POLITECNICO
MILANO 1863

TrackMe

Software Engineering II - Prof. Elisabetta Di Nitto

Design Document

Michele Gatti, Federica Gianotti, Mathyas Giudici

Document version: 1.0
December 10, 2018

Deliverable:	DD
Title:	Design Document
Authors:	Michele Gatti, Federica Gianotti, Mathyas Giudici
Version:	1.0
Date:	December 10, 2018
Download page:	https://github.com/MathyasGiudici/GattiGianottiGiudici
Copyright:	

Contents

Section 1

Introduction

1.1 Purpose

The goal of the Design Document (DD) is to provide a more technical functional description of the system-to-be, in particular it wants to describe the main architectural components, their communication interfaces and their interactions. Moreover it will also present the implementation, integration and testing plan. This type of document is mainly addressed to developers because it provides an accurate vision of all parts of the software which can be taken as a guide during the developement process.

1.2 Scope

The *TrackMe* project, as explaind in the RASD, has three different but connected goals to achieve:

- **Data4Help**: a service that allows third parties to monitor the location and health status of individuals. Through this service third parties can request the access both to the data of some specific individuals, who can accept or refuse sharing their information , and to anonymized data of group of individuals, which will be given only if the number of the members of the group is higher than 1000, according to privacy rules.
- **AutomatedSOS**: a service addressed to elderly people which monitors the health status of the subscribed customers and, when such parameters are below a certain threshold (personalized for every user using the data from Data4Help), sends to the location of the customer an ambulance, guaranteeing a reaction time less than 5 second from the time the parameters are below the threshold.

- **Track4Run**: a service to track athletes participating in a run. It allows organizers to define the path for a run, participants to enrol to a run and spectators to see on a map the position of all runners during the run. This service will exploit the features offered by Data4Help.

The system-to-be is structured in a four-layered architecture, which will be described in depth in this document and which is designed with the purpose of being maintainable and extensible.
In order to maximize software flexibility and upgradeability, development has been divided into smaller parts, which are as independent as possible. This causes the need to integrate and cooperate the different parts between them and to test their reliability.
An important goal that has been taken into account for software design is also to ensure very high security standards as the software collects users' health data. This data must not be accessible by unauthorized persons.
In order to guarantee this, in this document there is also an explanation of the choices made to ensure data security and a summary of the tests to be implemented and performed.

1.3 Definitions, Acronyms, Abbreviations

ACID: Atomicity, Consistency, Isolation and Durability (Set of properties of database transactions);

API: Application Programming Interface;

DB: Database;

DMBS: Database Management System;

DD: Design Document;

GPS: Global Positioning System;

GUI: Graphical User Interface;

RASD: Requirement Analysis and Specification Document;

UML: Unified Modeling Language;

UX: User eXperience;

1.4 Reference Documents

This document follows the EEE Standard 1016:2009 *System design - Software design descriptions* [1] and complies with the instructions contained in the project assignment document [2].
Furthermore, the constraints imposed in the *Requirements Analysis and Specification Document* [3] have been respected.

1.5 Document Structure

This document is structured as follows:

Section 1: Introduction. A general introduction and overview of the Design Document. It aims giving general but exaustive information about what this document is going explain.

Section 2: Architectural Design. This section contains an overview of the high level components of the system-to-be and then a more detailed description of three architecture views: component view, deployment view and runtime view. Finally it shows the chosen architecture styles and patterns.

Section 3: User Interface Design. This section contains the UX Diagrams of the system to be according to the mockups given in the RASD.

Section 4: Requirements Traceability. This section explains how the requirements defined in the RASD map to the design elements defined in this document.

Section 5: Implementation, Integration and Test Plan. This section identifies the order in which it is planned to implement the subcomponents of the system, the integration of such subcomponents and test the integration.

Section 6: Effort Spent. A summary of the worked time by each member of the group.

At the end there are an **Appendix** and a **Bibliography**.

Section 2

Architectural Design

2.1 Overview

The main high-level components of the system that will be taken into account are structured in four layers, as shown in Figure 2.1 below.

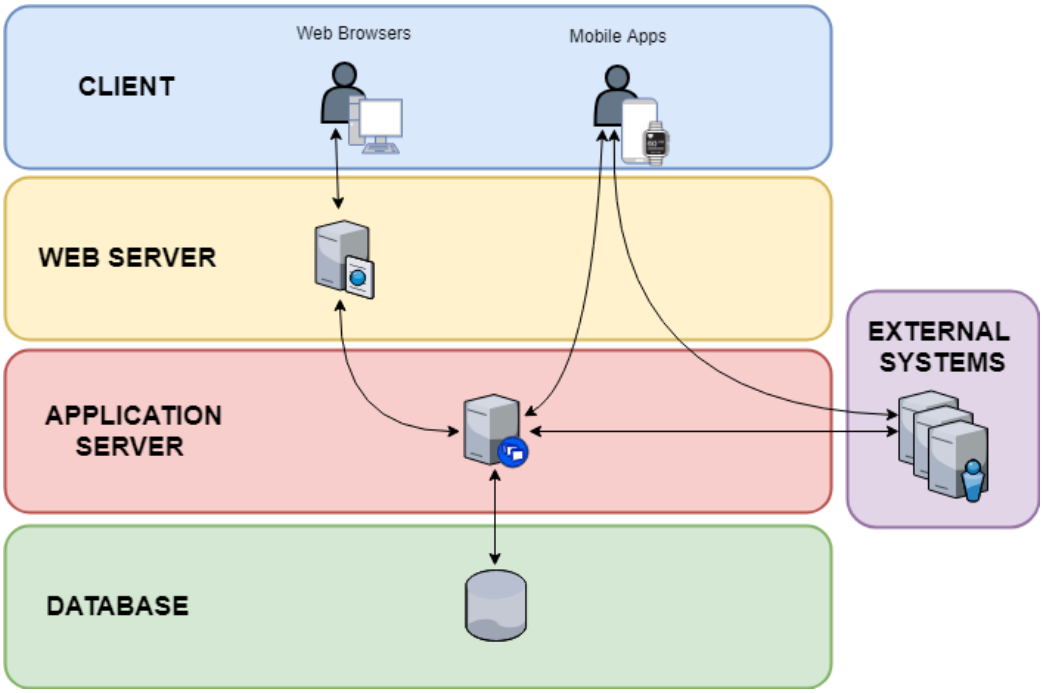

Figure 2.1: *Layered Structure* of the system

The considered high-level components are:

Mobile Applications: The *Presentation Layer* dedicated to mobile devices; it communicates with the Application Server. Notice that in the specific case of *AutomatedSOS* application it comumunicates also with an External Service and has a little part of logic inside of it.

Web Browsers: The *Presentation Layer* dedicated to web browsers; it communicates directly with the Web Server.

Web Server: This is the layer that provides web-pages for the web-based applications; it communicates with the Application Server and with the Web Browsers.

Application Server: This is the layer in which is contained almost all the *logic* for the application (except for the little part contained in *AutomatedSOS* mentioned before); it communicates with the Web Server and with the Database. Moreover it manages almost all the communications with External Services.

Database: The *Data Layer* of the system; it includes all structures and entities responsible for data storage and management. It communicates with the Application Server.

It has been taken the decision to separate the Application and Web Server in order to allow greater scalability. A more detailed description of the intereactions between the described system components is shown in Figure 2.2.

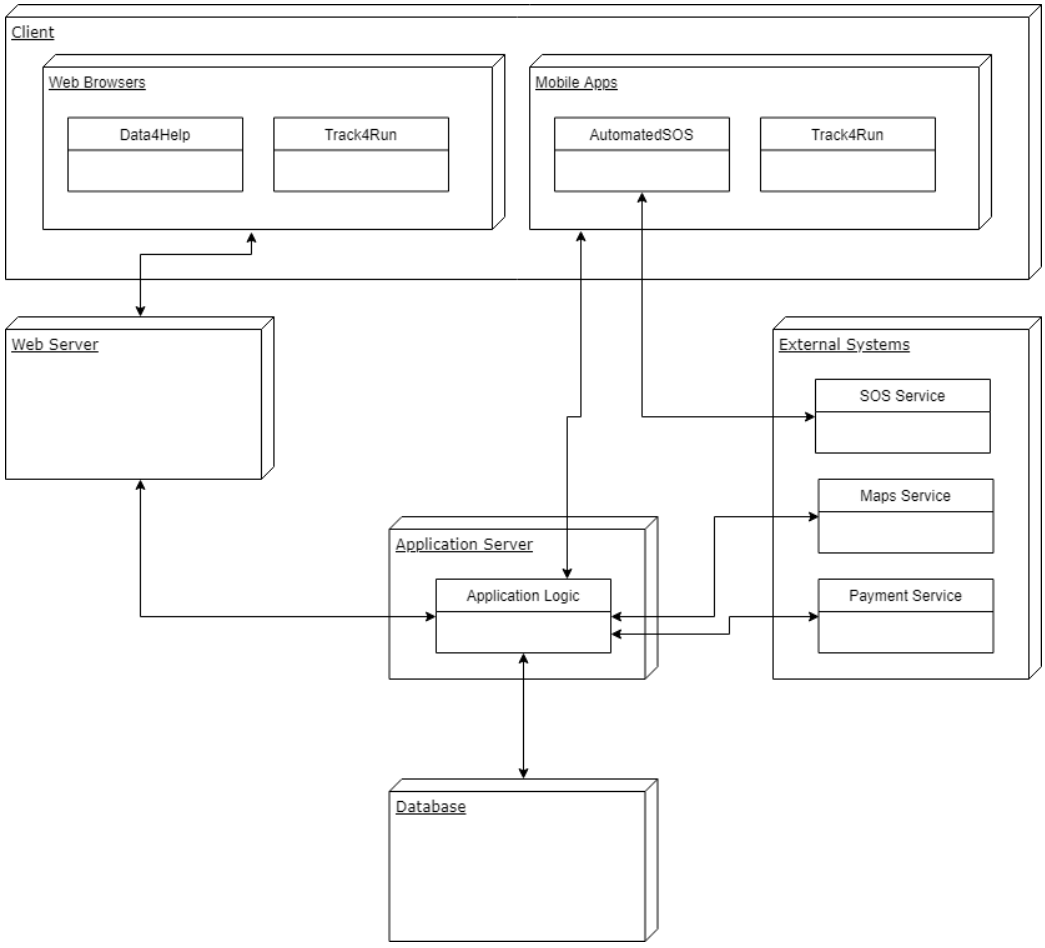

Figure 2.2: *High Level Components* of the system

2.2 Component View

In this section the system will be described in term of its components: their functionalities will be discussed and detailed.
Moreover interfaces among components and among external services will be shown.

2.2.1 Database

The application database will be managed using a Relational DBMS.
It will allow the reading of data, ensuring to the users the possibility to log in, access the application of interest and check the stored data. It will also be used for data manipulation (insertion, modification and deletion). The use of a Relational DBMS guarantees the fundamental properties for a database of this type:

- *Atomicity*: no partial executions of operations;
- *Consistency*: the database is always in a consistent state;
- *Isolation*: each transaction is executed in an isolated and independent way;
- *Durability / Persistence*: changes made are not lost.

The database will offer to the Application Server an interface that it can use to interact with it. The data stored in the database must be considered personal and confidential, therefore, procedures must be implemented to safeguard the stored information.
Particular attention must be paid to the reading permissions granted to users and to the encryption of passwords used to access to the offered services. Below is the designed E-R diagram (Figure 2.3).

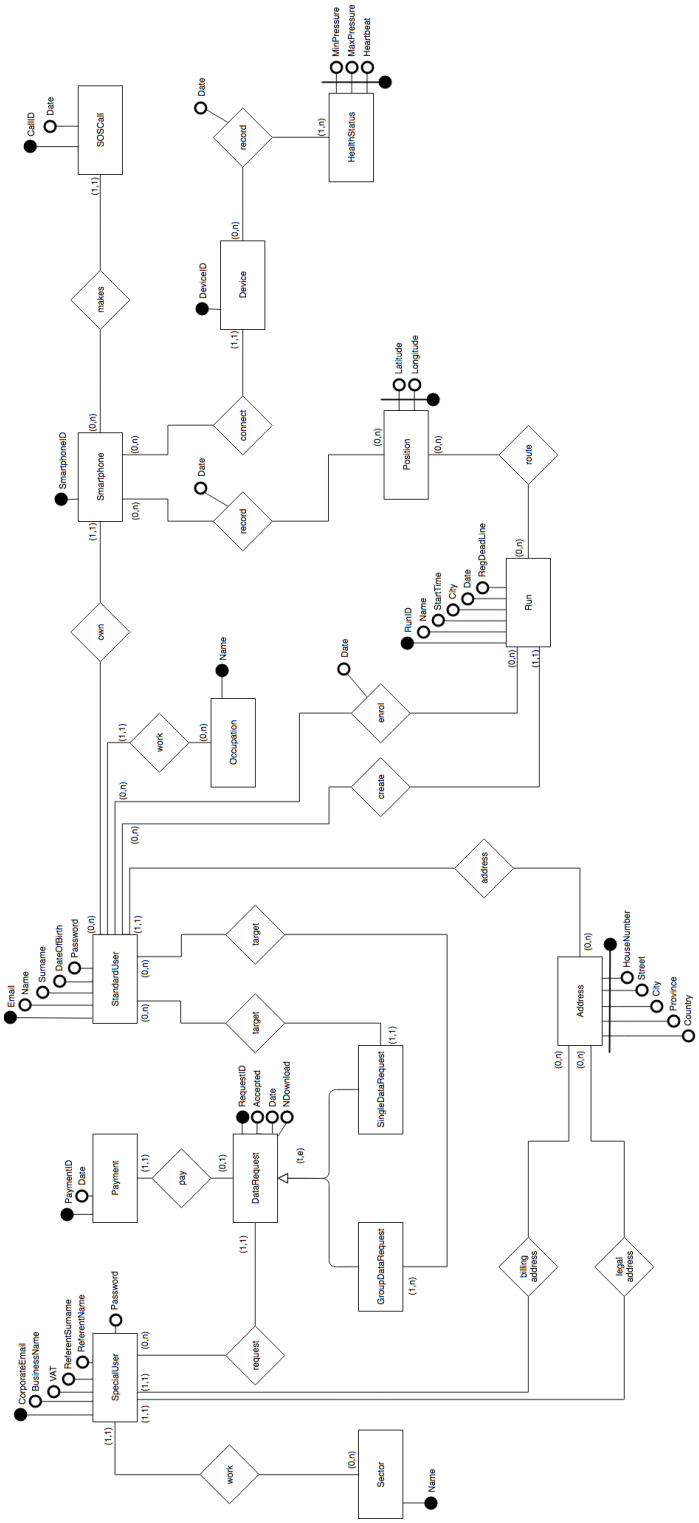

Figure 2.3: *E-R* Diagram

2.2.2 Application Server

This is the crucial layer of the system to be. The main feature of the *Application Server* is to define rules and work-flows of all the functionalities provided by the applications.
The *Application Server* must have intefaces to communicate with the *Web Server*, with *Mobile Apps* and also with some *External Services* (Maps Service and Payment Service).
Moreover, the *Application Server* must be the only entity of the system that will be granted to communicate with the DBMS.
In the brief introduction given below logic modules and their descriptions are presented, while all the connections among them can be seen in the *Global Component View* in Figure 2.4.

Data Collector Service This module offers a specific service to the *Standard User Manager* that allow it to store its acquired data into the *Data4Help* database.

Group Request Manager This module manages the third party requests for *Group Data Requirement*.

Mail Manager This module is a tool for other modules that allow the system to send e-mail (For instance in the registration and enrolment phase).

Maps Manager This module is an handler providing an interface to the external service of maps.

Payment Handler This module is an handler providing an interface to the external service of payment.

Persistence Unit This module is the unique interface to the database's DBMS.

Run Manager This module manages *Runs* in all their functionalities, like creation, deletion and enrolment.

Single Request Manager This module manages the third party requests for *Single Data Requirement*.

Special User Manager This module provides all functionalities related to the *Special User*.

Standard User Manager This module provides all functionalities related to the *Standard User*.

2.2.3 Mobile App

The *Mobile App* must communicate with the *Application Server* through APIs that have to be defined in order to describe the interactions between the two layers and that must be independent from the two implementations. The App UI must be designed user friendly and it has to follow the guidelines provided by the Android and iOS producer. The application must manage the GPS connection of the device and keep track of locations data, moreover it has also to manage the health data recieved from the devices connected to the phone.
The application must provide all collected data to the *Application Server* in order to process them.

2.2.4 AutomatedSOS App

In this Section we want to specify in a more detailed way the modules that compose *AutomatedSOS* application in order to explain better the important activity of detection of possible *Critical Situation*.
All connections among components can be seen in the *AutomatedSOS Component View* in Figure 2.5.

Application Server Handler This module is an handler providing an interface to the *Application Server*.

Background Health Monitor This module checks the health status data that it receives. If an SOS call must be done it calls *SOS Handler*.

GUI This module is the *Graphical* interface to the smartphone.

Health Status Service This module manage detected data and produce statistic for the *User*.

SOS Handler This module is an handler providing an interface to the external service of SOS.

2.2.5 Web Server

The *Web Server* must communicate with the *Application Server* through HTTPS protocol.
The *GUI* of the Web Server must be designed user friendly and it has to follow the guidelines provided by W3C standard (using HTML5, CSS and JS).
The *Control Unit* must manage all the actions played by a user and it must communicate with the *Application Server* through APIs, that are already explained in the *Mobile App* Section.

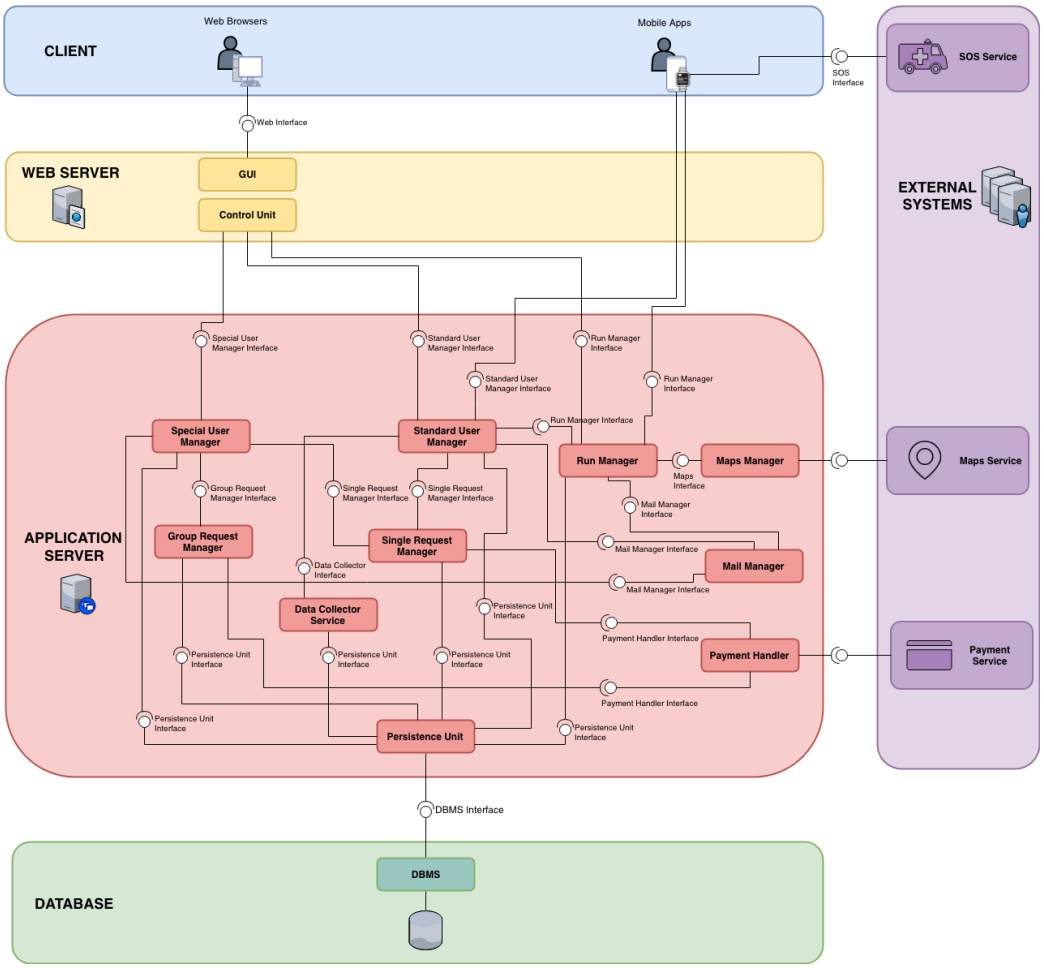

Figure 2.4: *Global Component View*

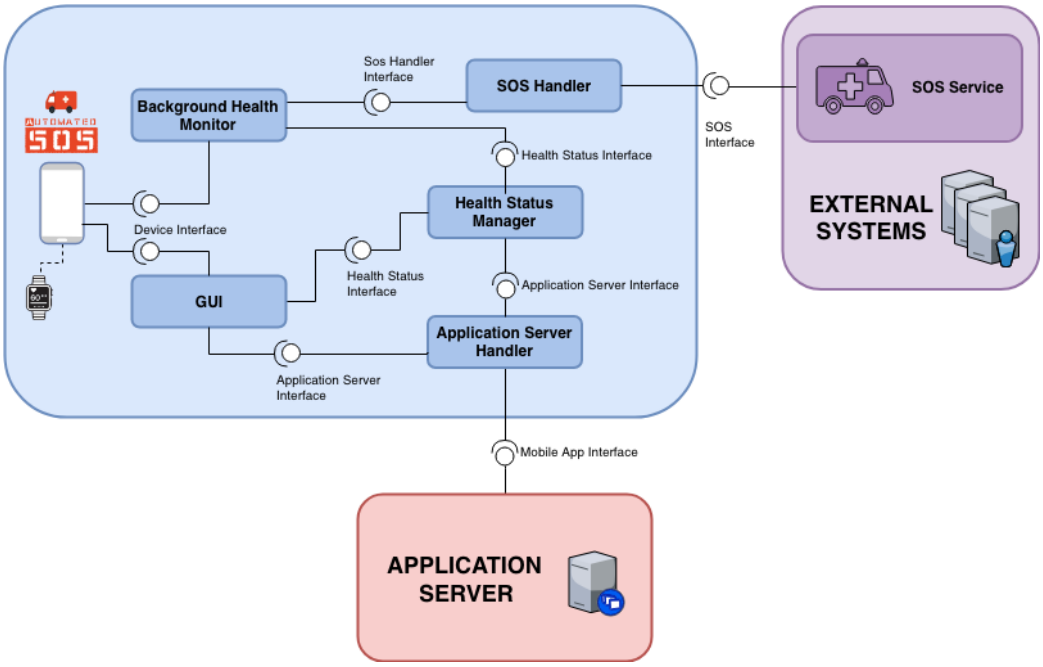

Figure 2.5: *AutomatedSOS Component View*

2.3 Deployment View

Below is the deployment diagram of the system to be (Figure 2.6).

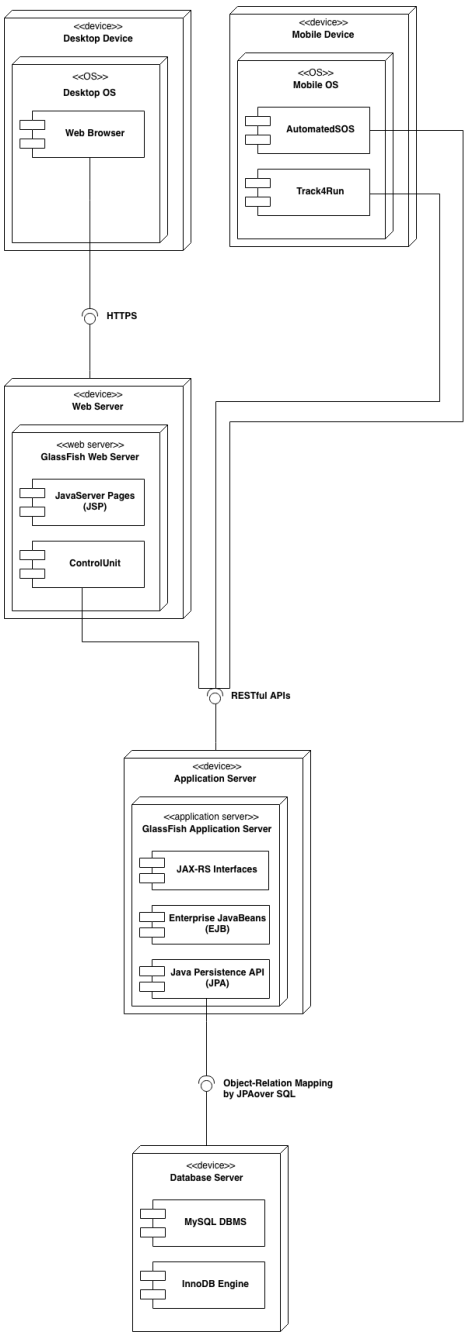

Figure 2.6: Deployment Diagram of the system to be

2.4 Runtime View

In this Section we want to specify the behaviour of our system. Some relevant cases are selected and explained using Sequence Diagrams.

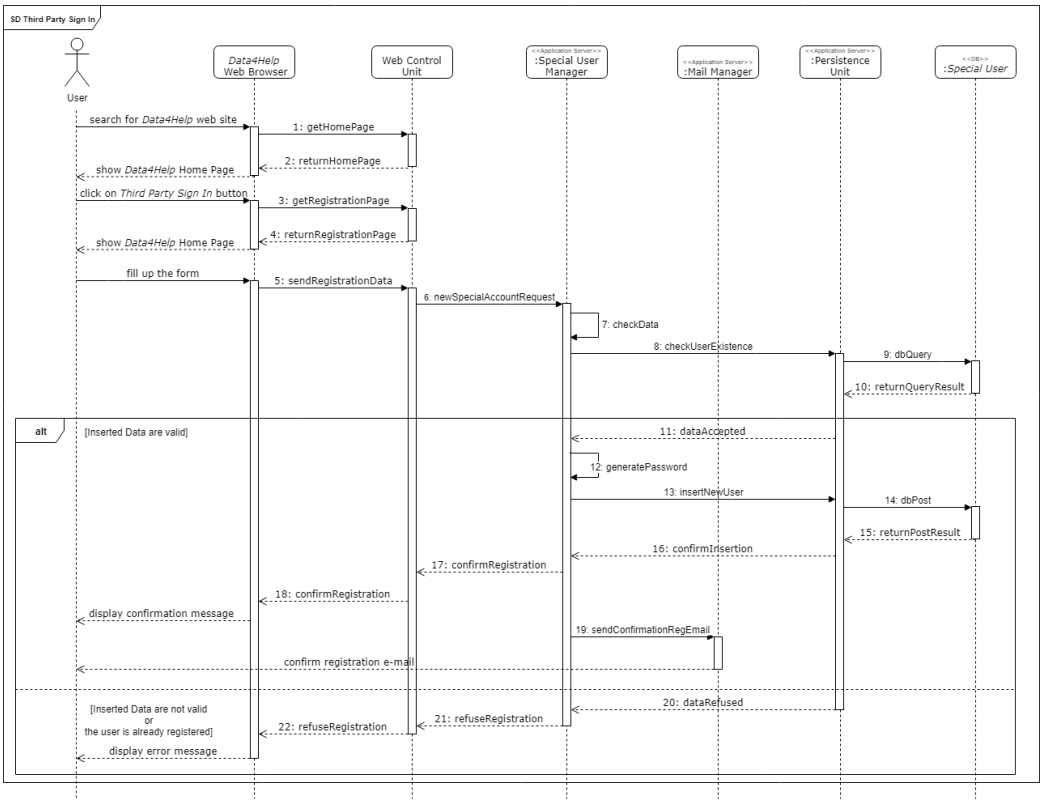

Figure 2.7: *Third Party Sign In* sequence diagram: it is described the process through which a third party can register itself to *Data4Help* services, obviously using *Data4Help*'s web site, since it is the only available platform for third parties

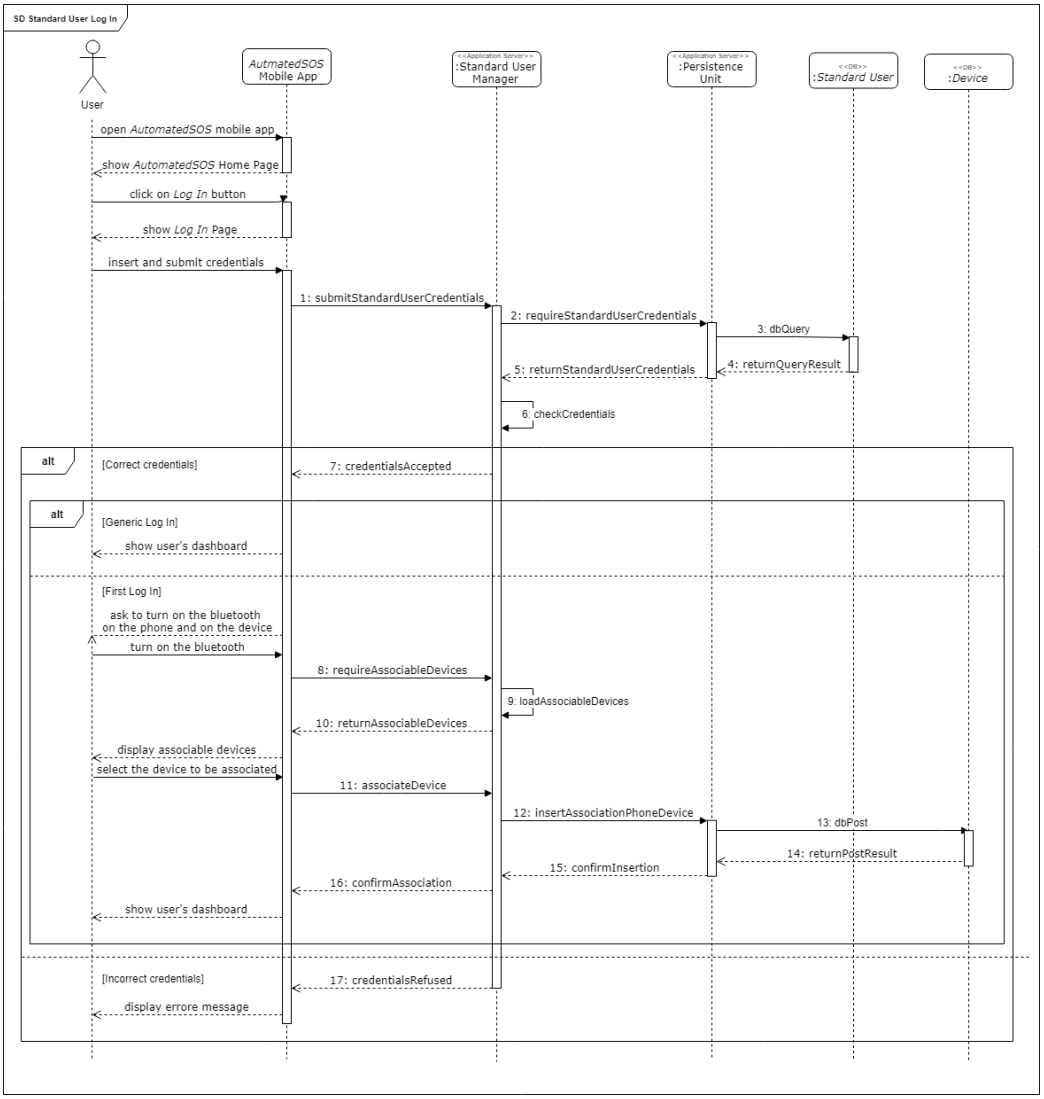

Figure 2.8: *Standard User Log In* sequence diagram: it is described the process through which a standard user can access to *AutomatedSOS* application. (Obviosly the process is exactly the same for *Track4Run* application)

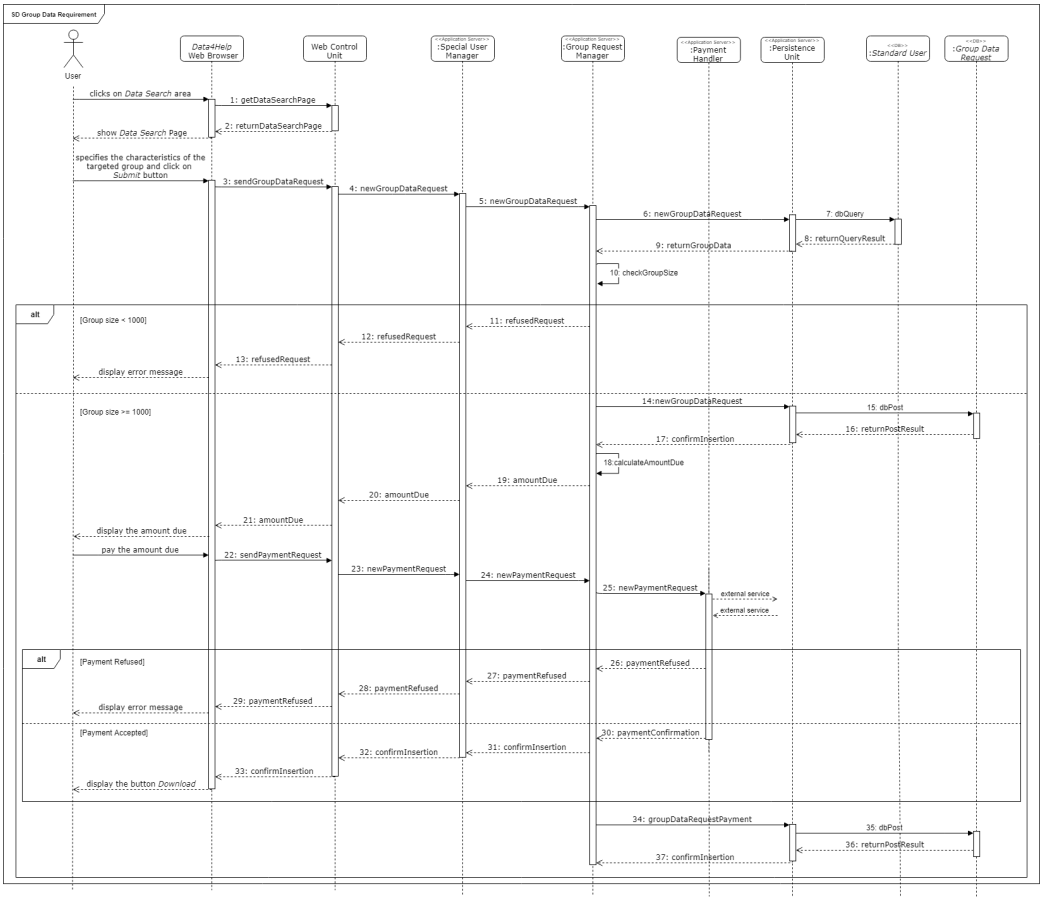

Figure 2.9: *Group Data Request* sequence diagram: it is described the process through which a third party can require data of a group of people and then pay them

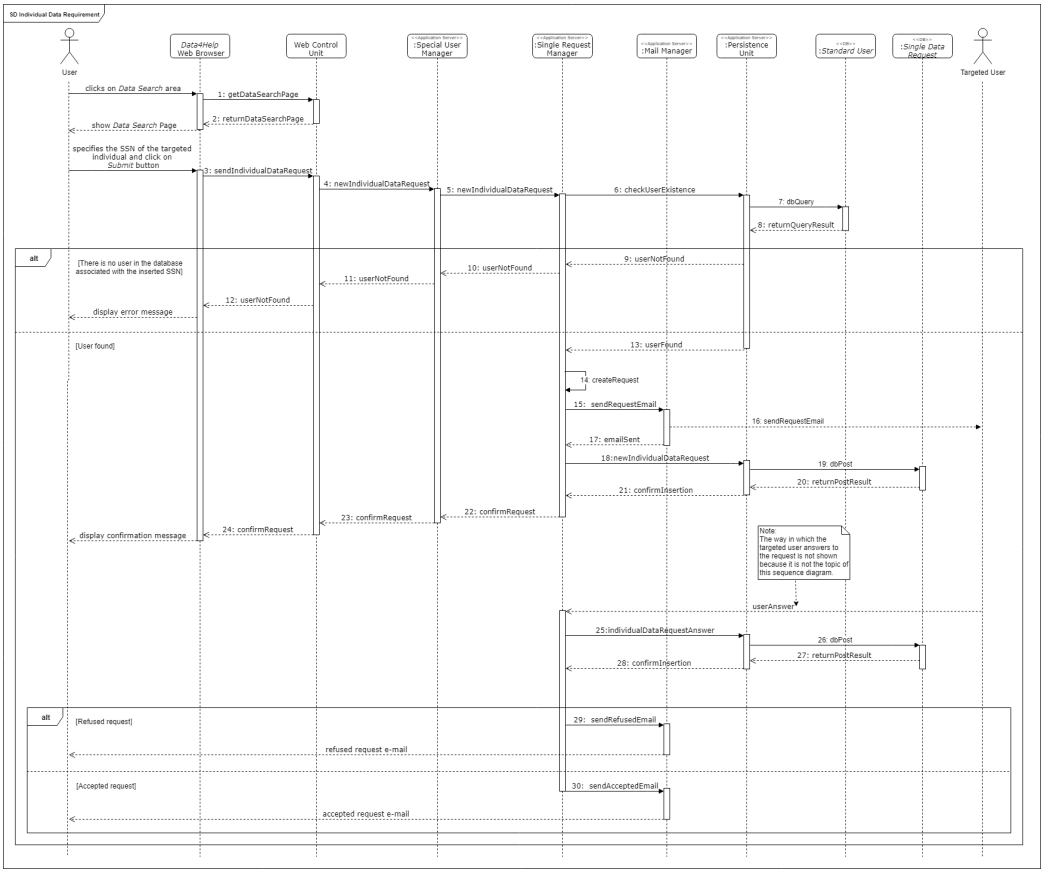

Figure 2.10: *Individual Data Request* sequence diagram: it is described the process through which a third party can require data of an individual user and wait for a response. It is not described the way in which the targeted user answers to the request because it is not the topic of this sequence diagram. It is not even described the way in which the third party pays and eventually downloads the data because it is not considerable and it has just been explained in the *Group Data Requst* sequence diagram

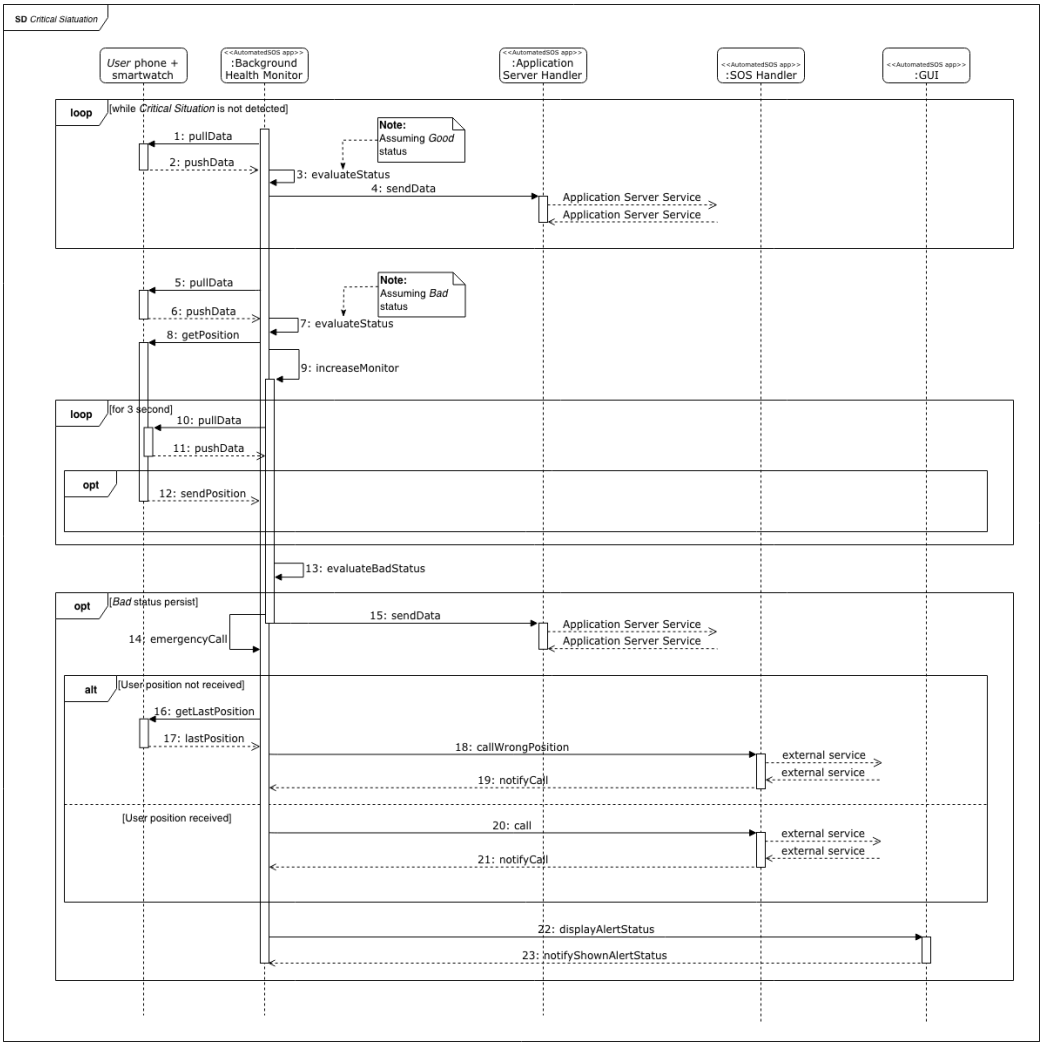

Figure 2.11: *Critical Situation* sequence diagram: it is described the process through which a critical situation is detected by *AutomatedSOS* application + device. It is not described the way in which *AutomatedSOS* application stored the acquired data because it is not considerable for this topic

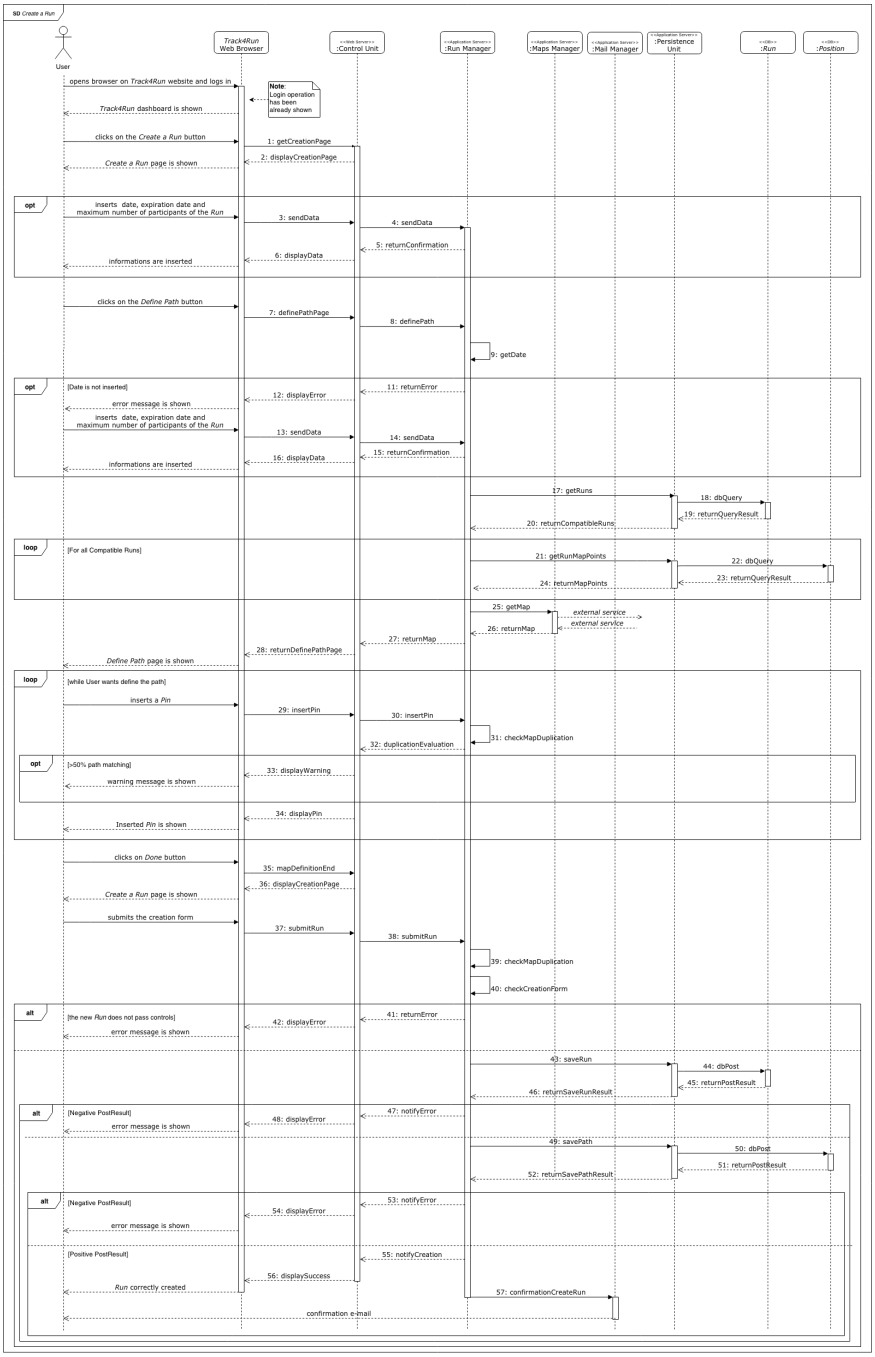

Figure 2.12: *Create a Run* sequence diagram: it is described the process through which a standard user can create a new run

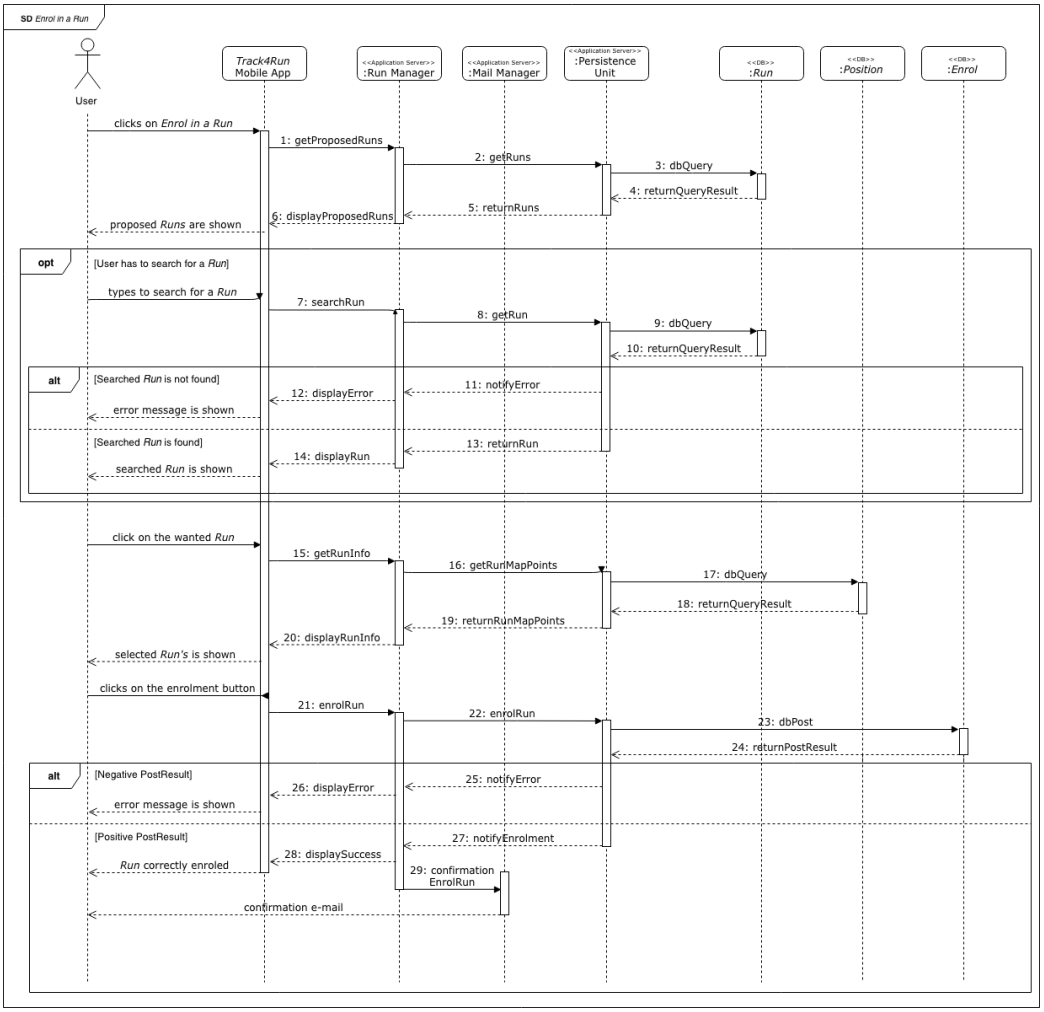

Figure 2.13: *Enrol in a Run* sequence diagram: it is described the process through which a standard user can enrol himself/herself in a run

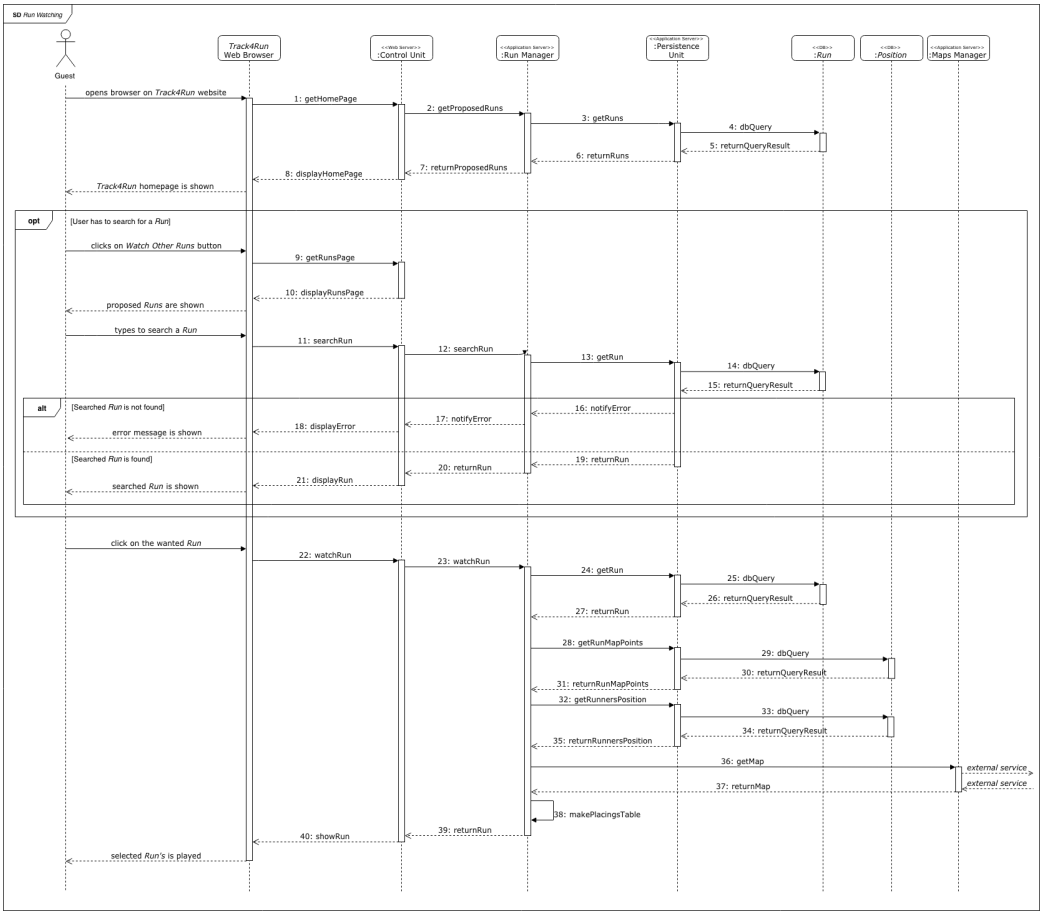

Figure 2.14: *Run Watching* sequence diagram: it is described the process through which a not registered user can watch a run. Notice that he/she can access directly to part of the services offered by *Run Manager* without any log in

2.5 Component Interfaces

This section contains some detailed informations about the interfaces between the components of the system.
The following description focuses the attention over the actions performed by a procedure of a certain component after that it is called by other components.
Huge part of these actions could be seen played in the *Sequence Diagrams* (Section 2.4).

2.5.1 Application Server

Data Collector Service

Following procedure is called by other components:

- **pushData** This procedure is called by *AutomatedSOS* or by *Track4Run* to send the newly acquired data, that will be sent, as soon as possibile, to the *Persistence Unit*.

Standard User Manager

Following procedures are called by other components:

- **associateDevice** This procedure is used to associate a selected device with the smartphone in use.
- **changeDevice** This procedure is called to start the process of changing the associated device.
- **changePassowrd** This procedure is called to change the password of the *Standard User*'s account.
- **deleteProfile** This procedure is called to delete the *Standard User*'s profile.
- **modifyPersonalInformations** This procedure is called to modify personal informations (among the modifiable ones: city of residence, address and occupation.)
- **newStandardAccountRequest** This procedure is called to start a *Standard User* registration process that can be then accepted or refused.

- **requireAssociableDevices** This procedure is used to induce the system to load the list of associable devices.
- **submitStandardUserCredentials** This procedure is called to let (or not) a *Standard User* log in into one of *Track Me* services.

Following procedures are called internally to the component:

- **checkCredentials** This procedure is called to check whether the credentials inserted by the *Standard User* during the login match with the ones saved in the system's database.
- **checkData** This procedure is called to check if the *Standard User* has inserted correctly all the required data.
- **generatePassword** This procedure is called to generate a random password to be assigned to the newly created *Standard User*.
- **loadAssociableDevices** This procedure is called to load the list of associable devices.

Special User Manager

Following procedures are called by other components:

- **changePassowrd** This procedure is called to change the password of the *Special User*'s account.
- **deleteProfile** This procedure is called to delete the *Special User*'s profile.
- **modifyPersonalInformations** This procedure is called to modify personal informations (among the modifiable ones: legal address, billing address, corporate e-mail address, sector and all the data regarding the legal representative)
- **newGroupDataRequest** This procedure is called to start the process of requiring data of a group of people and to send the useful data to the specialized component (*Group Request Manager*).
- **newIndividualDataRequest** This procedure is called to start the process of requiring to a *Individual User* the sharing of his/her data and to send the useful data to the specialized component (*Single Request Manager*).

- **newPaymentRequest** This procedure is called to start the process of paying some required data calling the corresponding procedure on the more specialized component (*Group Request Manager*).
- **newSpecialAccountRequest** This procedure is called to start a *Special User* registration process that can be then accepted or refused.
- **submitSpecialUserCredentials** This procedure is called to let (or not) a *Special User* log in into one of *Track Me* services.

Following procedures are called internally to the component:

- **checkCredentials** This procedure is called to check whether the credentials inserted by the *Special User* during the login match with the ones saved in the system's database.
- **checkData** This procedure is called to check if the *Special User* has inserted correctly all the required data.
- **generatePassword** This procedure is called to generate a random password to be assigned to the newly created *Special User*.

Group Request Manager

Following procedures are called by other components:

- **newGroupDataRequest** This procedure is called to start the process of requiring data of a group of pepole.
- **newPaymentRequest** This procedure is called to start the process of paying some required data calling the corresponding procedure on the dedicated component (*Payment Handler*).

Following procedures are called internally to the component:

- **calculateAmountDue** This procedure is called to calculate the amount due to download some required available data.
- **checkGroupSize** This procedure is called to check whether the size of the required group of people is over 10000 or not.

Single Request Manager

Following procedures are called by other components:

- **newAnswerRequest** This procedure is called when the answer given by the targeted *Standard User* is recieved.
- **newIndividualDataRequest** This procedure is called to start the process of requiring to an *Individual User* the sharing of his/her data.
- **newPaymentRequest** This procedure is called to start the process of paying some required data calling the corresponding procedure on the dedicated component (*Payment Handler*).

Following procedures are called internally to the component:

- **calculateAmountDue** This procedure is called to calculate the amount due to download some required available data.
- **createRequest** This procedure is called to generate a sharing data request that has to be sent to an individual user.

Mail Manager

Following procedure is called by other components:

- **confirmationCreateRun** This procedure sends a confirmation e-mail to the *User* that successfully has created a *Run*.
- **confirmationDeleteEnrolment** This procedure sends a confirmation e-mail to the *User* that successfully has delete an enrolment in a *Run*.
- **confirmationDeleteRun** This procedure sends a confirmation e-mail to the *User* that successfully has deleted a *Run*.
- **confirmationEnrolRun** This procedure sends a confirmation e-mail to the *User* that successfully has enrolled in a *Run*.
- **notifyDeleteRun** This procedure sends a notification e-mail to the *Users* that previously have enrolled in a *Run* that has been deleted.

- **sendAcceptedEmail** This procedure is called to inform a *Special User* that an *Individual User*, of which it has required the data, has accepted its request and so data will be immediatly available after completing the payment.
- **sendConfirmationRegEmail** This procedure is called to give the *User* his/her/its password for the newly created account and to confirm the creation it.
- **sendRefusedEmail** This procedure is called to inform a *Special User* that an *Individual User*, of which it has required the data , has refused its request.
- **sendRequestEmail** This procedure is called to send a sharing data request to an *Individual User*.

Maps Manager

Following procedure is called by other components:

- **getMap** This procedure retrives a map from the external service.

Run Manger

Following procedures are called by other components:

- **checkCreationForm** This procedure checks all the fields of a creation for *Create a Run* form.
- **definePath** This procedure is called to open the path definition interface and to prepare the system to manage avoiding of maps duplications.
- **deleteEnrolement** This procedure allow a user to delete an enrolment in a *Run*.
- **deleteRun** This procedure allow a user to delete a *Run*.
- **enrolRun** This procedure allow a user to enrol in a *Run*.
- **getProposedRuns** This procedure is called to retrive *Runs* to propose to the users.
- **getRunInfo** This procedure is called to retrive information of a particular *Run*.

- **insertPin** This procedure is called to insert a *Pin* on a map during hte definition of a path.
- **makePlacingsTable** This procedure is called to create the *Placings Table* of a given *Run* knowing path and runners' position.
- **searchRun** This produce is called to search a given *Run*.
- **sendData** This procedure is used to submit data informations of a *Run* and these data are used to compute functional requirement expecially for the *Create a Run* use case (for instance avoiding 50% of map duplication).
- **submitRun** This produce is called when a *User* has defined a *Run* and wants to publish it.
- **watchRun** This procedure is called to watch a given *Run*.

Following procedures are called internally to the component.

- **checkMapDuplication** This procedure is called to check the duplication of a given path with the compatible ones.
- **getDate** This procedure is used to get the inserted *Date* during the *Create a Run* phase.

Persistence Unit

Following procedures are called by other components:

- **checkExistingUser** This procedure queries the credentials of a specific *User* to check whether they are already present in the database or not.
- **deleteEnrolement** This procedure deletes an enrolment of a user on the DB through the DBMS.
- **deleteRun** This procedure deletes a *Run* on the DB through the DBMS.
- **enrolRun** This procedure posts an enrolment of a user on the DB through the DBMS.

- **getRun** This procedure queries required *Run* on the DB through the DBMS.
- **getRunMapPoints** This procedure queries required *Run path* on the DB through the DBMS.
- **getRunnersPosition** This procedure queries *Runners* position of a certain *Run* on the DB through the DBMS.
- **getRuns** This procedure queries required *Runs* on the DB through the DBMS.
- **groupData** This procedure queries the data of a specified group of people on the DB through the DBMS.
- **groupDataRequestPayment** This procedure post the state of the payment of a certain group data request on the DB through the DBMS.
- **individualDataRequestAnswer** This procedure posts the answer recieved by an *Individual User* for an individual data request on the DB through the DBMS.
- **insertAssociationPhoneDevice** This procedure posts a new association phone-device on the DB through the DBMS.
- **insertNewUser** This procedure posts a new *User* on the DB through the DBMS.
- **newGroupDataRequest** This procedure posts a new group data request on the DB through the DBMS.
- **newIndividualDataRequest** This procedure posts a new individual data request on the DB through the DBMS.
- **requireSpecialUserCredentials** This procedure queries the credentials of a *Special User* (its email-password) to check if they correspond with the ones inserted during the login.
- **requireStandardUserCredentials** This procedure queries the credentials of a *Standard User* (his/her email-password) to check if they correspond with the ones inserted during the login.
- **savePath** This procedure posts a new path for a *Run* on the DB through the DBMS.
- **saveRun** This procedure posts a new *Run* on the DB through the DBMS.

Payment Handler

Following procedures is called by other components:

- **newPaymentRequest** This procedure is called to pay some required data, it calls the dedicated external service.

2.5.2 AutomatedSOS

Application Server Handler

Following procedures are called by other components:

- **getData** This procedure is called to get data information of the user from the DB via *Application Server*.
- **login** This procedure is called to log in *AutomatedSOS*.
- **sendData** This procedure is called to send data information of the user to the *Application Server* and store them into the DB.

Background Health Monitor

Following procedures are called by other components:

- **lastPosition** This procedure is called to send the last *User*'s position to *AutomatedSOS*.
- **pushData** This procedure is called to send data that *AutomatedSOS* has to evaluate.
- **sendPosition** This procedure is called to send user's position to *AutomatedSOS*.

Following procedures are called internally to the component.

- **evaluateBadStatus** This procedure is called when the system is in an *Alerted Status* and has to evaluate if it has to call an ambulance or the *Alerted Status* was generated from an abnormal measure.
- **evaluateStatus** This procedure evaluates the data received from the user's device.
- **increaseMonitor** This procedure is called at the *Alerted Status* beginning to increase the life paramethers detection.

Health Status Manager

Following procedures are called by other components:

- **makeStatistics** This procedure is called to get data informations of the user and makes weekly and monthly statistics to visualize.

GUI

Following procedures are called by other components:

- **displayAlertStatus** This procedure shows on the smartphone's screen the emergency call in progress.
- **displayDashboard** This procedure shows on the smartphone's screen the dashboard of a logged user.
- **displayLogin** This procedure shows on the smartphone's screen the login page.
- **displayStatistics** This procedure shows on the smartphone's screen the statistics' page of a logged user.

Following procedures are called internally to the component.

- **logout** This procedure logs out the user and gets back to the *Login* page of the application.

SOS Handler

Following procedures are called by other components:

- **call** This produce call an ambulance and send it the user's position.
- **callWrongPosition** This produce call an ambulance notifying that there is an error in getting the user's position and the lastone is sended.

2.6 Selected Architectural Styles and Patterns

Following there are the choices made regarding architectural styles and patterns.

2.6.1 Client and Server

A Client Server architecture has been chosen for the system implementation: it allows the exchange of information between Internet-connected devices used by users and the centralized server that collects data and offers *Data4Help services*. In particular:

- The *AutomatedSOS* and *Track4Run* applications installed on users' devices are connected to the Application Server that receives and processes data.
- The desktop eb browsers used by users (both standard and special) to access the services offered by Data4Help communicate with the Web Server that provides an HTTPS interface.
- The Web Server is seen as a server by the web browser and as a client by the application server that allows it to process requests.
- The Application Server is seen as a server by the web server and applications installed on smartphones and as a client from the database that receives and processes its requests (queries).
- The database is seen as a server by the application server that sends the requests and waits for the answers.

2.6.2 Multi-tiered architecture

A multi-tiered client-server architecture is used to ensure the separation between the MVC model levels (Model, View, Controller). In this way a greater stability of the system is also guaranteed:

- In case of failure of the application server, the applications installed on users' devices continue to work and collect data. In particular, *AutomatedSOS* is still able to make an emergency call. This remains valid also in the case in which the device is temporarily without an Internet connection (but with emergency calls available). Once the Application Server or the Internet connection has been restored, the device will reconnect itself and send the data collected during the period of absence of the service to the server.

- In case of failure of the Web Server, the Application Server remains active and the applications installed on users' devices continue to work correctly (and to send data correctly to the server). The services offered through the use of the web browser will not be available.

2.6.3 Thin Client

The services accessible through web browsers are considered thin clients because the *User* doesn't have to install software belonging to *Data4Help* on his personal computer. All the necessary information required to use the service are offered by the Web Server through HTTPS, including a graphical interface and the application logic. Also the *Track4Run* app is considered thin client because, even if the application that provides the graphical user interface is installed on the phone, most of the application logic (and in particular the one used to manage the races) is based on the application server to which it connects.

2.6.4 Thick Client

The *AutomatedSOS* app has to be considered a thick client: in the app, in addition to the graphical interface and the logic related to the acquisition of user data, there is also the logic that allows the phone to make a call to the external service used for handling emergency calls. Thanks to this, the app is able to detect a state of emergency and alert the ambulance even if the connection with the server is not available.

2.6.5 Model-View-Control

The MVC model identifies three main components in the management of a client-server application. They are the Model, the View and the Controller. The following diagram shows the management of these three levels according to the use of thin or thick clients (2.15).

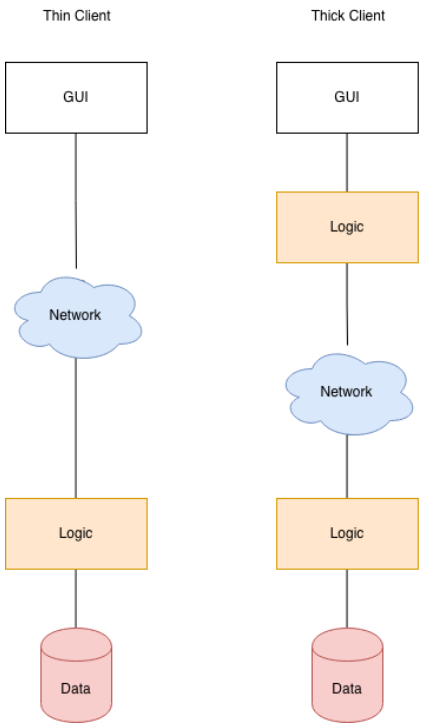

Figure 2.15: Possible management of the client-server connection according to the MVC model. Thin or thick client

2.7 Other Design Decisions

2.7.1 User authentication

In order to guarantee the confidentiality of user data, the access to the system, both for standard users and for special users, takes place by entering a username and password. The username used for the access coincides with the email address used during registration.

2.7.2 Password storing

The passwords given to the users are saved in the database after being encrypted using a SHA512 algorithm. This way, the passwords that have to be used to access to *Data4Help* services are not stored in plaintext in the user's database, but they are the result of a transformation from which, even with very powerful computers, it is very difficult to trace the original ones. This way even using the normal Internet connection, if an intruder succedes in picking up the string containing the password, he won't be able to trace the password used to access the service.

2.7.3 Security in the transfer of information

The transfer of data between the clients and the server is done using the most modern data encryption systems. For example, the Web Server offers an HTTPS interface to the web browser.

2.7.4 Maps

In order to manage the *Runs* offered by *Track4Run*, the external Google Maps service is used. Longitude and latitude are used to locate a position within the map. To ensure a better service to the user who is responsible for creating the *Run*, it is also given to him/her the opportunity to create the route using the features provided by Google Maps (such as search by address, search for places of public interest, and so on). In the database the *Run* path will still be saved as a sequence of positions in terms of longitude and latitude.

2.7.5 Emergency call

Emergency calls are handled using an external service. This ensures that even in the case that the data network is not available, or the application server is out of service, it is still possible to call for help. In fact, the External Service works via the Internet connection or, in the absence of a connection, via SMS. When the *AutomatedSOS* application detects an anomaly, it invokes the external service and sends it the location of the device and a precompiled text containing information related to the health status of the user. Finally this information is sent to the rescue service by the External Service.

Section 3

User Interface Design

3.1 UX Diagrams

The purpose of this section is to show the possible screens that the user might encounter during the use of the system. The displayed information and the possible interactions by the user are also shown in these diagrams.
To allow a better reading of the diagrams themselves, they have been divided into 4 parts:

- Diagram of *Data4Help* (accessible from Web Wrowser), Figure 3.1.
- Diagram of *Track4Run* (accessible from Web Wrowser), Figure 3.2.
- Diagram of *Track4Run* (accessible from the App on Smartphone), Figure 3.3.
- Diagram of *AutomatedSOS* (accessible from the App on Smartphone), Figure 3.4.

The AutomatedSOS diagram contains a screen unreachable by the user. It is the screen that is displayed when an emergency is detected and then an ambulance is alerted. For this screen to be displayed, the user must be logged in to the app and have connected a device.

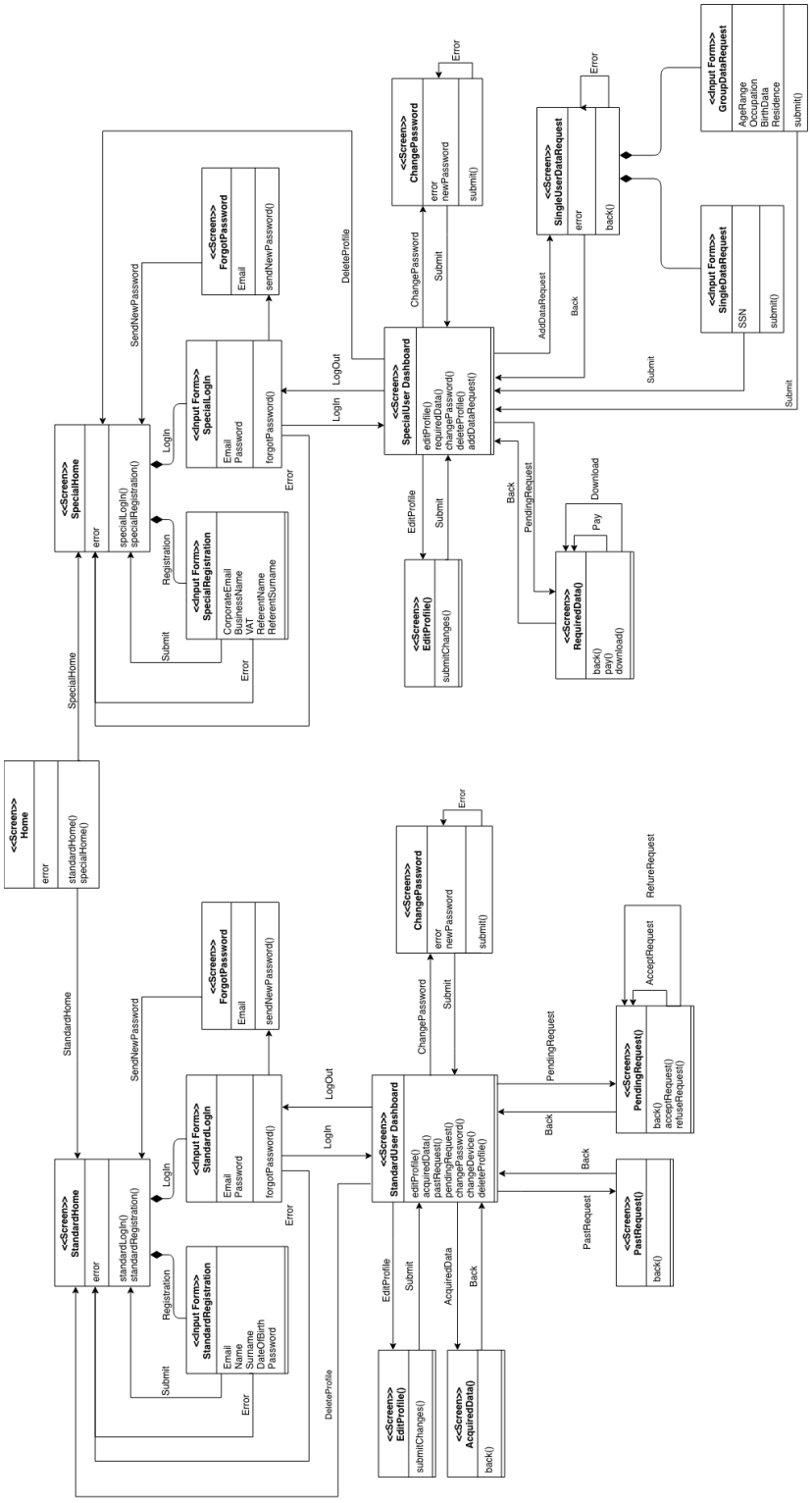

Figure 3.1: UX Diagram *Data4Help* (Web)

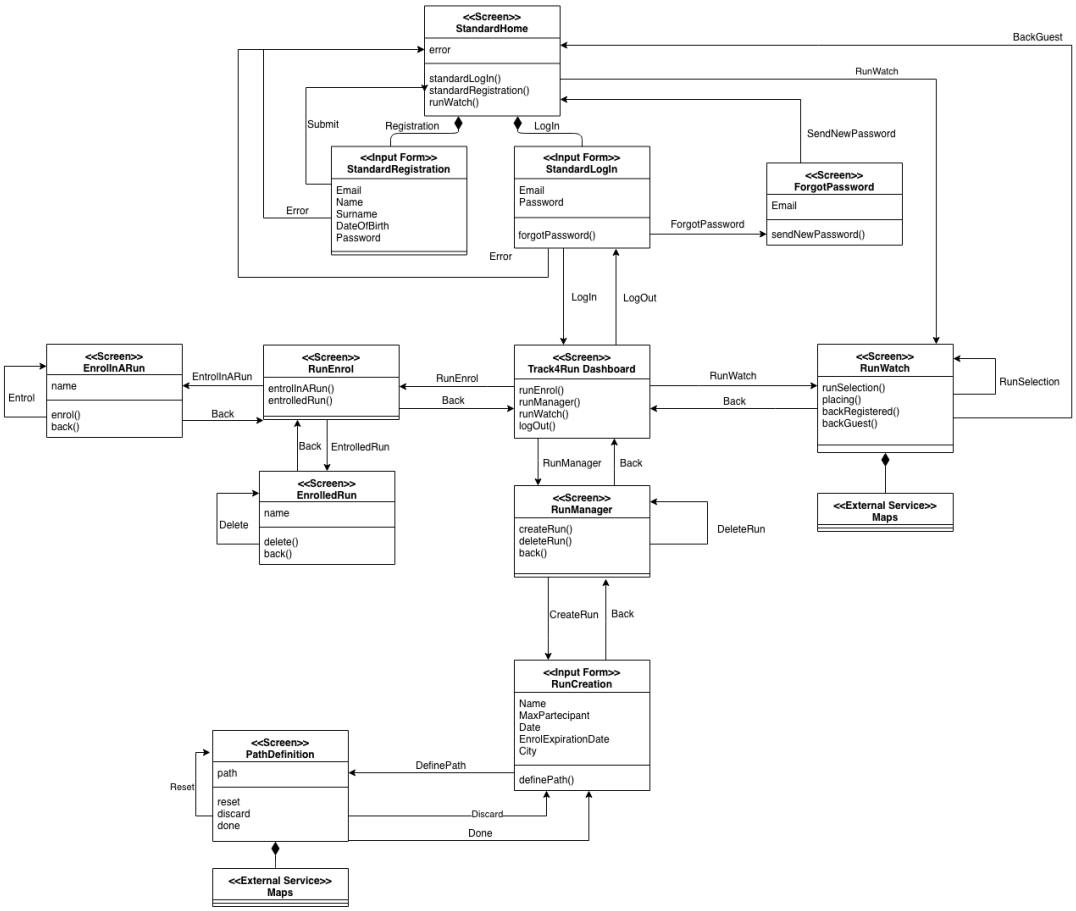

Figure 3.2: UX Diagram *Track4Run* (Web)

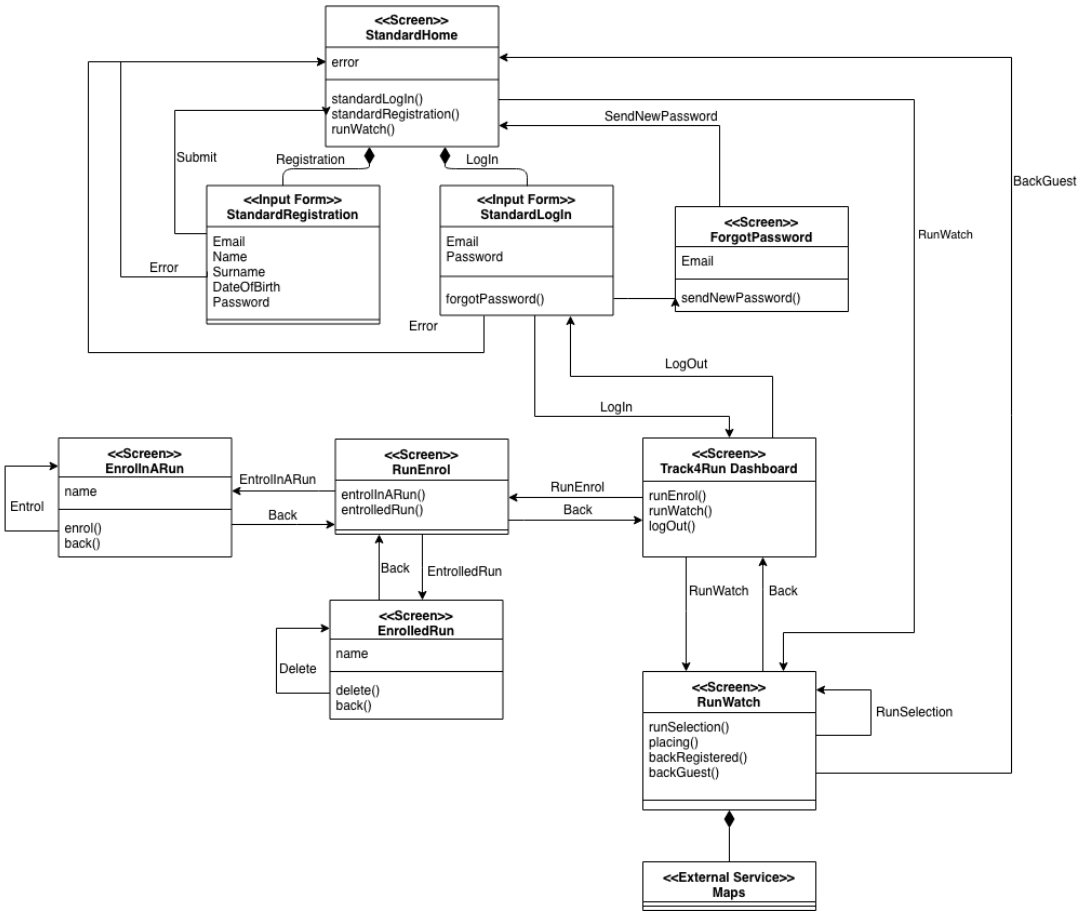

Figure 3.3: UX Diagram *Track4Run* (App)

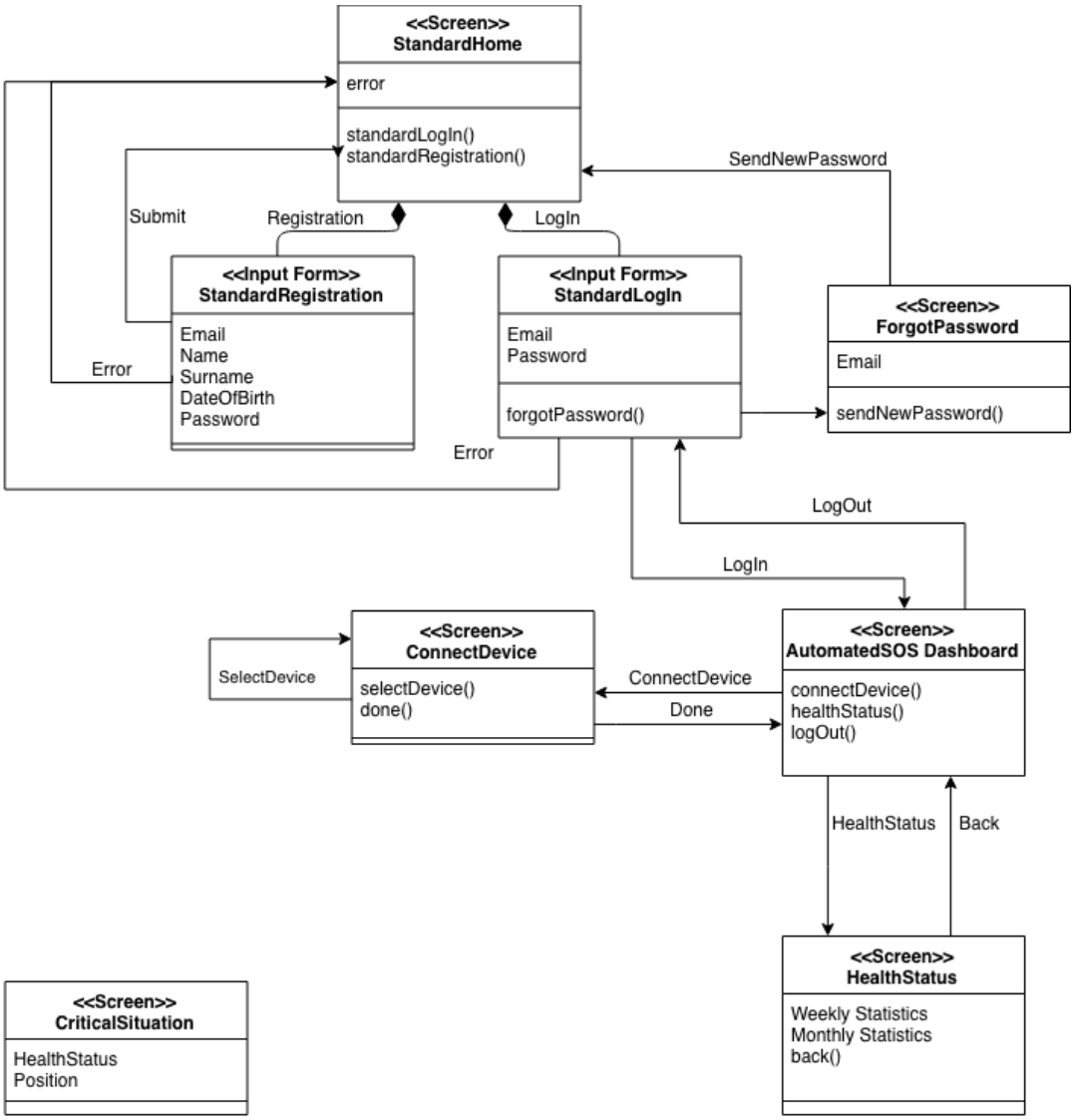

Figure 3.4: UX Diagram *AutomatedSOS* (App)

3.2 User Interfaces

For user interfaces, please refer to the Requirements Analysis and Specification Document where the mockups have been included and explained.

Section 4

Requirements Traceability

4.1 Functional Requirements

In the table 4.1 below it is highlighted the mapping between DD components and RASD goals and functional requirements. Notice that it is missing the component *Persistence Unit* because the aim of the table is to focus on application logic components while the *Persistence Unit* is only an interface with the database's DBMS.

4.2 AutomatedSOS Functional Requirements

Since there is a part of logic in the *AutomatedSOS* application (explained in section 2.6) there is an additional table below (4.2), dedicated to *AutomatedSOS* specific components with the correspondig achived goals and requirements.

Goal	DD Component	RASD Requirements
1 2 3	Special User Manager	3.2.2 Third Party Sign In 3.2.4 Third Party Log In 3.2.6 Manage Third Party Profile
1 2 3	Standard User Manager	3.2.1 Individual Sign In 3.2.3 Individual Log In 3.2.5 Manage Individual Profile 3.2.7 Individual Data Requirement
4	Single Request Manager	3.2.7 Individual Data Requirement
5	Group Request Manager	3.2.8 Group Data Requirement
6	Data Collector Service Manager	3.2.7 Individual Data Requirement 3.2.8 Group Data Requirement
9 10 11 12 13 14 15	Run Manager	3.2.11 Create a Run 3.2.12 Delete a Run 3.2.13 Enrol in a Run 3.2.14 Delete an Enrolment in a Run 3.2.15 Run Watching
10 15	Maps Manager	3.2.11 Create a Run 3.2.15 Run Watching
1 4 10 11 12 13	Mail Manager	3.2.1 Individual Sign In 3.2.2 Third Party Sign In 3.2.7 Individual Data Requirement 3.2.11 Create a Run 3.2.12 Delete a Run 3.2.13 Enrol in a Run 3.2.14 Delete an Enrolment in a Run
4 5	Payment Handler Manager	3.2.7 Individual Data Requirement 3.2.8 Group Data Requirement

Table 4.1: Mapping between DD components and RASD goals and functional requirements

Goal	DD Component	RASD Requirements
7 8	Background Health Monitor	3.2.9 Health Status Visualization 3.2.10 Critical Situation
8	SOS Handler	3.2.10 Critical Situation
7	Health Status Manager	3.2.9 Health Status Visualization

Table 4.2: Mapping between DD components of *AutomatedSOS* and RASD goals and functional requirements

Section 5

Implementation, Integration and Test Plan

5.1 Software Development Process

Given the detailed specifications available for the development of this software, the life cycle that is most suitable to choose is the *Waterfall* model. In fact, this life cycle involves the sequential execution of the main steps from the collection of information to the maintenance of the software. The steps that must be followed according to this life cycle are shown in figure 5.1. It is very easy to couple the steps carried out up to now with those foreseen by the chosen life cycle.

- Concept Exploration coincides with the analysis of the project assignment document.
- Requirements Analysis coincides with what has been done for the production of the Requirements Analysis and Specification Document.
- System Design coincides with what has been done for the production of this document.

Next steps:

- Implementation
- Testing
- Deployment
- Maintenance

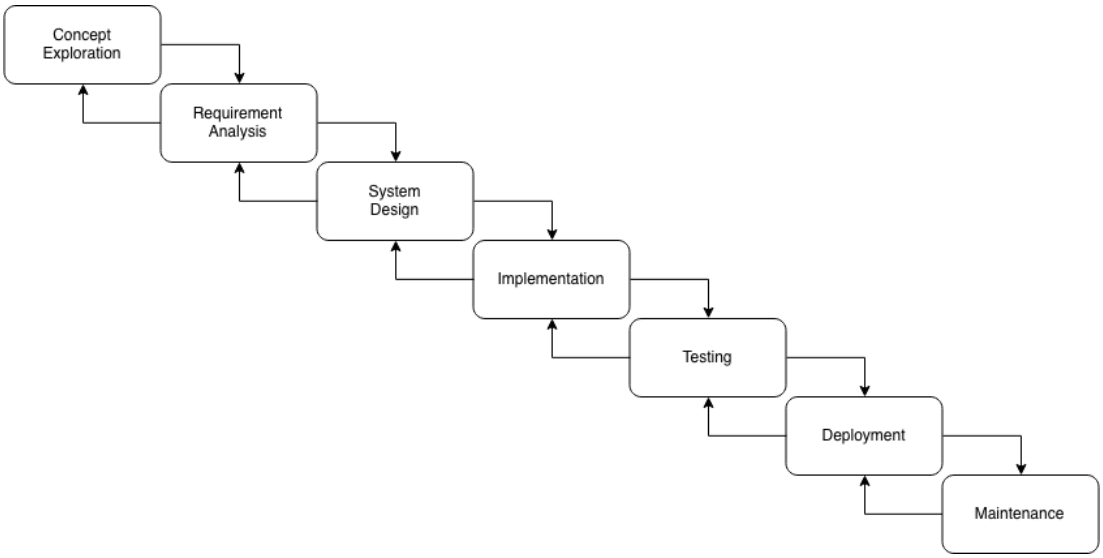

Figure 5.1: *Waterfall* Model

5.2 Implementation Plan

The Architectural Styles and Patterns of the system to be are huge explained in Section 2.6.
The implementation of our system will be done module by module and component by component, the most relevant and complex part of the system to be is the *Controller* of the MVC model.
The development of the modules must be done keeping attention in writing good *Documentation of the Code*, doing *Unit Test* and finally making *Code Inspection and Analysis*; these topics are explained in the Section 5.3.
Below there is a Figure in order to explain the implementation flow, several detailed on the *Application Server* components that are the huge part of the *Controller*.

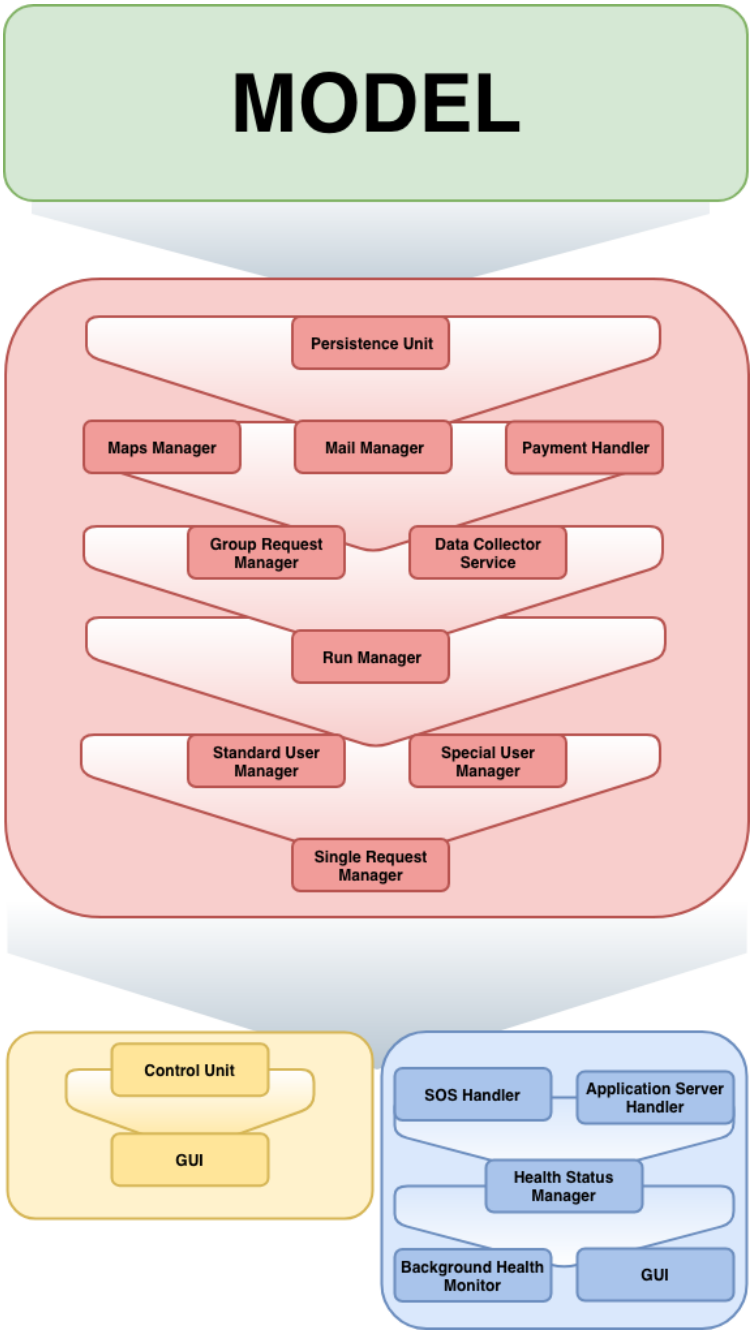

Figure 5.2: *Implementation Plan* of the system

5.2.1 Implementation Plan Discussion

The *Model* is the first part of the system that must be implemented. This choice is supported by the facts that the *Model* is used by most of the components in the *Application Server* and that some part of it are represented in the *Database*. Moreover, this way it is avoided any type of duplication or confusion between what we consider *Model* and what *Controller*.

The second most relevant part of the system that must be developed is the *Controller*, and we find in it components of the *Application Server*.

- **Persistence Unit** is the first component of this part that must be implemented because it manages all communication with the DBMS and communicates with a large part of the components in this part of the system.
- **Maps Manager**, **Mail Manger** and **Payment Handler** are components the provides the majority of services in common among other components and *Maps Manager* and *Payment Handler* are also the components that interfaces with the external services.
- **Group Request Manager** and **Data Collector Service** provides some of the services provides by *Data4Help*. They are at the base of other components so their implementation must be done keeping attention.
- **Run Manager** is the component that allows our system to manage the *Run*'s services, fault tollerance must be guarantee. It is the core of this service and its implementation is crucial. It also provides an interface to the *Standard User Manager* components and it use the *Maps Manager* component.
- **Standard User Manager** and **Special User Manager** are the core of all actions performed by a *Generic User* of *Data4Help*. They communicate with most of the components described before and their implementation has to guarantee very high tolerance to fault and good performance since they represent a possible bottleneck of the application.

- **Single Request Manager** like some previous components it provides one of the core service of *Data4Help*. Its implementation is inserted at this point because it requires some important functionalities of *Standard User Manager* and *Special User Manager*.

The *View* is the last part of the system to be that must be implemented because it needs most of the functionalities of the *Controller*.
The implementation flow of the internal components of *AutomatedSOS* is specified in Figure 5.2 to avoid any possible problems, being a critical service.

5.2.2 Implementation Choices

Database

The choices that have been taken for the *Database* are MySQL 5.7 as relational DBMS and InnoDB as Database Engine. This last choice has been taken to manage concurrency while accessing same tables.

Application Server

The implementation of this important layer is done with Java Enterprise Edition 7 (JEE) using a GlassFish Server. Java Persistence API (JPA) is used to interface with the DBMS, JAX-RS to implement RESTful APIs to interface with mobile application, *Web Server* and also with the *External Servicies*.

Mobile App

The mobile applications must be implemented in two different architecture respecting native languages, Swift for iOS application and Java for Android ones. The communication with the device must be done using the default frameworks of the respective system, moreover the communication with the *Application Server* and *External Services* must be performed with RESTful APIs.

Web Server

As we decided to implement the *Application Server* with JEE, that choice is reflected to this layer. We have only to specify that the implementation of the *GUI* must be performed with HTML5 and CSS; *Control Unit* using JavaServer Pages (JSP).

5.3 Integration and Testing

The Integration and Testing Section provides the main guidelines to explain the integration test phase, describing which tools will be used and planning integration phase in a detailed way.

5.3.1 Entry Criteria

This section describes which are the prerequisites needed before approach the integration phase. We can find three main topics the must be done before starting:

- **Documentation of the Code**: Sometimes the code's documentation is understimate. Good documentation provides tools to review the code and its mean; moreover it is important in huge project, like the *TrackMe* one, where more people will develop the system. It makes easier the understanding of the classes' functionalities and behaviour and it also makes easier their reusing.
  Where only text documentation is not sufficient to explain a certain functionality a formal language of specification like JML (Java Modelling Language) is required.

- **Unit Test**: All classes of the project must be tested with Unit Tests that check for each one its behaviour. Unit Test must be done using JUnit tool and line coverage of 90% is required. Only the *View* part of our project is authorized not to respect the line covarge constraint.

- **Code Inspection and Analysis**: Code Inspection and Analysis is an important phase of the testing part because with an automated tool, such as SonarQube, we can look for code smell and possible bugs of the classes. Solving issues in this phase provides less probability of complex and big problems in the integration phase.

5.3.2 Elements to be Integrated

Our system is structured in a multi-tiered client-server architecture and we can divide our system in four main layers (according to Figure 2.1).
The integration of the different components in a layer must be done component by component and then when issues of internal integration will be fixed, layers integration that concerns communication between components must be performed.

5.3.3 Integration Testing Strategy

To avoid any possibility of complex problems during the integration phase, it must be performed incrementally, when it is possible, during the growing of the system (As it has been done for the implementation phase).
Where integration test is not possible during the developing phase a bottom-up approach of integration must be followed in order to avoid any possible failure of the core components of the system.

5.3.4 Integration Sequence

In this Section the integration flow is explained.
Tables will be used to better explain the components that have a role in the integration activity.

Application Server Components Integration

Huge part of the integration component/component must be done in the *Application Server* that contains a big part of the *Logic* of the system. The integration must be done keeping attention and avoiding any possible issues.

#	**Component**	**Integrated with**
I01	Data Collector Service	Persistence Unit
I02	Group Request Manager	Persistence Unit
I03	Group Request Manager	Payment Handler
I04	Run Manager	Persistence Unit
I05	Run Manager	Maps Manager
I06	Run Manager	Mail Manager
I07	Standard User Manager	Persistence Unit
I08	Special User Manager	Persistence Unit
I09	Standard User Manager	Mail Manager
I10	Special User Manager	Mail Manager
I11	Standard User Manager	Data Collector Service
I12	Special User Manager	Group Request Manager
I13	Standard User Manager	Run Manager
I14	Single Request Manager	Persistence Unit
I15	Single Request Manager	Payment Handler
I16	Special User Manager	Single Request Manager
I17	Standard User Manager	Single Request Manager

Table 5.1: *Application Server* components integration table

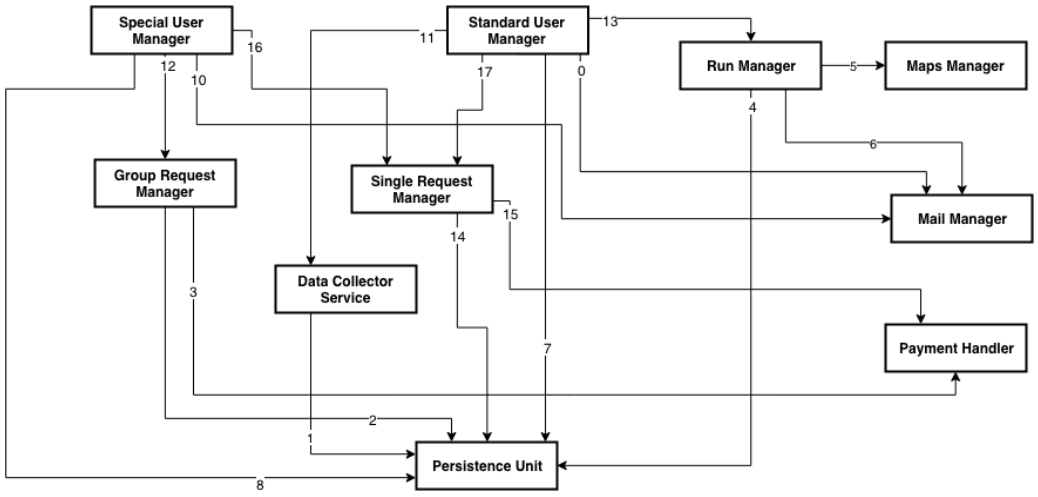

Figure 5.3: *Application Server* components integration diagram

AutomatedSOS Components Integration

Such as in the *Application Server Components Integration* (Section 5.3.4) now we want to focus our attention in the integration of *AutomatedSOS* components.

#	**Component**	**Integrated with**
I01	Health Status Manager	Application Server Handler
I02	Background Health Monitor	SOS Handler
I03	Background Health Monitor	Health Status Manager
I04	GUI	Application Server Handler
I05	GUI	Health Status Manager

Table 5.2: *AutomatedSOS* components integration table

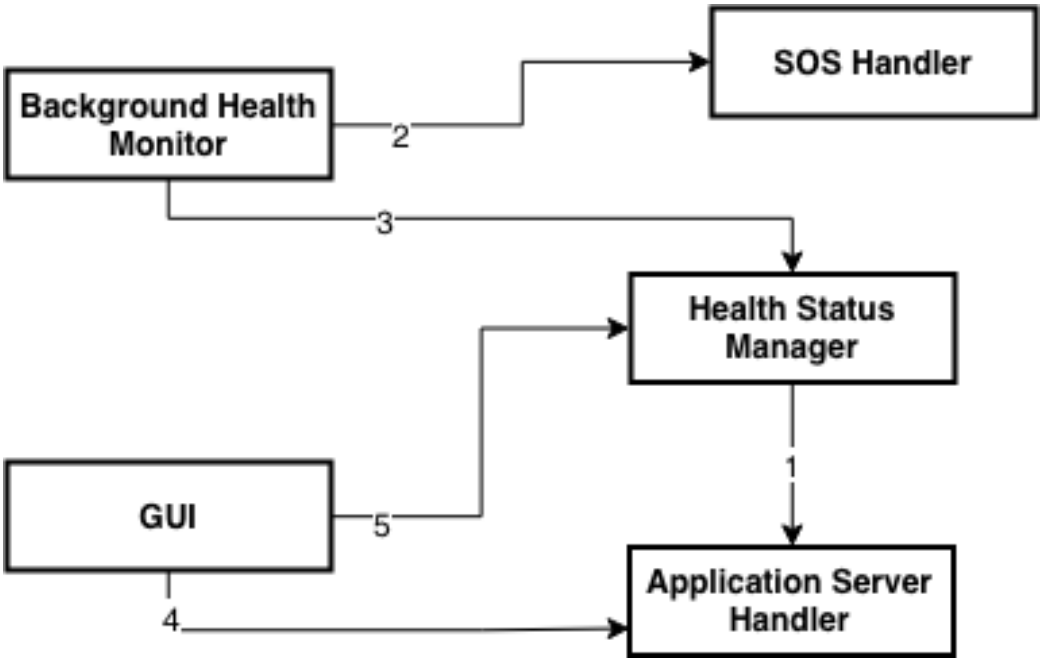

Figure 5.4: *AutomatedSOS* components integration diagram

Subsystems Integration

Now we present the integration sequence of subsystems. As we mentioned before a bottom-up approach must be followed.

#	**Subsystems**	**Component**	**Integrated with**
I01	Database, Application Server	Persistence Unit	DBMS
I02	Application Server, External Servicies	Maps Handler	Maps Service
I03	Application Server, External Servicies	Payment Hendler	Payment Service
I04	Application Server, Web Server	Special User Manager	Control Unit
I05	Application Server, Web Server	Standard User Manager	Control Unit
I06	Application Server, Web Server	Run Manager	Control Unit
I07	Application Server, Mobile Client	Standard User Manager	Application Server Handler
I08	Application Server, Mobile Client	Run Manager	Application Server Handler
I09*	Mobile Client, External Services	SOS Handler	SOS Service

Table 5.3: *Subsystems integration* table. *:this integration activity is valid only for *AutomatedSOS* application

Section 6

Effort Spent

6.1 Michele Gatti

Task	**Hours**
Team work	8
E-R Diagram	3
Deployment View	1
Implementation, Integration and Test Plan	1
Selected Architectural Styles and Patterns	2
Other Design Decisions	3
UX Diagram	7
Software Development Process	2
Requirements Traceability	1
Team Revsion	3
Total	**31**

6.2 Federica Gianotti

Task	**Hours**
Team work	8
Introduction	1
Overview and High Level Components	2
Sequence Diagrams	8
Component Interfaces	3
Implementation, Integration and Test Plan	2
Requirements Traceability	3
UX Diagram	1
Team Revsion	3
Total	**31**

6.3 Mathyas Giudici

Task	**Hours**
Team work	8
Global Component Diagram	4
Sequence Diagrams	7
Component Interfaces	3
Implementation, Integration and Test Plan	4
Requirements Traceability	1
UX Diagram	1
Team Revsion	3
Total	**31**

Appendix A

Appendix

A.1 Software and Tools

- LaTeX used to build this document;
- *GitHub* used to manage the different versions of this document;
- *draw.io* used to draw diagrams;

A.2 Changelog

- **1.0** : First release of this document;

Bibliography

[1] IEEE Standard 1016:2009 *System design - Software design descriptions*

[2] Elisabetta Di Nitto - Software Engineering 2 Slides (AY 2018/2019) *Project goal, schedule and rules*

[3] Michele Gatti, Federica Gianotti, Mathyas Giudici *Requirements Analysis and Specification Document*