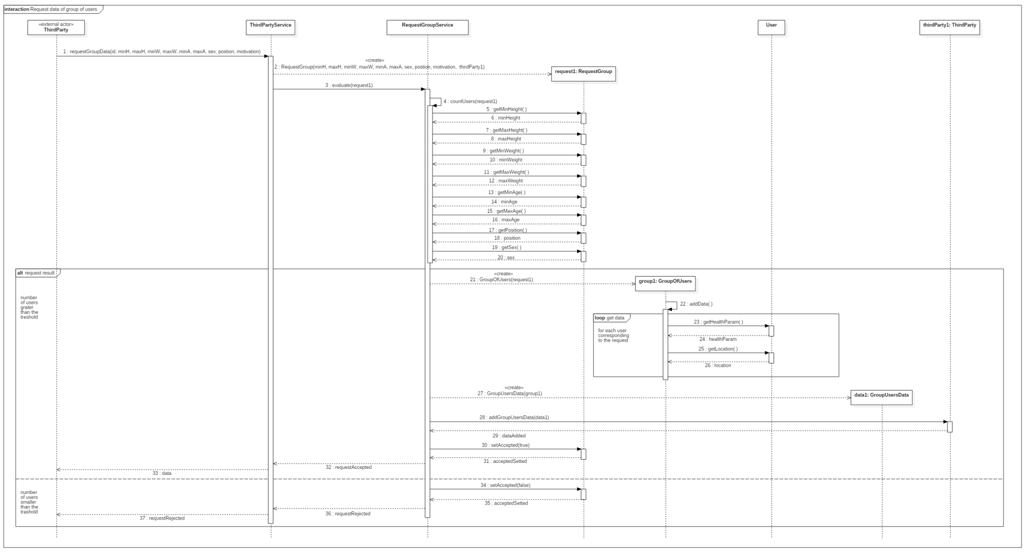


Figure 2.13: Sequence diagram about requesting data about a group of users.

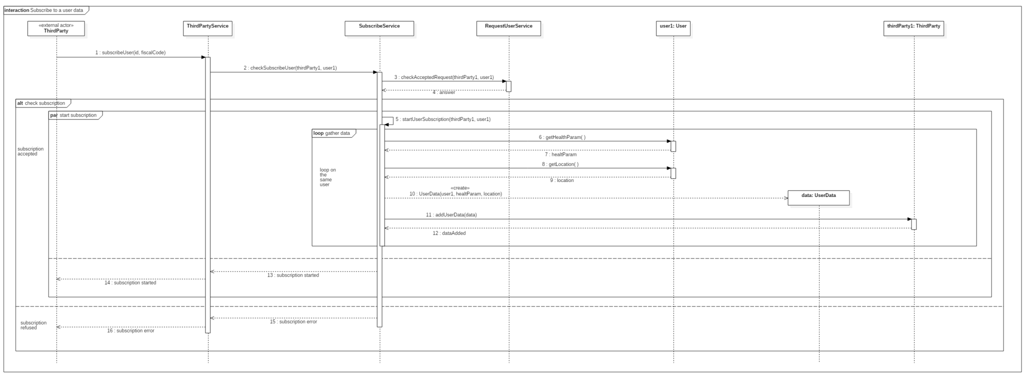


Figure 2.14: Sequence diagram about subscribing to a user data.

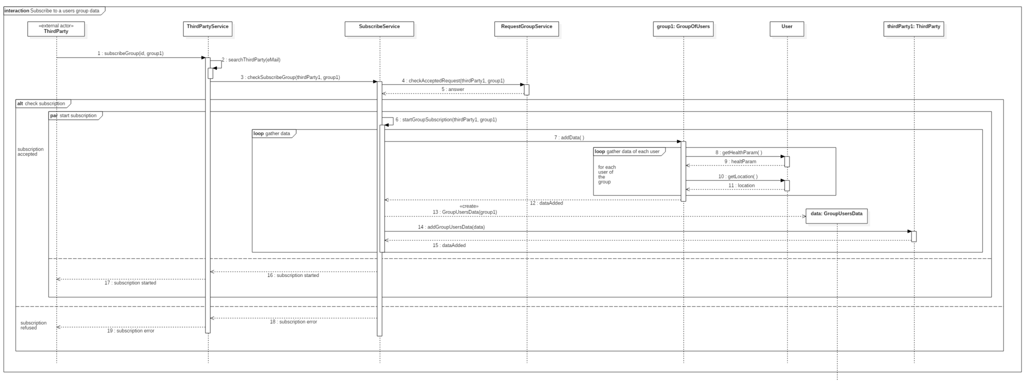

interaction Subscribe to a users group data
«external actor» ThirdParty
ThirdPartyService
SubscribeService
RequestGroupService
group1: GroupOfUsers
User
thirdParty1: ThirdParty
1 : subscribeGroup(id, group1)
2 : searchThirdParty(eMail)
3 : checkSubscribeGroup(thirdParty1, group1)
4 : checkAcceptedRequest(thirdParty1, group1)
5 : answer
alt check subscription
par start subscription
subscription accepted
6 : startGroupSubscription(thirdParty1, group1)
loop gather data
7 : addData( )
loop gather data of each user
for each user of the group
8 : getHealthParam( )
9 : healtParam
10 : getLocation( )
11 : location
12 : dataAdded
«create»
13 : GroupUsersData(group1)
data: GroupUsersData
14 : addGroupUsersData(data)
15 : dataAdded
16 : subscription started
17 : subscription started
subscription refused
18 : subscription error
19 : subscription error

Figure 2.15: Sequence diagram about subscribing to a users group data.

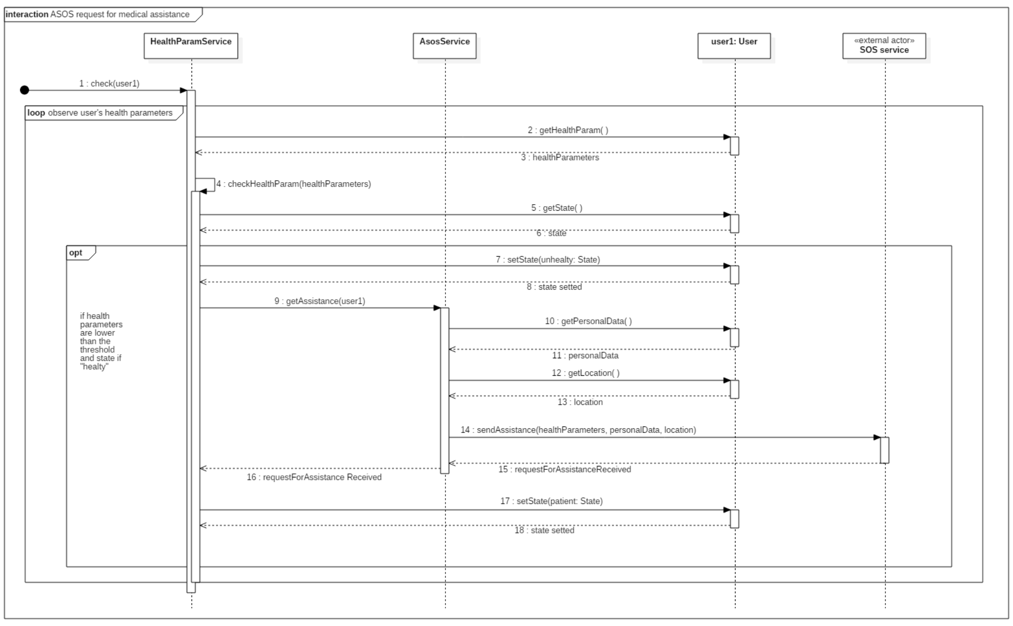

interaction ASOS request for medical assistance
HealthParamService
AsosService
user1: User
«external actor» SOS service
1 : check(user1)
loop observe user's health parameters
2 : getHealthParam( )
3 : healthParameters
4 : checkHealthParam(healthParameters)
5 : getState( )
6 : state
opt
7 : setState(unhealty: State)
8 : state setted
9 : getAssistance(user1)
if health parameters are lower than the threshold and state if "healty"
10 : getPersonalData( )
11 : personalData
12 : getLocation( )
13 : location
14 : sendAssistance(healthParameters, personalData, location)
15 : requestForAssistanceReceived
16 : requestForAssistance Received
17 : setState(patient: State)
18 : state setted

Figure 2.16: Sequence diagram about ASOS requesting for assistance for a user. In the "Activate ASOS service" diagram the way the sequence of actions presented in this diagram starts can be better seen.

2.5 Component Interfaces

This section is devoted to give more datails on the interface between all different components presented in section 2.2.

Client - Application Server

The communication between the client and the Application server is done via Restful APIs provided by the application server and implemented using JAX-RS.

Application Server - External Service

The external service is an Emergency SOS service, as already said in the RASD, it could be the ECall and E112 system provided in Europe and with already implemented interfaces in most of the countries outside the EU. It is supposed that this system provides the interface APIs to which the Application Server can adapt itself, via the ASOS manager component, to send the health parametes and the location of an unhealty user.

Application Server - Database

Because of the chosen architecture and because of the closed layers the Data Layer can only be access by the App Layer. The Application Server is the only one that can access to the database and this is done through the Persistence Unit which implement the Java Persistence API (JPA) which create a sort of mapping between java objects and the elements inside the DB.

2.6 Selected architectural styles and patterns

The following styles and patterns have been used:

Three layer architecture

As already said in section 2.1 it has been chosen a three layer architecture formed by the Presentation layer, the Application layer and the Data layer. The use of this architecture allows to phisically divide presentation, application processing and data operations.
The presentation layer is extensive and includes both users and third parties interactions in various device (multichannel access) and tools for managing the tier. It is implemented based on the design pattern MVC (Model-View-Controller) but the direct link to the data layer is connected to the app tier, which is made accessible using services components, consisting of the service model of the application.

Thin Client for Third Parties

It has been chosen to use the thin client approach while designing the interaction between third parties' devices and the system. Because of this the main logic is implemented by the App Server, which has been designed to have sufficient computing power and to work in an efficient way. It has also been picked because, by this way, the app via device doesn't take to much space, it can work even in case of limited computing power and it can be updated in an easier way.

Thick Client for Users

The thick client approach has been picked while designing the interaction between users' devices and the system. Even though the main logic is still implemented by the App Server, this type of client has been chosen to deal with the associated devices' sensors devoted to collect the user's health parameters. However, the added logic shouldn't require to much space, so also the users' application should still be lightweight and easily updatable.

Model-View-Controller

The system is design using the Model-View-Controller (MVC) pattern which separates internal representations of information from the ways information is presented to and accepted from the user.

- The model directly manages data, logic and rules of the application while beeing independent of the user interface.
- The controller receives all the users' inputs and performs their requests interacting with the model.
- The view gives a representation of the model to the users.

The choice of using the MVC pattern is made because goes well with the three layer architecture and garants efficient code reuse and parallel development.

Singleton

The model's "service" classes are designed as singletons. Those includes UserService and ThirdPartyService which are the classes in charge of manage the interactions with clients; RequestUserService and RequestGroupService which manages data requestes made by third parties; SubscribeService which gather data from users and group of users and gives them to third parties; HealthParamService and AsosService which are in charge of monitor health parameters of users and send assistance to them if their health parameters goes below a certain threshold.
The choice of design these classes as singletons is made because is needed only one object of them to coordinate actions across the system.

2.7 Other design decisions

Password Security

In order to guarantee the highest possible secutiry of credentials, they won't just be encripted but they will be also salted; this choice has been made because people usually reuse the same password and becasue it is the access key to the personal area and personal data which must be protect from strangers.

Section 3

User Interface Design

In this section there aren't mockups showing the design of the mobile apps because they have already been inserted in the RASD but there are mockups representing the third parties' desktop application.
After a reasonable analysis of the real world in which the final product will be insert, the third parties' app has been designed as a multichannel application . Third parties are companies, no profit societies, govarnement offices so it is reasonable to think that the main way to access to the app is via one or more desktops inside them; it has been chosen to add also the mobile version to obtain a larger possible base of third parties.
There are also two UI flowchart diagram representing the different screens provided by the apps and the dynamic contents. The purpose of these diagrams is to show the interaction among different scenes, which actions that let to moove from one scene to the other and which actions generate errors. Even though there are three GUI, only two diagrams can be found in this section because the third parties's app is a multichannel one and both desktop and mobile app implements the same functinalities.
All images in the diagrams have been taken by the RASD,section 3.

3.1 UI diagrams

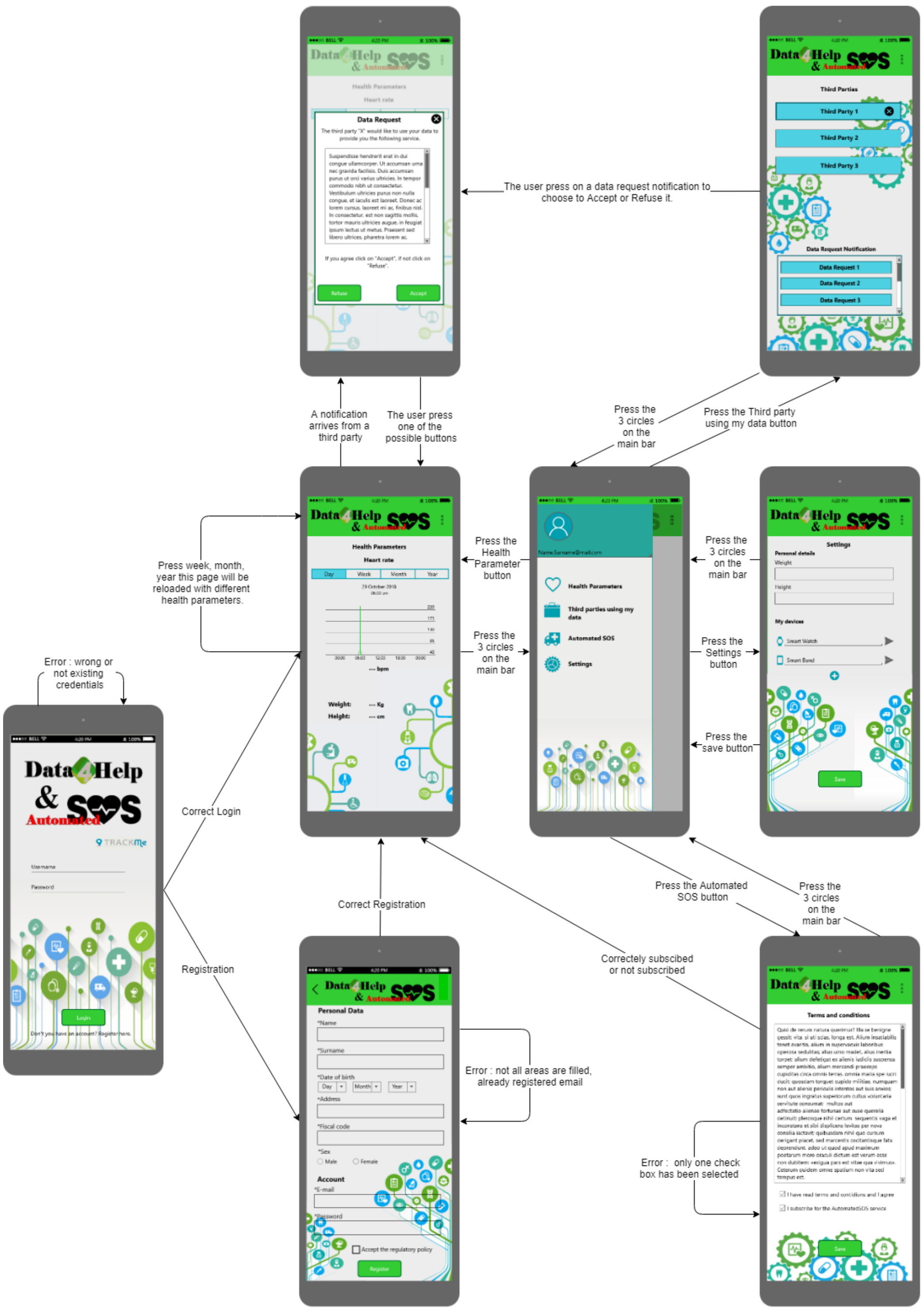

The user press on a data request notification to choose to Accept or Refuse it.
A notification arrives from a third party
The user press one of the possible buttons
Press the 3 circles on the main bar
Press the Third party using my data button
Press week, month, year this page will be reloaded with different health parameters.
Press the Health Parameter button
Press the 3 circles on the main bar
Press the 3 circles on the main bar
Press the Settings button
Press the save button
Error : wrong or not existing credentials
Correct Login
Registration
Correct Registration
Press the Automated SOS button
Press the 3 circles on the main bar
Correctely subscibed or not subscribed
Error : not all areas are filled, already registered email
Error : only one check box has been selected
Data Request
Refuse
Accept
Third Parties
Third Party 1
Third Party 2
Third Party 3
Data Request Notification
Data Request 1
Data Request 2
Data Request 3
Health Parameters
Heart rate
Day
Week
Month
Year
Weight:
Height:
Health Parameters
Third parties using my data
Automated SOS
Settings
Personal details
Weight
Height
My devices
Smart Watch
Smart Band
Save
Username
Password
Login
Personal Data
*Name
*Surname
*Date of birth
*Address
*Fiscal code
*Sex
Male
Female
Account
*E-mail
*Password
Accept the regulatory policy
Register
Terms and conditions
I have read terms and conditions and I agree
I subscribe for the AutomatedSOS service
Save

Figure 3.1: UI flowchart diagram of the users' mobile app.

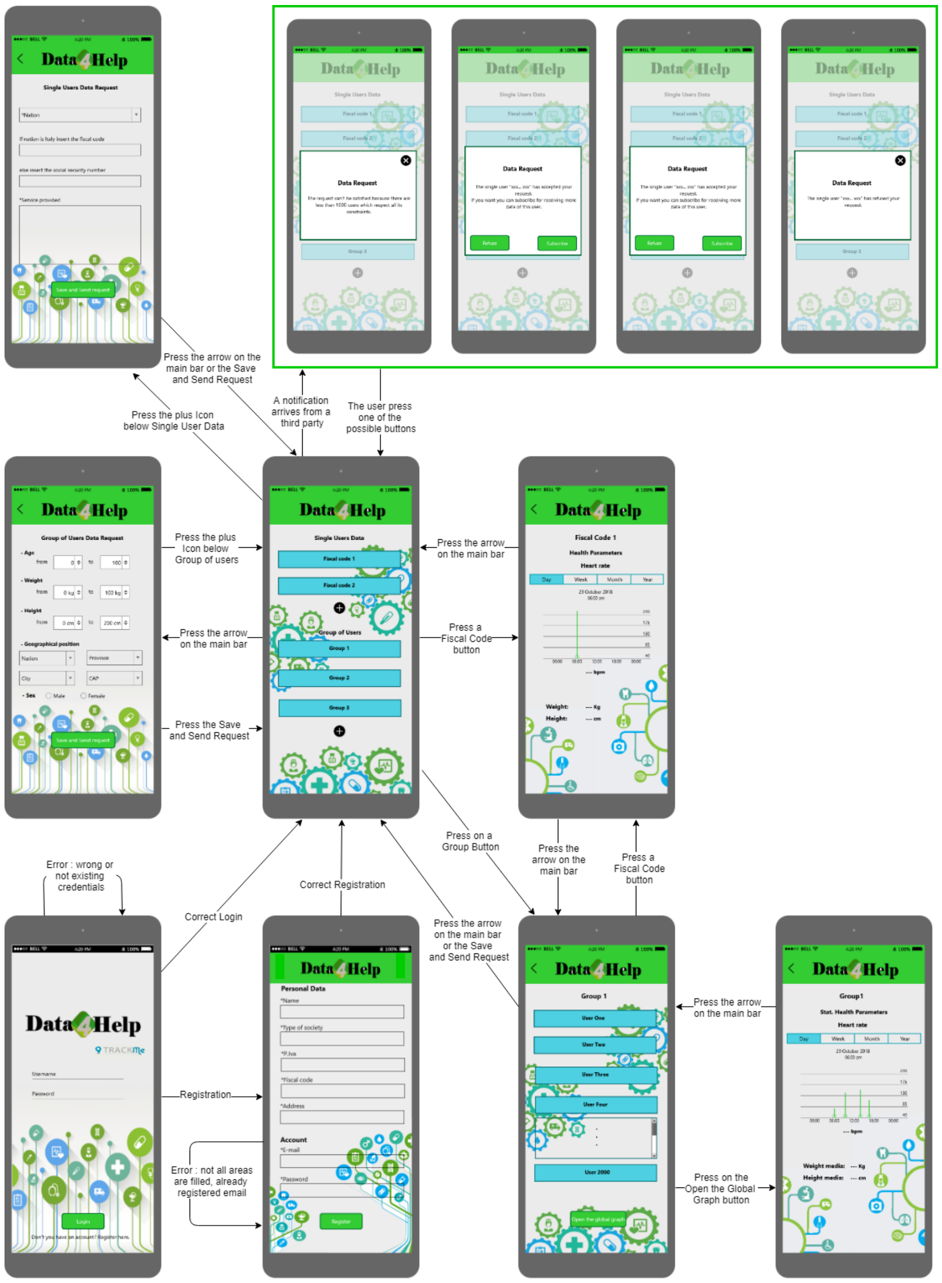

Press the arrow on the main bar or the Save and Send Request
Press the plus Icon below Single User Data
A notification arrives from a third party
The user press one of the possible buttons
Press the plus Icon below Group of users
Press the arrow on the main bar
Press the Save and Send Request
Press the arrow on the main bar
Press a Fiscal Code button
Press on a Group Button
Press the arrow on the main bar
Press a Fiscal Code button
Error : wrong or not existing credentials
Correct Login
Correct Registration
Press the arrow on the main bar or the Save and Send Request
Registration
Error : not all areas are filled, already registered email
Press the arrow on the main bar
Press on the Open the Global Graph button

Figure 3.2: UI flowchart diagram of the third parties' mobile app. It is equal to the desktop UI flowchart diagram because it does the same operation possible on the mobile app.

3.2 Desktop Interface

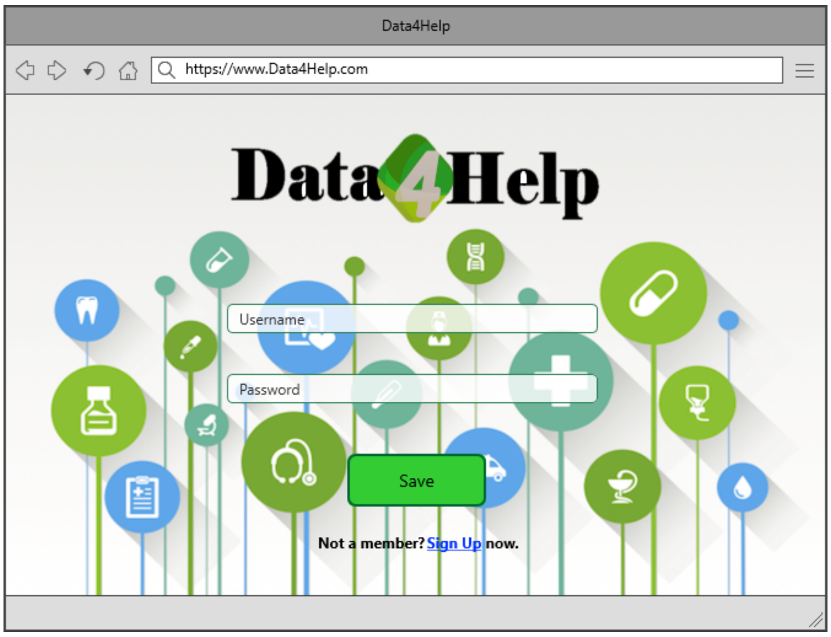


Figure 3.3: Mock up - Login

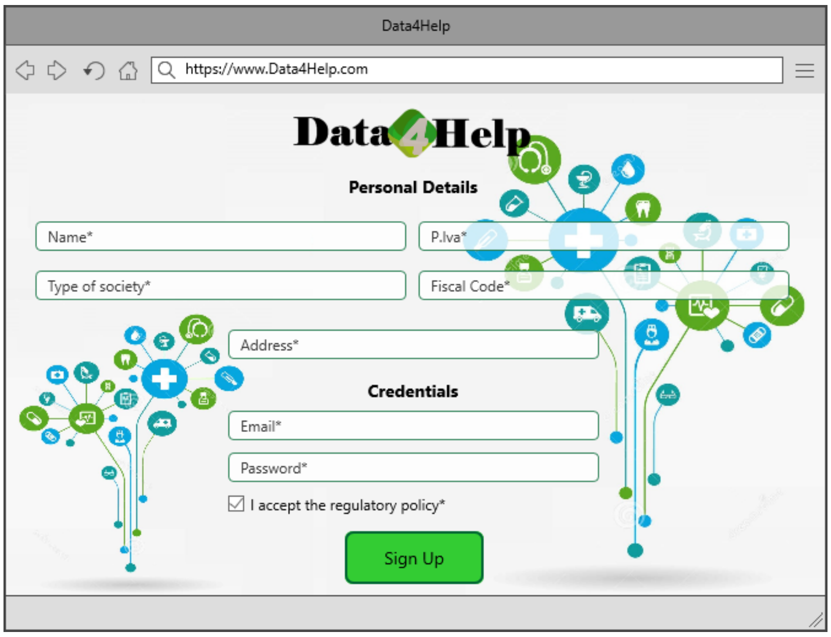


Figure 3.4: Mock up - Registration

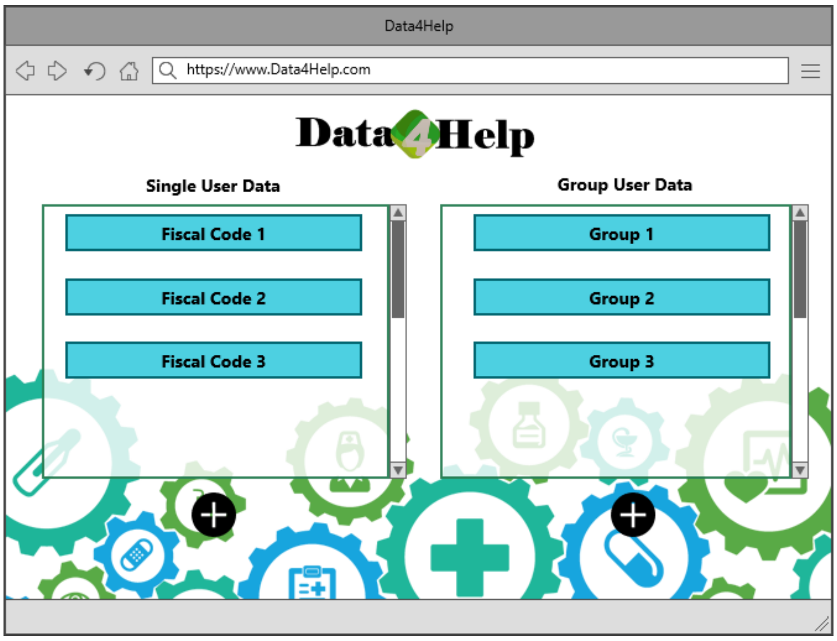
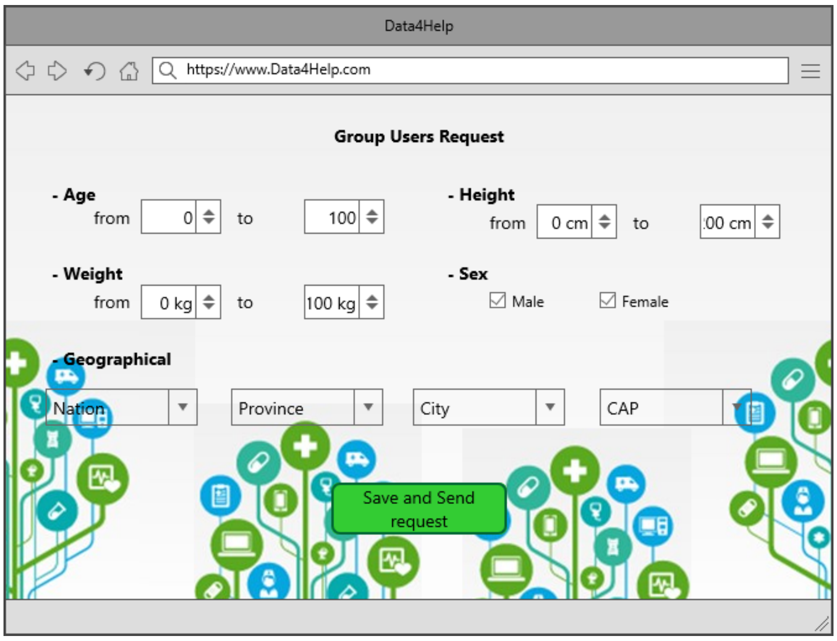


Figure 3.5: Mock up - Main Scene



Group Users Request

- Age from 0 to 100
- Height from 0 cm to 00 cm
- Weight from 0 kg to 100 kg
- Sex ☑ Male ☑ Female
- Geographical Nation Province City CAP

Figure 3.6: Mock up - Settings for a group of users request.

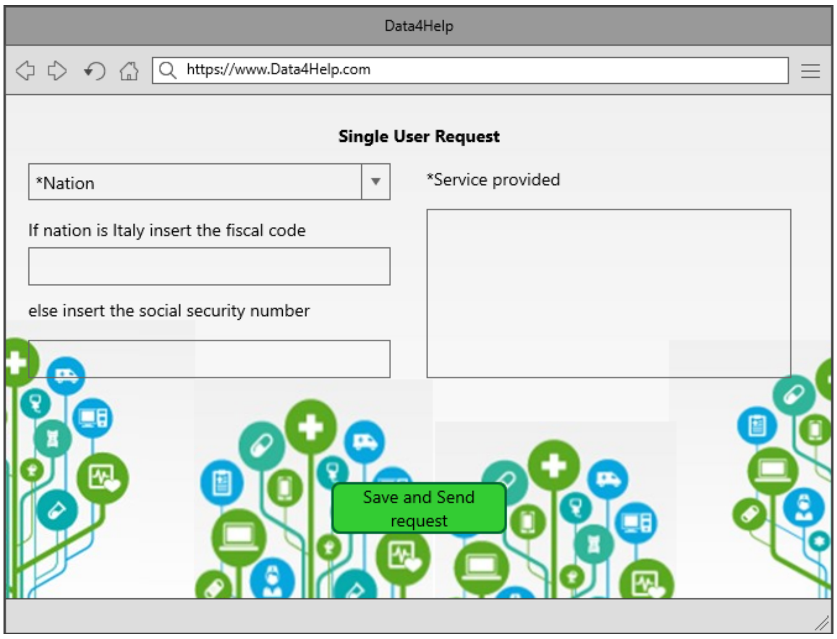
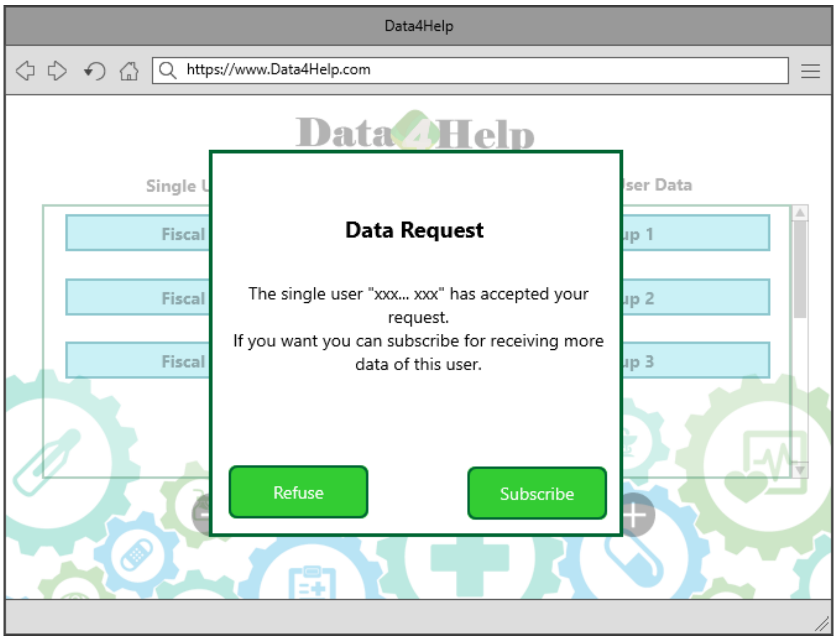

Data4Help
https://www.Data4Help.com
Single User Request
*Nation
*Service provided
If nation is Italy insert the fiscal code
else insert the social security number
Save and Send request

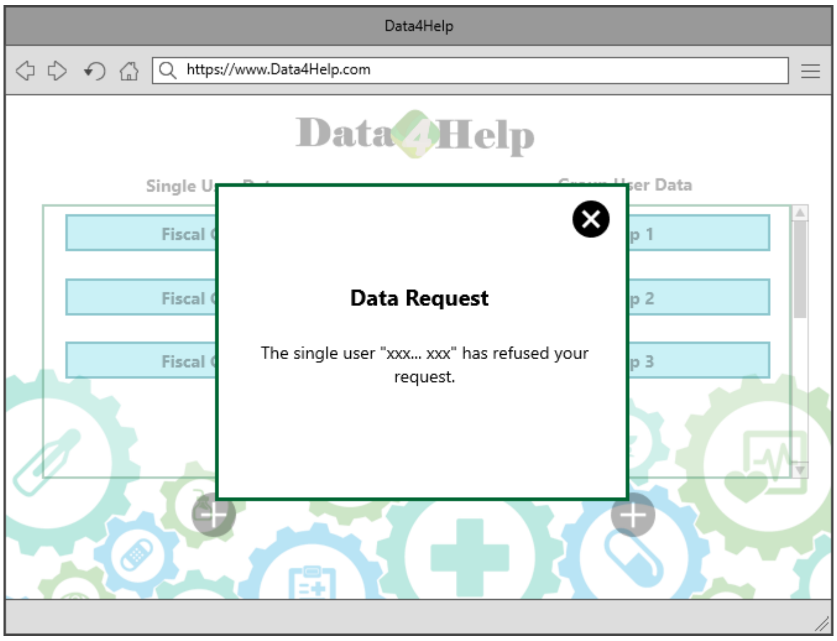
Figure 3.7: Mock up - Settings for a single user request.



Data Request

The single user "xxx... xxx" has accepted your request.
If you want you can subscribe for receiving more data of this user.

Figure 3.8: Mock up - Positive notification related to a single user request. It is similar in case of a positive group of users request.



Data Request

The single user "xxx... xxx" has refused your request.

Figure 3.9: Mock up - Negative notification related to a single user request. It is similar in case of a negative group of users request.

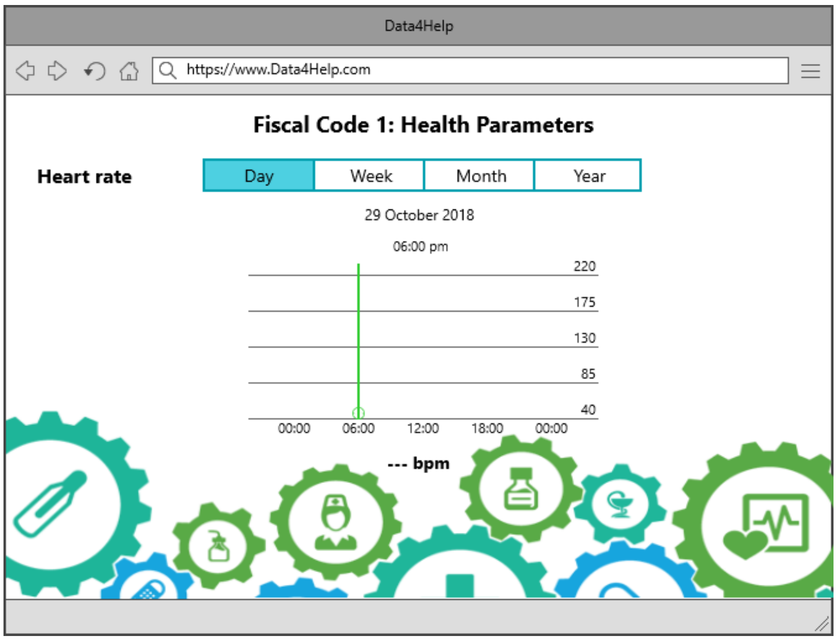


Figure 3.10: Mock up - Area realted to a fiscal code.

Section 4

Requirements Traceability

The purpose of this document is to meet all goals and requirements that have been specified in the RASD. In particular this section associate to goals and requirements the design components thought to guarantee them.
The following lines provide relations between goals, which are re-insert here to better understand the matching, requirements and components.
All requirements aren't report here because this section would become too confuse, they can be found in section 3.3 of the RASD.

[**G1**] Allows a person to register and to have a personal area to which he/she can access with his credentials.
(Requirements - **R1 - R2 - R3 - R3.1 - R3.2 - R3.3**)

- UserService

[**G2**] Allows the third party to register and to have a personal area to which it can access with his credentials.
(Requirements - **R4 - R5 - R6 - R6.1 - R6.2 - R6.3**)

- ThirdPartyService

[**G3**] Allows the third party to require data.

[**G3.1**] Third party can require single person's data.
(Requirements - **R7**)

* ThirdPartyService
* RequestUserService

[**G3.2**] Third party can require anonymized data of group of people.
(Requirements - **R8 - R8.1 - R8.2**)

* ThirdPartyService

      * RequestGroupService

[G4] Allows the user to accept or not to let a third party to have access to his/her data.
(Requirements - **R9 - R10**)

- UserService
- RequestUserService

[G5] Allows third party to subscribe for new data.
(Requirements - **R11 - R12 - R13 - R14**)

- ThirdPartyService
- SubscribeService

[G6] Allows third party to see users' or groups of users' data obtained through a successful request.
(Requirements - **R15 - R16**)

- ThirdPartyService
- RequestGroupService

[G7] Allows users to monitor their health parameters.
(Requirements - **R20**)

- UserService
- HealthParamService

[G8] Allows users to choose to benefit or not of *ASOS* service on top of *D4H*.
(Requirements - **R17**)

- UserService
- AsosService

[G9] Allows an unhealty user to receive quick help if have the *ASOS* service activated on his account.
(Requirements - **R18 - R19**)

- AsosService
- HealthParamService

Section 5

Impletementation, Integration and Test Plan

5.1 Implementation Plan

Considering some factors, such as the possible dependencies between different components and modules, it is necessary to provide an order in which would be better to implement the various app's elements. All most important services offered by *D4H* are related to the storage of health parameters which are analyzed in case a user has subscribed to the *ASOS service* to provide tempestive medical assistance; they are also sent to third parties which have required data of group of users or of single users.
Starting from the points written above and considering that the implementation plan has also the aim to highlight and modify as early as possible all mistakes and progettual problems found, a track for the order of the components' development is presented in this section.

- The model, which corresponds to the application logic component and directly manages data, logic and rules of the application.
- The persistence unit, which will map each DB's tables and relations to the corresponding objects of the model.
- The controller, which will receive users' inputs and will act accordingly to them on the model.
- The view, which will show a representation of the model to the users.

5.1.1 Model

The structure of the model is presented in the 2.2.2 section of this document; note that the UserService and the ThirdPartyService classes in the class diagram in that section won't be part of the model but those will be part of the controller. This part of the software will be developed immediatly after the database because it contains the objects that will be mapped to the DB's tables and relations via JPA and also all the logic and rules of the application, thing that makes this component a very critical one.

5.1.2 Persistence unit

Via the scope of this unit, it is sensible to develop it after the completition of the model or even in parallel with this one. It is not presented in this document a link map beetween the database and the model but it should be simple to deduce it by the model's class diagram in the section 2.2.2 and the database's entity relationship diagram in the section 2.2.3. The use of OpenJPA gives the possibility to do not implement a real DB because it will be created by the API particular tags.

5.1.3 Controller

The structure and the interactions of the controller with the model are presented in the section 2.2.2 of this document; as already said, note that the controller's classes in the class diagram in that section are the UserService and the ThirdPartyService classes. This part will be implemented after the model because it's scope is to act on it to perform the user's actions, so it's the most sensible decision according to the projectual choices made for this software.

5.1.4 View

The representation of the model that the users will see is presented in the section 3 of this document and also in the section 3 of the RASD. This will be the last part to be developed because it represents the least critical and also the most dynamic component of the software.

5.2 Testing Plan

To develop a well working software it is required to do both the structural testing (white box testing) and the functional testing (black box testing) during the implementation and integration of the product. Here below are both analyzed more in the details.

5.2.1 Structural testing

For the coverage of paths, statements, edges and conditions it will be required at least a percentage of 90 for the model's part because of the high risk of errors, faults and failures that lays in the nature of this component; for the other parts of the software will be enought a coverage percentage of 80. The process to do the tests must be:

- Unit testing will be made in parallel with the implementation of each module. It is needed to give particular attention to the modules which contains critical algorithms such as the HealthParamService, SubscribeService and RequestGroupService classes.
- Integration testing will be made after each step of the integration of the modules. In this phase it is needed to give particular attention to the correct integration of the modules which are related to the core functions of the system.
- System testing will be made after the completion of the implementation and integration steps to test the conformity with the functional and non-functional requirements.
- Regression testing will be made constantly during the life cycle of the software after it's release.

5.2.2 Functional testing

The functional testing is left to the software development team; for the develop of this tests it will be needed again to give particular attention to the limit cases and particular cases related to the core functions of the software.

5.3 Integration Plan

The integration order should follow the implementation's one: first of all the model's classes related to the data storage will be integrated with the corresponding database's tables and relations via the persisentence unit, then the other classes of the model will be integrated with each other and with the UserService and ThirdPartyService and last the view's part and its relations with the server.

Each component should be integrated with the others only after it's almost totally completion and with a satisfying unit testing's result to avoid and contain the impact of each possible error, fault and failure on the system. For the same scope, right after a component has been integrated with the system, the relative integration testing should be made.

Appendix A

Appendix

A.1 Software and tools used

The following are all tools used to produce this document:

- **LaTeX:** used as typesetting system to build this document;
- **Draw:** used to draw a few diagrams such as the E-R diagram (online version can be found at `http://www.draw.io`);
- **StarUML:** used to draw some diagrams (the latest version is 3.0.2 and it can be found at `http://staruml.io/download`);
- **Mockplus:** used to draw the mockups (the latest Windows' version is 3.4.1.0 and it can be found at `https://www.mockplus.com/download`);
- **GitHub:** used to work in a distributed way and to manage different versions of the document (it can be download at `https://github.com`);
- **GitHub Desktop:** this is the official GitHub app which offers a simple and user friendly way to contribute to a git project (it can be download at `https://desktop.github.com/`);

A.2 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: ~43 hrs;
- Stefano Bagarin: ~45 hrs;

A.3 Bibliography

- AA 2018/2019 Software Engineering 2 - *Requirements Analysis and Specification Document* - Stefano Bagarin, Alessandra Pasini
- 2015 O'Reilly - *Software Architecture Patterns* - Mark Richards
- some information related to the implementation choices have been found in the following web sites:
  - `http://tomee.apache.org/index.html`
  - `https://www.html.it/guide/guida-mysql/` in particular lessons 12 and from 44 to 50
  - `https://apihandyman.io/do-you-really-know-why-you-prefer\-rest-over-rpc/`
  - `https://docs.oracle.com/html/E13946_01/ejb3_overview_why.html`
- slides of *Software Engineering I - II*