POLITECNICO
MILANO 1863

Software Engineering 2 Project

TrackMe

Data4Help & AutomatedSOS

Design Document

A.Y. 2018/2019

Giorgio Polla – 921260

Alberto Tiraboschi – 920775

version 1.0

10/12/2018

Table of Contents

1. Introduction

1.1 Purpose

This is the Design Document (DD) related to the project of TrackMe's new system, as described in the related RASD, reported in the references section. The goal of this document is to give an overall description of the architecture of the system-to-be and to provide guidance in the realization of the software itself.

1.2 Scope

1.2.1 Definition of the problem

TrackMe is a company that wants to develop a software-based service allowing third parties to monitor the location and health status of individuals. This service is called **Data4Help**. The service supports the registration of individuals who, by registering, agree that TrackMe acquires their data (data acquisition can happen through smartwatches or similar devices). Also, it supports the registration of third parties. After registration, these third parties can request:

- Access to the data of some specific individuals (we can assume, for instance, that they know an individual by his/her social security number or fiscal code in Italy). In this case, TrackMe passes the request to the specific individuals who can accept or refuse it.
- Access to anonymized data of groups of individuals (for instance, all those living in a certain geographical area, all those of a specific age range, etc.). These requests are handled directly by TrackMe, that will accept any request for which the number of individuals whose data satisfy the request is higher than 1000, in order to protect the identity of its clients.

As soon as a request for data is approved, TrackMe makes the previously saved data available to the third party. Also, it allows the third party to subscribe to new data and to receive them as soon as they are produced.

1.2.1.2 AutomatedSOS

TrackMe decides to build a new service, called **AutomatedSOS**, on top of Data4Help. AutomatedSOS monitors the health status of the subscribed customers and, when such parameters are below certain thresholds, sends to the location of the customer an ambulance, guaranteeing a reaction time of less than 5 seconds from the time the parameters are below the threshold.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- ***Registered User***: an individual registered to the Data4Help service, and that has therefore given consent to TrackMe to acquire their data.
- *[Registered User's]* ***Data***: information about a RU's location and health status. As no more precise information are reported in the problem description, further information regarding the definition of “health status” are reported in section 2.4.1.1.
- ***Current Saved Data***: it's the data currently saved in the Data4Help system.
- ***Registered Third Party***: a third-party company, independent form TrackMe, that registered itself to the Data4Help service.
- *[Health parameters]* ***threshold***: a precisely defined threshold of the health parameters: if these parameters descend below the said threshold, an individual is considered to be in danger. As no more precise details are reported in the problem description, further information regarding the definition of “heath parameters' threshold” is reported in section 2.4.1.1.
- ***Consenting Registered User*** *[w.r.t. a request from a RTP]*: a RU that has accepted a request for his/hers data by a RTP.

1.3.2 Acronyms

- ***RU***: Registered User
- ***CRU***: Consenting Registered User
- ***RAU***: Registered AutomatedSOS User
- ***RTP***: Registered Third Party
- ***RASD***: Requirement Analysis and Specification Document
- ***DD***: Design Document
- ***API:*** Application Programming Interface
- ***DB:*** Database
- ***MVC:*** Model-View-Cotroller
- ***GUI:*** Graphical User Interface

1.3.3 Abbreviations

- **[G*n*]**: the goal number *n.*
- **[D*n*]**: the domain assumption number *n.*
- **[Rn]**: the functional requirement number *n.*

1.4 Revision history

At the moment there is no revision history to show, as this current version is the only existent one.

1.6 Document structure

This document is composed by six main parts, including a final appendix.

I. ***Introduction.*** In the first part the design document is presented, with the description of the purpose of the document itself and all the information needed in order to better understand the content of the following sections.

II. ***Architectural Design.*** The second part is a comprehensive general description of the system: there is the presentation of the main components of the system-to-be and the relationships between them, in order to provide an understanding of the general working structure of the software. This section will have a particular focus on the main architectural styles and design patterns adopted in the design of the system.

III. ***User Interface Design.*** The third part presents mock-ups of the user interface, in order to offer a preview of how the user interface should look at the end of the development process.

IV. ***Requirements Traceability.*** The fourth part displays the relations between the decision taken during the redaction of the Requirements and Specification Document with the choices adopted in the present Design Document, and in particular each goal and requirement identified in the RASD is here related to the components that aim to fulfil it.

V. ***Implementation, Integration and Test Plan.*** The fifth part presents the decisions that have been made regarding the implementation of the subcomponent, as well as their integration into the complete systems. In this section will be presented also the test plan for the system and the integration.

VI. ***Appendix.*** The sixth and last part comprehends a list the tools used to redact this document and its contents and a detailed report of the hours and efforts spent by each member of the group during the project.

2. Architectural design

2.1 Overview

In this section a brief description of the high level components of the system will be presented, as well as the depiction of the interactions occurring between them.

The general architecture of the system is composed by three logical layers: presentation, application and data.

- **Presentation layer.** The first layer is composed by all the user interface related components, regarding matters of interfacing with the user as well as forwarding the user's choices and actions towards the application layer.
- **Application layer.** The second layer is in charge of the logical part of the platform itself, with the receiving of the input from the presentation layer and the elaboration and forwarding of the appropriate response.
- **Data layer.** The third and final layer is where all persistent data regarding RUs and RTPs is stored: it is a set of Database servers and their DBMS.

As an additional measure of security, periodically the data from the Database Server is copied and saved into a Backup Server. Database Servers, Backup Servers and their DBMS are provided by an external service, and therefore are not further specified in the following paragraphs.

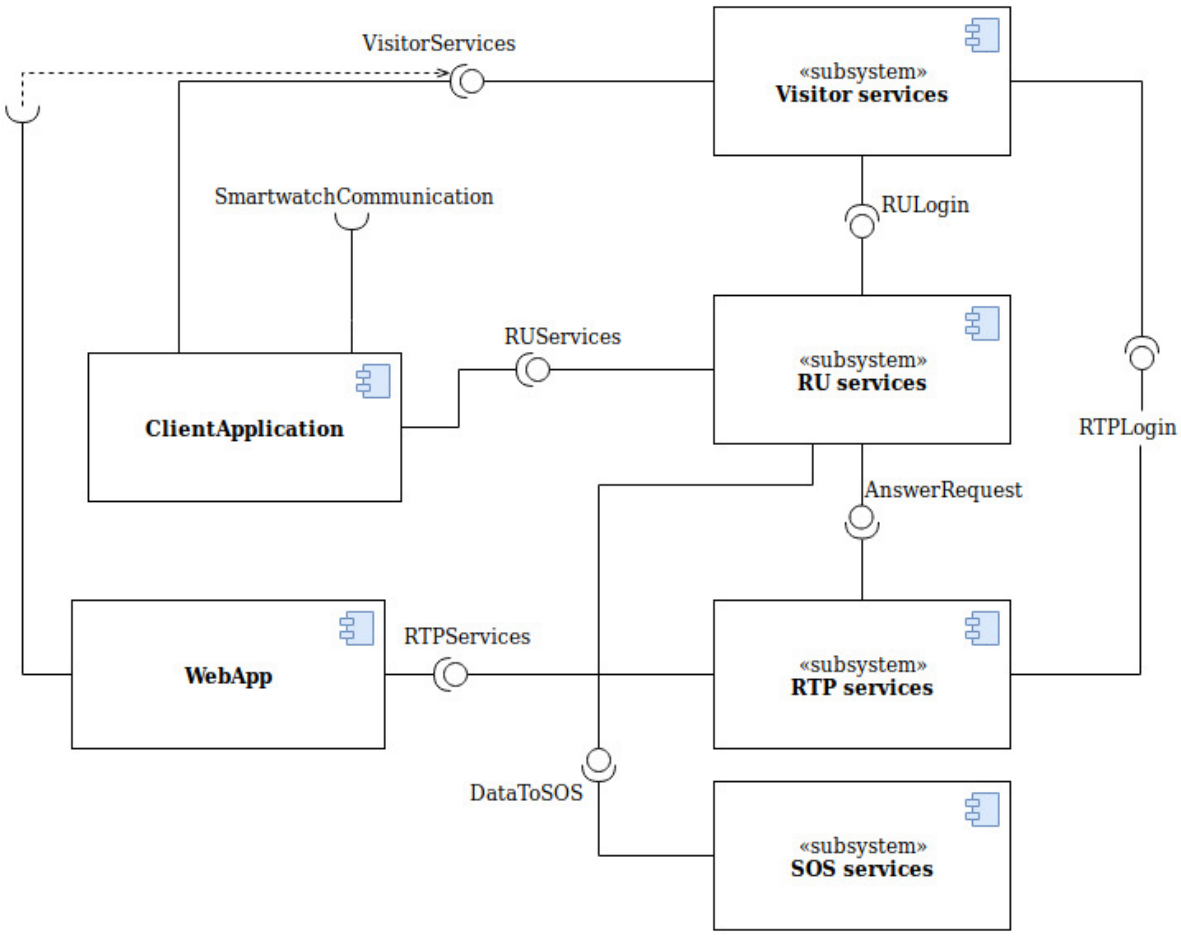
2.2 Component view

In this section a more detailed examination of the components previously mentioned is provided, with particular focus on the interactions occurring between them. In order to facilitate the understanding of the system and to maintain a certain level of generalization, every single component is uniquely identified by its interface.

In order to facilitate the understanding of the system architecture, the component view has been divided in subsystems, as better explained in the following paragraphs.

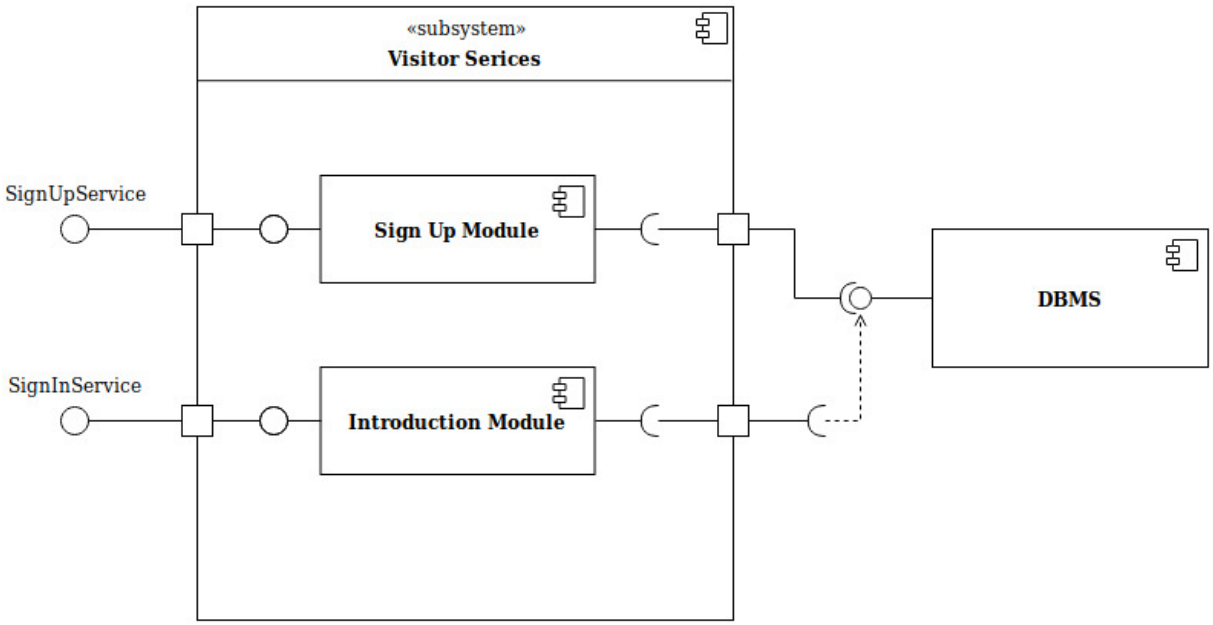
2.2.1 General view

General presentation of the component view: the system has already been divided into four subsystems. In this depiction no external resource is presented apart from the Mobile Application and the Web App, because a better description of the integration of the system with the DBMS will be provided in the following diagrams.



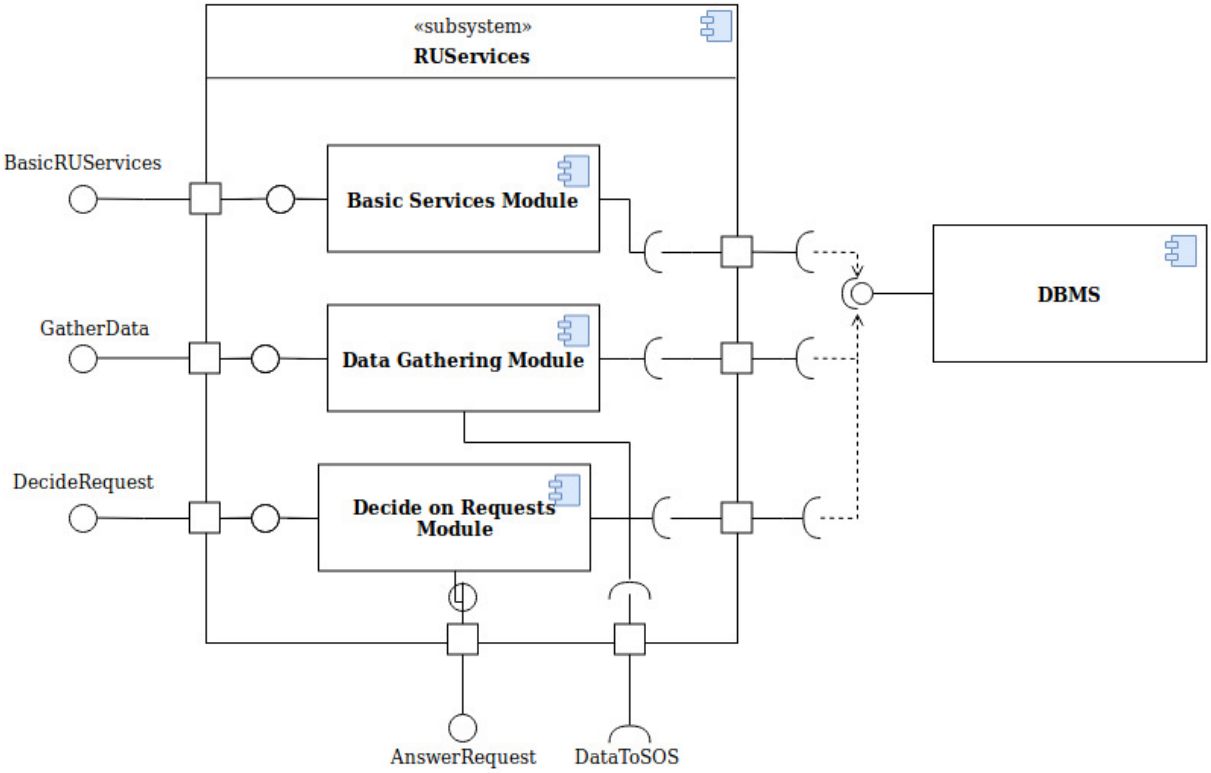
2.2.2 Visitor services modules

Depiction of the Visitor Services module, related to the first interaction with the platform: it concerns both the registration of the new RU or RTP and the log in of the old users. It interacts with the DBMS, in order to access the necessary data relative to the log in ad sign up operations; with the users themselves, via interfaces towards the smartphone application and the web app, and with the other components of the system that are related to the proper functionalities of the RUs and RTPs.



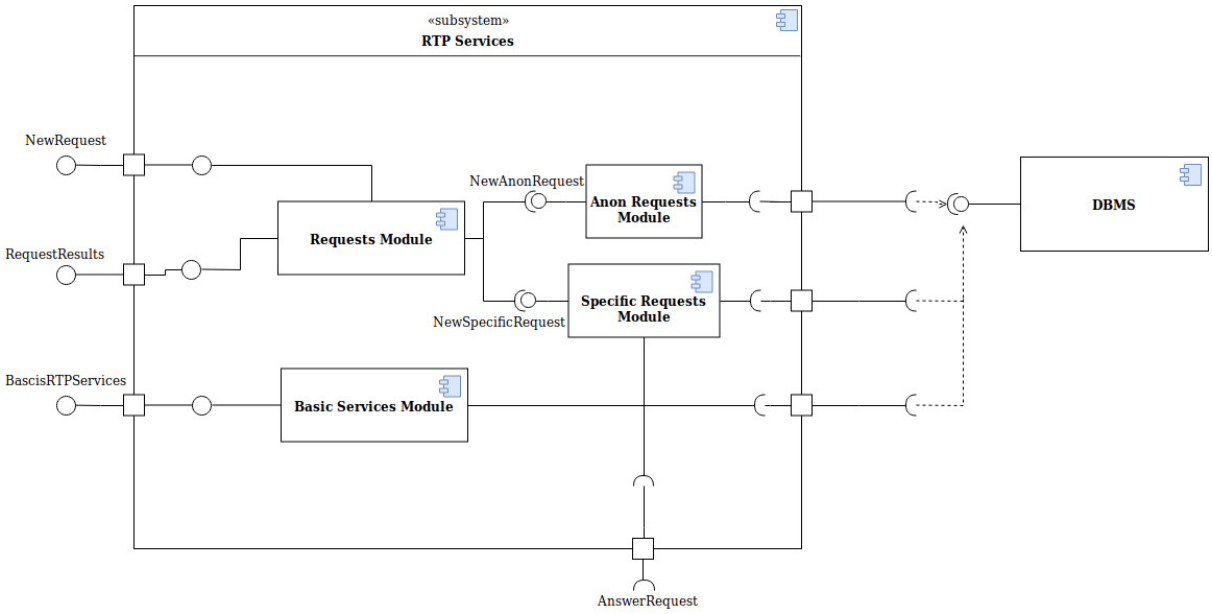
2.2.3 Registered user's services modules

Depiction of the Registered User's module, related to the functionalities offered to Registered Users.
In particular, it concerns the confirmation of the log in module, granting the status of RU; the possibility to upgrade as a Registered AutomatedSOS User; the gathering of the data itself, fundamental for the core functionalities of the platform; the possibility to decide whether to consent or not to the usage of the RU's private data by an external RTP.
It communicates with the DBMS, in order to both check the log in information and to store the user's data; with the smartphone application, in order to interact with the users him/herself; with the RTP services module, regarding the answer to the specific usage of the user's data; with the AutomatedSOS module, if the user is also a RAU.
In particular, the DecideOnRequest module also has to properly queue and eventually act on the requests accessing the database.



2.2.4 Registered third party's services modules

Depiction of the Registered Third Company's module, related to the functionalities offered to RTPs.
In particular, it concerns the the confirmation of the log in module, granting the status of RTP; the possibility to subscribe to new data; the possibility to file a new request, that can be both an anonymous or a specific one.
It communicates with the DBMS, in order to both check the log in information and to access the user's data; with the web app, in order to interact with the RTP itself; with the RU services module, regarding the answer to the specific usage of the user's data.

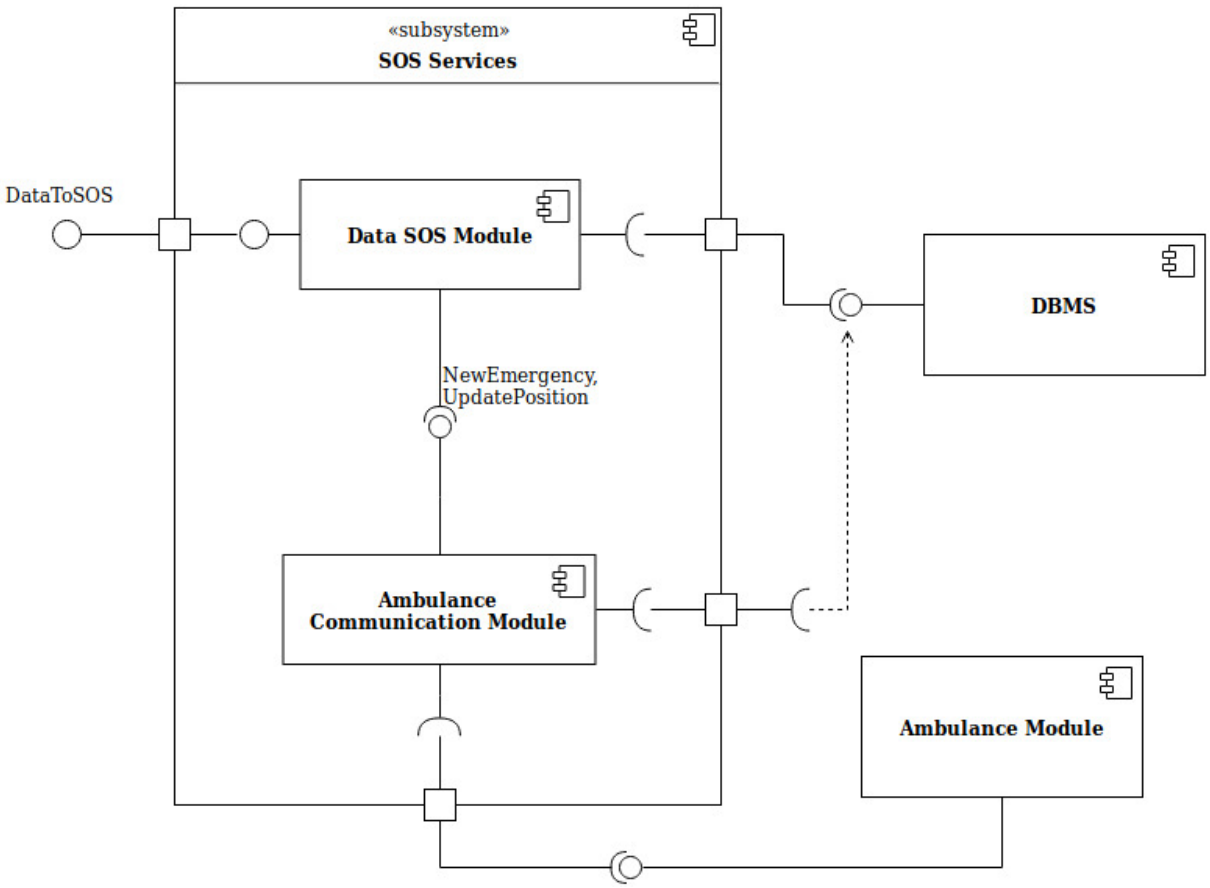


2.2.5 SOS services modules

Depiction of the SOS Services module, related to the functionalities offered by the AutomatedSOS service.

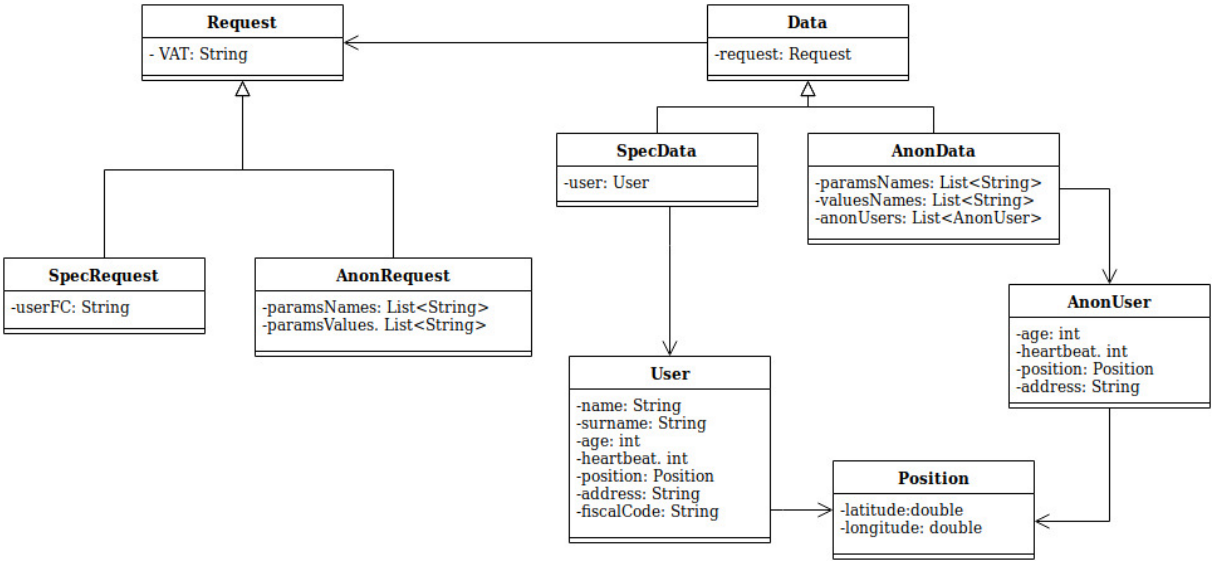
In particular, it concerns the the analysis of the Registered AutomatedSOS User's data, in order to check for abnormalities; the dispatching of an ambulance, if the said RAU is considered to be in danger.

It communicates with the DBMS, in order to acquire information regarding the RAU's thresholds and the contact of the ambulances; with the Ambulance Module, in order to dispatch an ambulances ad to give it the correct location of the RAU in danger. As previously specified, the Ambulance Module is considered to be external to the system and therefore it is not further analysed in this document: for the scope of this project the boundary of the system is the sending of the emergency message and the relative location of the RAU in danger.



2.2.6 Others

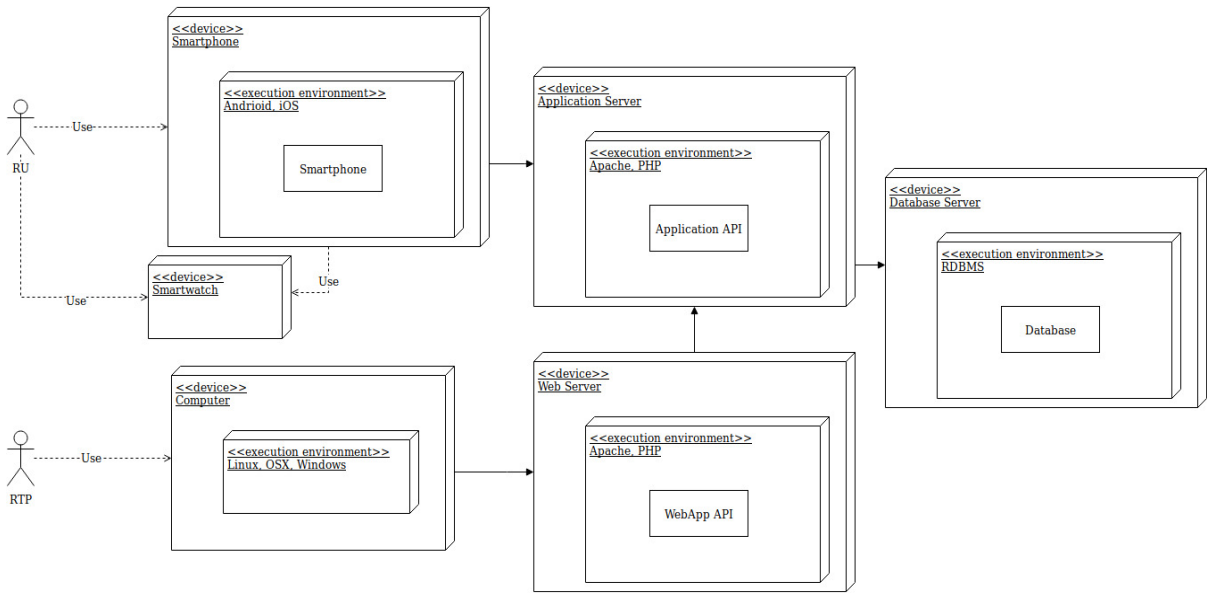
In this section some of the most important elements of the model of the system are represented. They are classes, often recalled in the description of the interfaces in this project.
In order to better present the whole system-to-be and to provide an higher level of clarification, it was decided to briefly present here some of these classes, that may be of help regarding the understanding of some of the mechanics presented in this document.
In particular, it should be noted the class Data, that generalizes SpecData and AnonData, that represent the data made accessible to the RTPs after a successful specific or anonymous request. They are accessed by the BasicRTPService Module, as described in this document.



2.3 Deployment view

In this section a detailed exposition of the deployment model is presented. In particular, the system requires the deployment of the software across the following nodes:

- **Smartphones.** The application utilised by the final RU will be deployed on smartphones. They will communicate to the Application server in order to grant the proper functionalities of the system.
- **Web Server.** The web app utilised by the final RTP will be deployed on a dedicated web server. It will communicate to the Application server in order to grant the proper functionalities of the system.
- **Application Server.** The main logic of the system will be deployed in a dedicated application server. This server will then communicate to the application deployed in the smartphones and to the Database Server that registers the needed information.
- **Database Server.** An additional server dedicated as a Database will be used to store all the information that need persistence, regarding both the RUs and the RTPs.



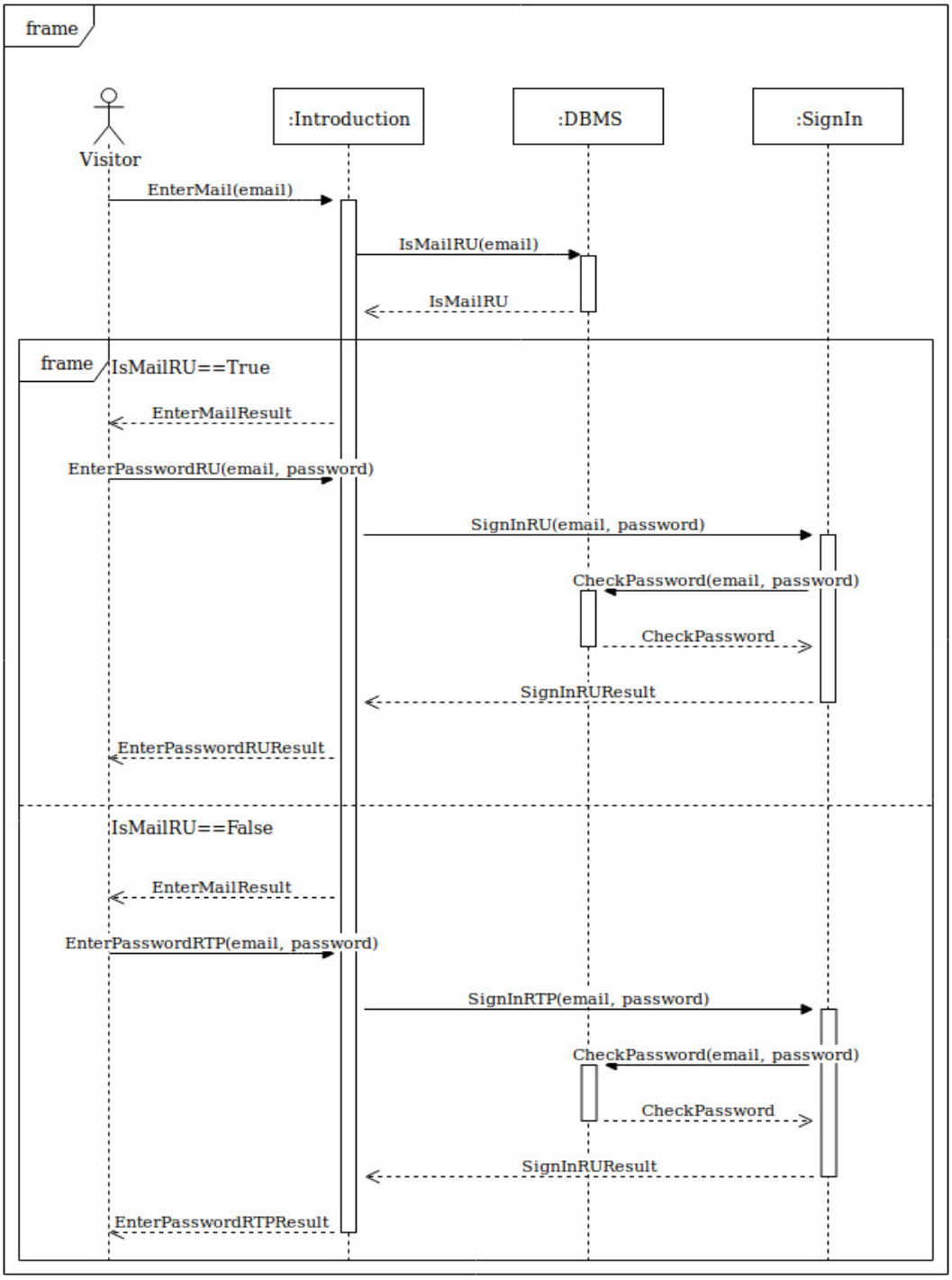
2.4 Runtime view

In this section a set of sequence diagrams are used to describe the interactions that happen at runtime between the main components of the system-to-be, as presented in the previous paragraphs. It should be noted that this representation is still an high level description of the system, and therefore some functions may be added and/or modified during the development of the platform.

In order to facilitate the understanding of the diagrams, some simplifications have been made, as explained in the brief paragraph related to each diagrams.
Important note: it should always be noted that any call to the DBMS has been properly named after the action that needs the said call: for example, in the IsMailRU(email) call, it is intended that the call to the DBMS has the purpose to deduct whether the given mail belongs to a Registered User or not, and that same call implicitly calculates already whether the mail satisfies the condition or not. Of course, that control is made by the caller and not by the callee, and that choice was made in order to make the presentation less heavier, without the repetition of very similar actions, without in any reducing the significance of the diagrams themselves in any mean.

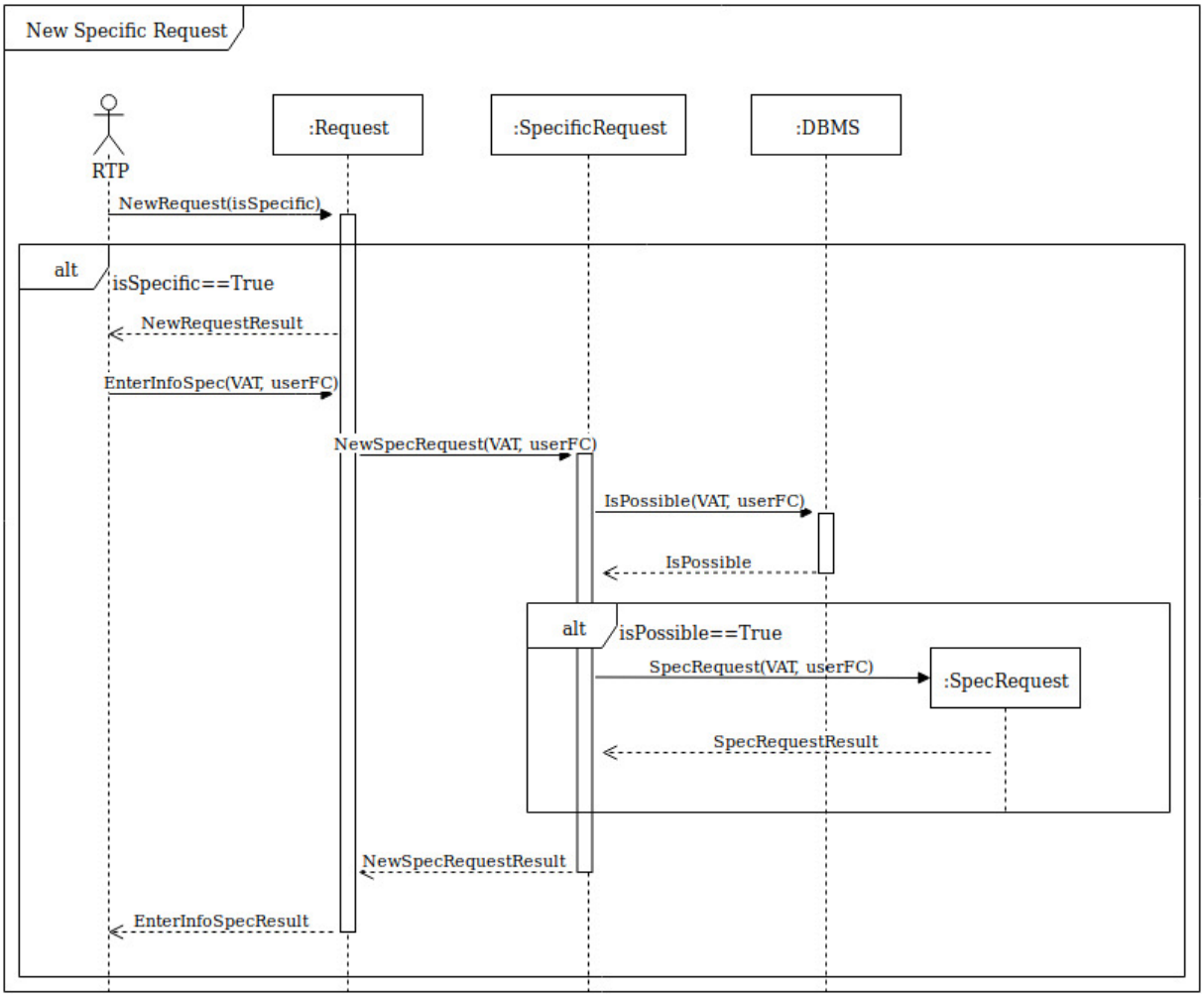
2.4.1 Login

This diagram represent the action of logging in by an already registered visitor, being it a RU or a RTP, exploring both cases. In order to make the diagram more understandable and avoid meaningless repetitions, the visitor is considered to represent also the platform it is using to access the system, namely the WebApp if it is an RTP and the Smartphone Application if he/she is a RU.



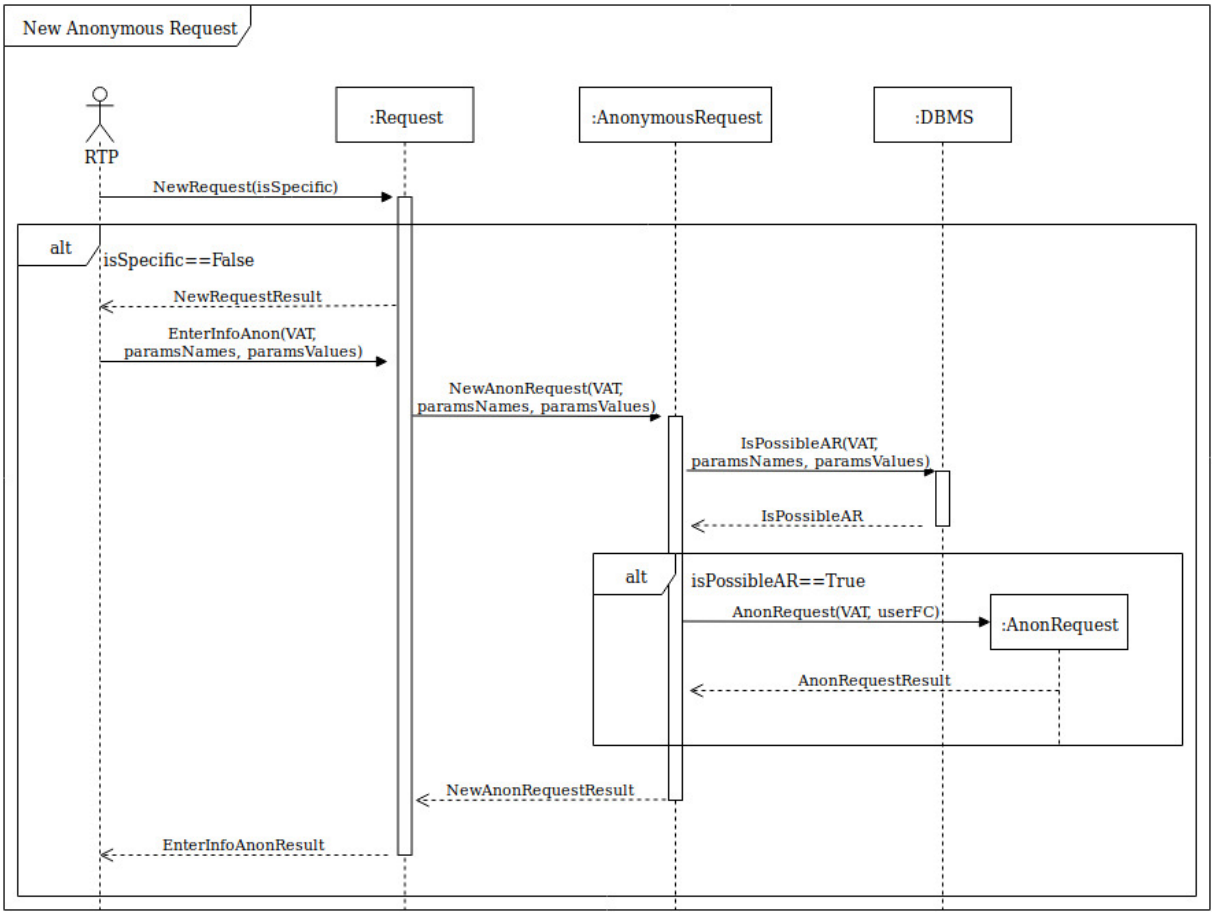
2.4.2 Specific request

This diagram represent the action of creating a new specific request by an already logged in RTP: therefore, in order to simplify the diagram itself, we consider the current RTP as corresponding to the WebApp it is logged in, to prevent confusion and facilitate the understanding of the diagram. Since the structure of the architecture foresees two cases of Request, namely Specific and Anonymous, and the selection of the type happens early on, this diagram and the following one in section 2.4.3 could be seen as the same one, divided in two in order to avoid an heavier presentation.



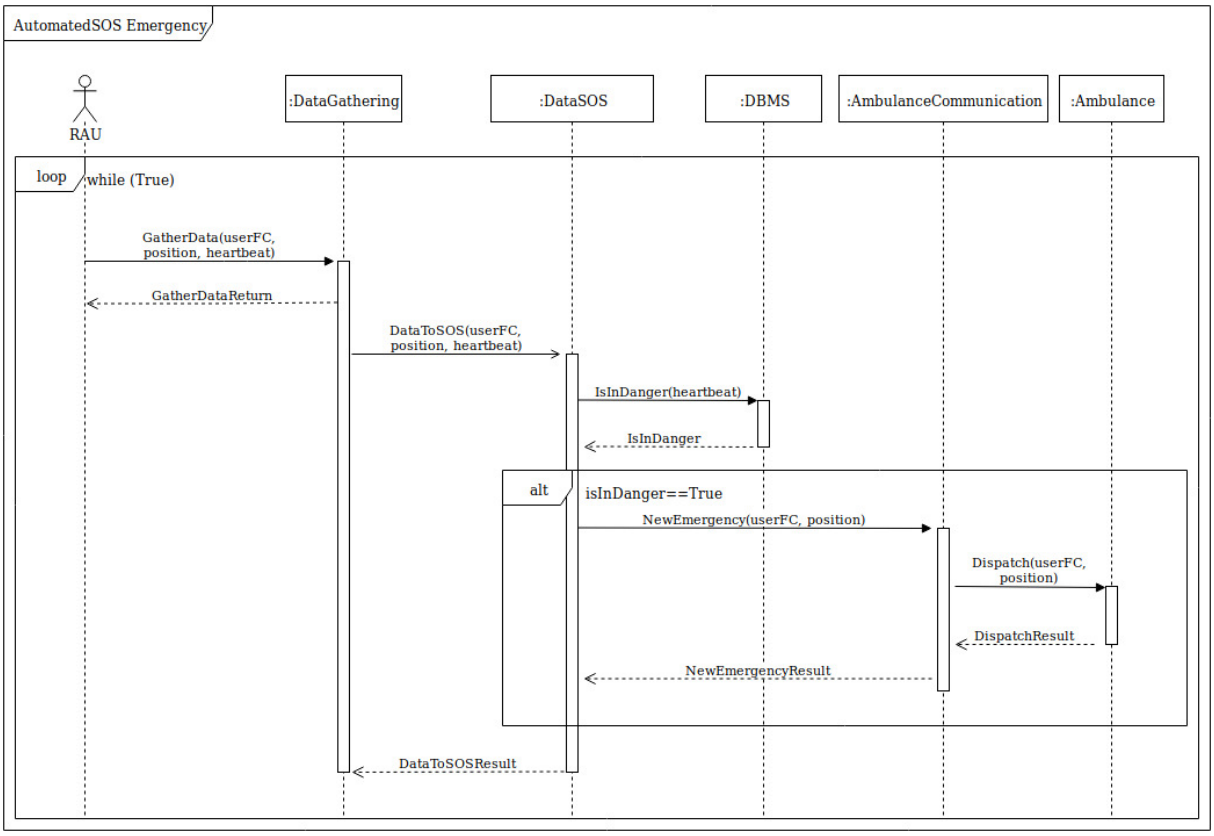
2.4.3 Anonymous request

This diagram represent the action of creating a new anonymous request by an already logged in RTP: therefore, in order to simplify the diagram itself, we consider the current RTP as corresponding to the WebApp it is logged in, to prevent confusion and facilitate the understanding of the diagram. Since the structure of the architecture foresees two cases of Request, namely Specific and Anonymous, and the selection of the type happens early on, this diagram and the previous one in section 2.4.2 could be seen as the same one, divided in two in order to avoid an heavier presentation.



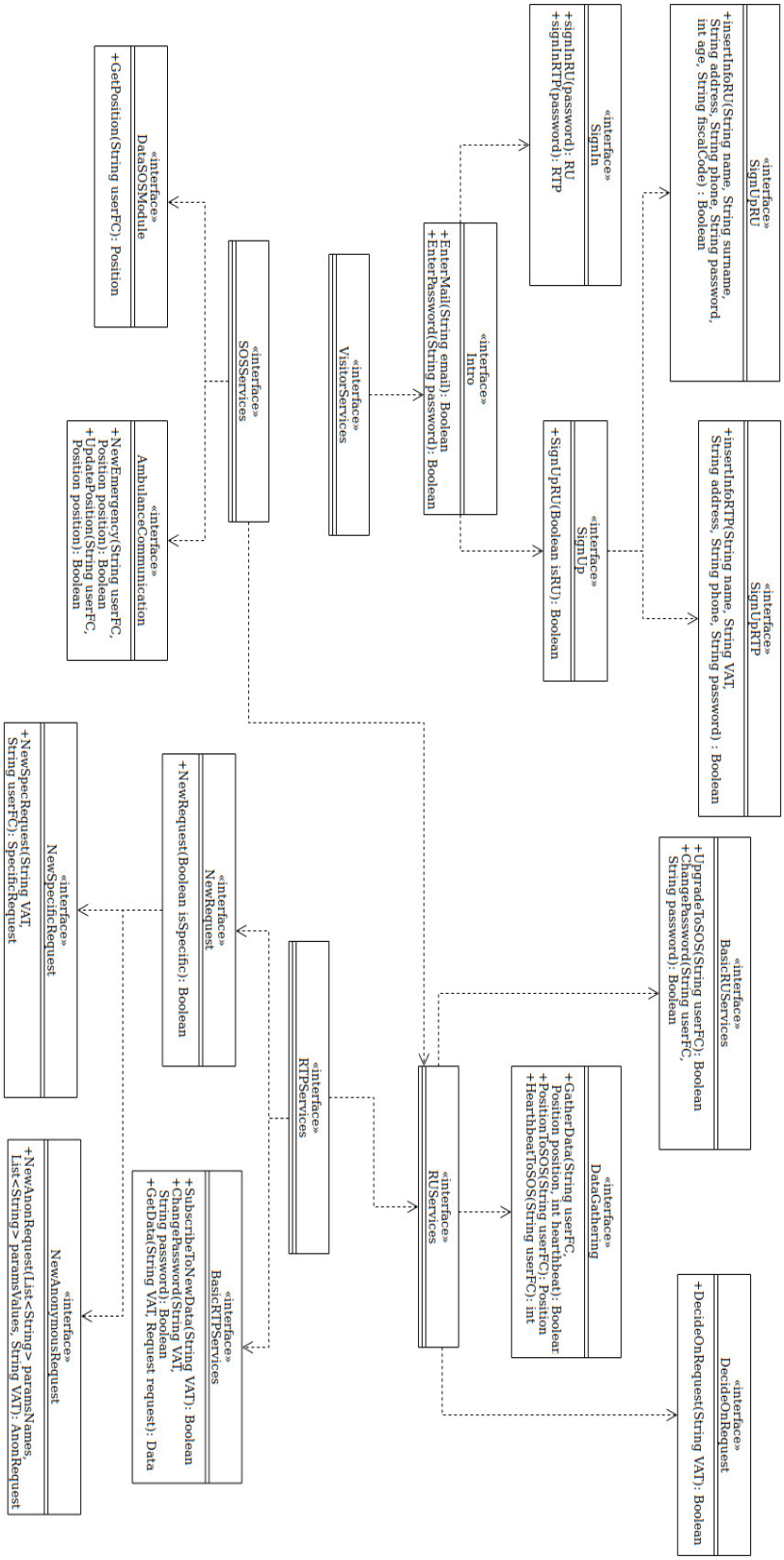
2.4.4 SOS emergency

This diagram represent the handling of an emergency situation by the system: the Registered AutomatedSOS User is considered to be logged in, and therefore, in order to simplify the diagram itself, we consider the current RAU as corresponding to the Smartphone Application it is logged in, to prevent confusion and facilitate the understanding of the diagram. It is notable that there is a everlasting loop, regarding both the transmission of data and its analysis: as long as the RAU is logged in, and then the conditions of this diagram holds, data will eventually be sent to the DataGathering module, and this data will be analysed by the AutomatedSOS system, in order to discover any emergencies. In this diagram, in order to facilitate the understanding, the only parameter considered to deduct the health status of the individual is the heartbeat, as explained in the related RASD. The thresholds used in the process are taken from the DBMS.



2.5 Component interfaces

In this section there is the description of the main methods belonging to the interfaces of the said components presented in the previous paragraphs.



2.6 Selected architectural styles and patterns

2.6.1 General view on architectural styles

As previously mentioned, the architecture chosen for the new TrackMe's system-to-be is a three tier architecture, composed by a presentation layer, an application layer and a data layer.
The presentation layer regards the communication with the final users, being them RU or RTP: it displays the GUI of the system and is directly accessed by the end users.
The application layer regards the main logic of the platform: it constitutes the most substantial part of the system and communicates directly with the other two layers, in order to control the application's functionalities.
The data layer is mainly dedicated to the management of persistent data, being that both administrative and organizational ones, such as passwords and other user-relater information, and the core RUs' data that represents the core of the system itself, such as the position and health parameters of the said RUs.

This choice is motivated by the necessity to adapt to a variety of end devices from the presentation layer's side, facilitated by a layered structure with clearly defined interfaces; moreover, a tiered structure also grants a great flexibility, maintainability and scalability.
In particular, the system's functionalities are strongly bound to the number of current users of the system itself, because of the great amount of data that a single user could produce or request: therefore, an architecture with great scalability is ideal especially in the first phases of the platform life cycle, making it possible to start with the management of a relatively small amount of users, such as the 20.000 Registered Users, 1.000 Registered Third Parties and 1.000 AutomatedSOS Users expected to be handled by the system in its first release. However, the number of users is expected to increase rapidly and therefore a scalable structure is an excellent choice.
In addition to that, a tiered structure can also offer possibilities regarding the duplication or, inn general, the management of multiple coexistent redundant modules, in order to offer increased availability and reliability. If these aspects are not of extreme importance in the Data4Help service, they are indeed absolutely necessary in the AutomatedSOS system, since the health of individuals directly depends on the system itself. For this purpose, ideally a system of redundant application serves could be implemented in order to manage the increased number of users, while maintaining excellent availability and reliability.

Moreover, all the advantages of a tiered architecture are of course offered, mostly regarding the dependencies between modules: an important change affecting a particular layer would only affect the layer itself, without the need of any intervention on the other tiers.

2.6.2 Design patterns

In the present project the many design and architectural patterns were adopted. The main ones are briefly presented in the following list.

1. MVC pattern

In order to effectively interact with the user and to adapt the system to the choices that he or she makes, the recommended architectural pattern is the Model-View-Controller. The software platform is divided in three interconnected parts, in order to avoid possible problems regarding coupling and erroneous interactions between the different parts of the system. In this case, the user interface and appearances are the View, the core functionalities and data managing are the Model, and the active response to the user's input is the Controller.

2. Facade pattern

This pattern substitutes multiple interfaces with a single one, within the context of a single system. It was used in the definitions of the main subsystem components, such as RUServices and RTPServices.

3. State pattern

This pattern allows the system to behave differently in relation to the state it is into. It is implicitly used in multiple occasions, for example to set that a certain user is logged in.

4. Proxy pattern

This pattern puts a middle entity between two other communicating elements, in order to allow it to act like a pass through entity or as a placeholder object. In this case, this pattern was used to better manage certain interactions, such as the one regarding new Requests.

Moreover, the following design patterns are recommended for the implementation f the system:

1. Observer pattern

An essential part in the effective adoption of an MVC pattern, the observer permits that some objects are notified when the state of some other observed objects changes.

2. Strategy pattern

This patten defines a set of multiple algorithms, in order for the system to choose the best one when a certain behaviour is needed. Even if is not essential, this pattern is recommended in order to grand an higher adaptability of the system to the different situations that may happen.

3. Factory pattern

This pattern exposes a method that can create other objects, in order to allow another class to control the creation of said objects. Because of its flexibility, it is very common in MVC contexts.

2.7 Other design decisions

While the Data4Help service could be developed and managed without the need of any particular external service (obviously excluding the DBMS and related services), AutomatedSOS, because of its own nature, needs an integration with an ambulance dispatching system.

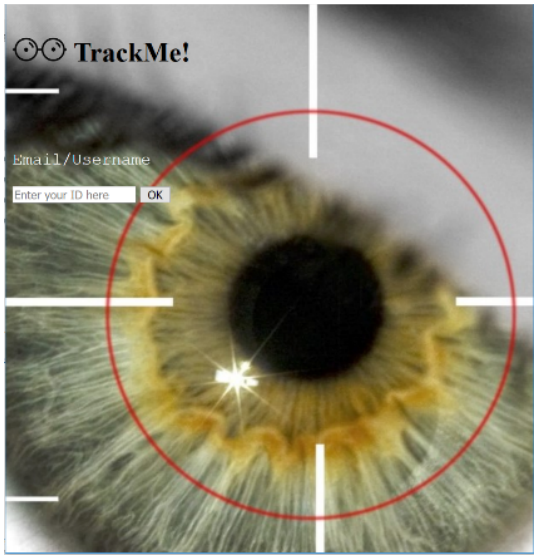
However, the description of the previously mentioned ambulance system is outside of the scope of this document: it is assumed that an external service is provided, and that the said service offers an interface towards which the AutomatedSOS system can direct its dispatching instructions. The boundary of the system in then set on the sending of the emergency alert message from the Ambulance Communication Module and it is assumed, as also exposed in the related RASD, that the external ambulance dispatching service will promptly react to the message, respecting all the non-functional requirements already presented in the RASD itself, with particular attention to the 5 seconds time boundary on the response time.

Another important design decision regards the limited resource that most smartphones have at their disposition: in order to grant an efficient system, the design of the ClientApplication should take into account this constraints, especially considering the low battery life of these devices.

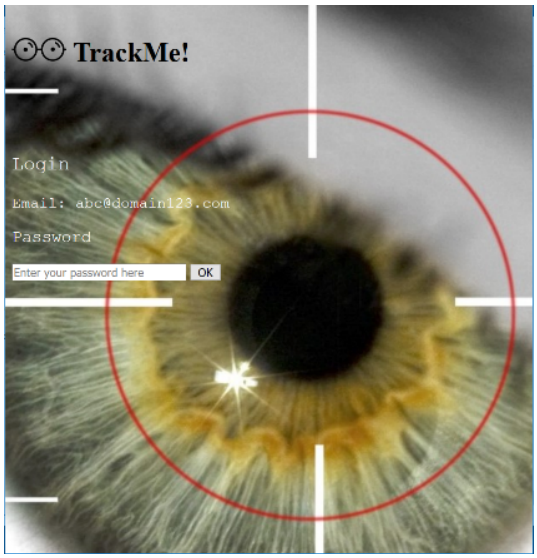
As a brief last consideration, the ClientApplication should obviously able to effectively communicate with smartwatches and similar devices, as exposed multiple times in this document and in the related RASD: the design process should then take into account this important factor, in order to offer a better service in general.

3. User interface design

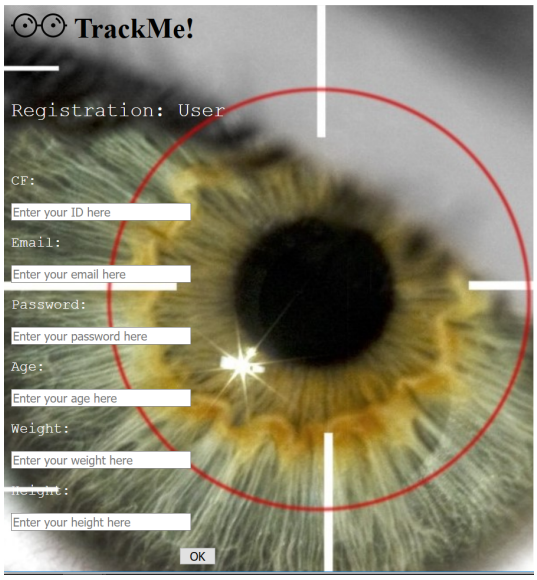
This is how the app will look to the users and to the third party. Here it has been used HTML, CSS, and Javascript. Although they are not directly supported by a native Android Application, there exists many “framework” which can translate them into native code, supporting the functionality of the device’s keys.



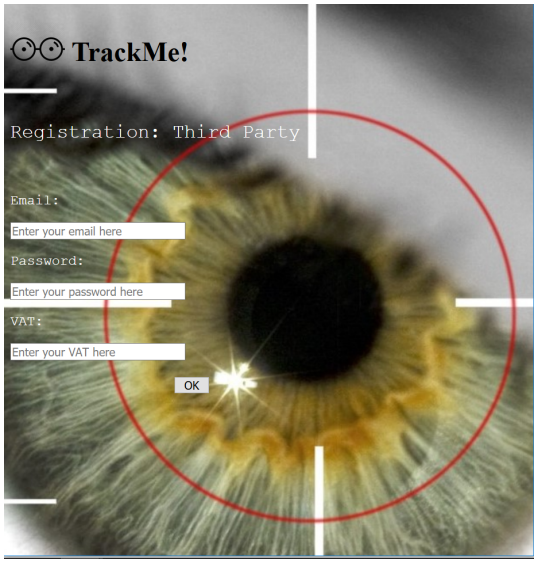
This allows the user/third party to be identified and to be redirected appropriately, whether it is a new user/third party or a registered user/third party.



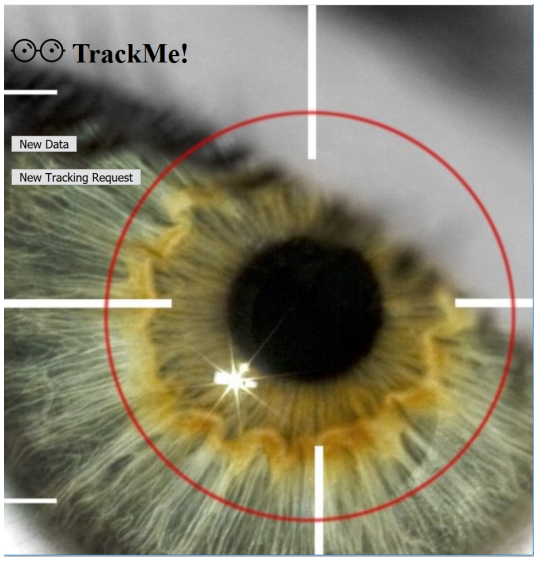
This is the following screen, presented if you are already registered



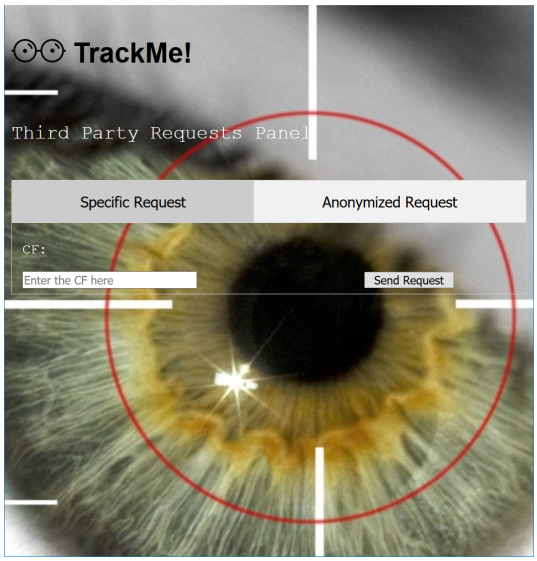
This is the registration page, in case you are an user



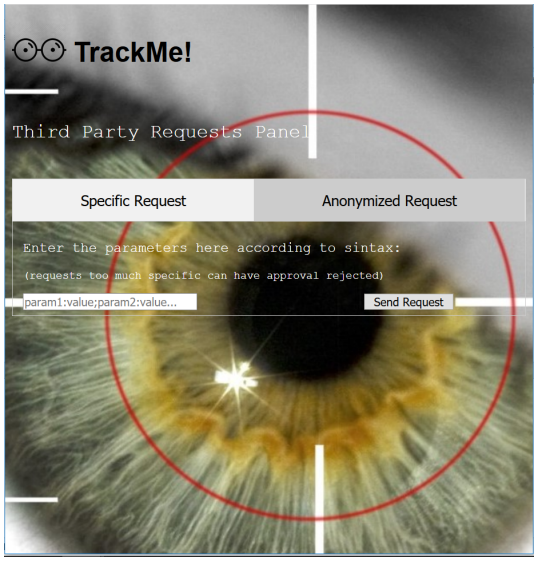
This is instead the registration page when it's the case of a business



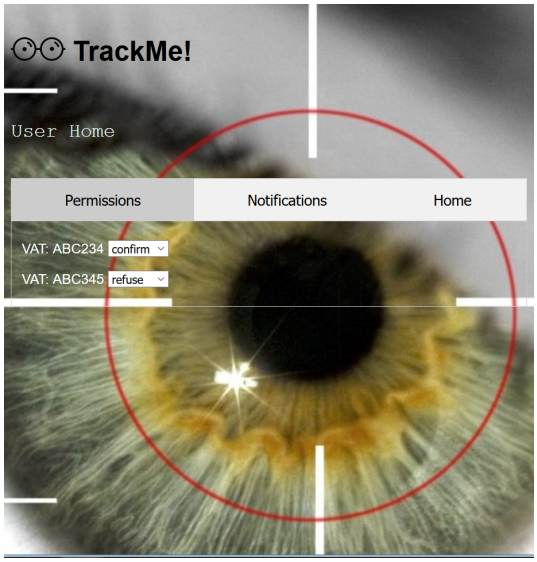
Here the third party will decide what to do, subscribe to new data, or pose a new request, which lead to the next image…



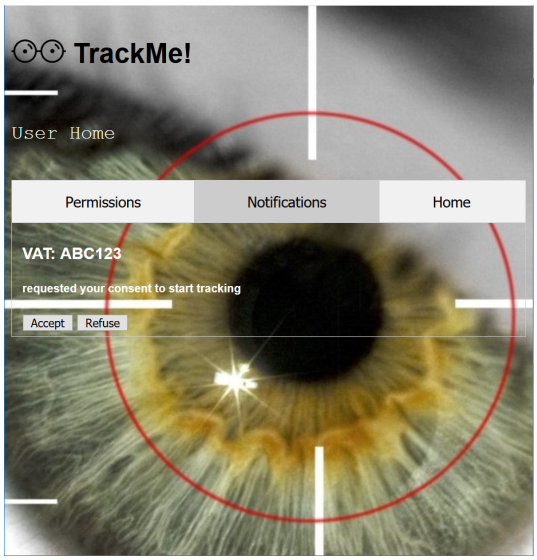
A specific request will need only the CF of the interested user.



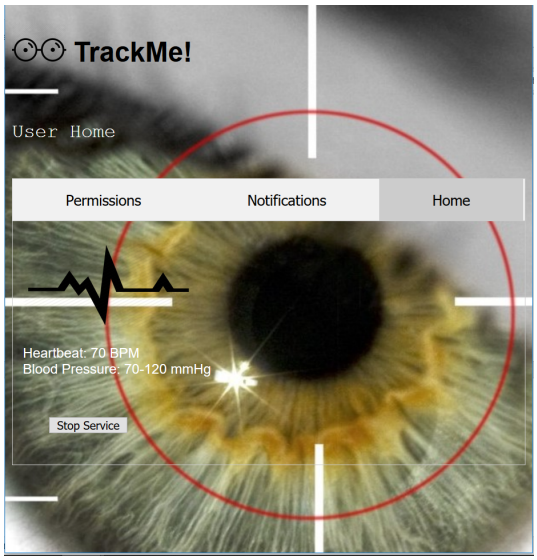
A generalized request will need instead a list of pair: parameter-value, to retrieve the users related to the parameters



In the permission tab, user side, the user will set each business the right permissions



Notifications: real time messages sent from the server



In the home tab, user will see in real-time its vital parameters along with the possibility to stop the service at any time

4. Requirements traceability

In this section there is the description of how the architecture presented in this document is able to meet all the goals and requirements specified in the relative Requirements and Specification Document. In particular, every component is hereby related to the goals and requirements it aims to fulfil.

4.1 [G1]

Allow an individual to become a Registered User, by agreeing to the TrackMe privacy policy and by providing credentials.
Requirements *[R1] – [R2]*

- Introduction
- SignUp

4.2 [G2]

Allow a third-party company to become a Registered Third Party by providing credentials.
Requirements *[R2] – [R3]*

- Introduction
- SignUp

4.3 [G3]

Allow a Registered Third Party to have access to a specific, properly identified Registered User's data if and only if the said User gives its consent to that action.
Requirements *[R4] – [R8]*

- DataGathering
- DecideOnRequest
- Request
- SpecificRequest
- BasicRTPServices

4.4 [G4]

Allow a Registered Third Party to access anonymized data of groups of individuals, if and only if the said group comprehends at least 1000 individuals.
Requirements *[R9] – [R10]*

- DataGathering
- Request
- AnonymousRequest
- BasicRTPServices

4.5 [G5]

Allow a Registered Third Party to subscribe to new data, in order to receive the latest data as soon it is produced.
Requirements *[R11] – [R12]*

- BasicRTPServices

4.6 [G6]

Allow a Registered User to become a Registered AutomatedSOS User.
Requirements *[R13] – [R14]*

- BasicRUServices

4.7 [G7]

Send an ambulance to a Registered AutomatedSOS User's location, if and only if his/hers health parameters drop below a certain threshold.
Requirements *[R15] – [R17]*

- DataGathering
- DataSOS
- AmbulanceCommunication

5. Implementation, integration and testing

5.1 Implementation plan

The modularity of the structure of the application facilitates the implementation of the whole system itself, with the gradual implementation of the different modules.
The order of the implementation of said modules may depend a lot of different factors and unpredictable variables that could potentially change during the implementation of the modules themselves, such as the discover of defects in the proposed design: therefore, the order reported in the following paragraph may be affected by changes during the implementation of the modules.
In particular, it is here suggested to implement first the Data4Help service, and later the AutomatedSOS one, because the first is completely self-sufficient, while the second heavily depends on the former service.

5.1.1 Data4Help implementation plan

The proposed order of implementation of the different modules regarding the Data4Help service is the following:

1. Model
Since the various parts of the model are needed in almost every other component of the system, it is important that the model is completed first, in order to facilitate the implementation of the other components depending on it.

2. ClientApplication and DataGathering
As the components that constitute the core functionality of the service, the reported modules should be implemented as soon as their dependencies in the model are completed. DataGathering is of primary importance, since it is the gate through which data flows in the system, and therefore an early implementation is useful and can prevent later delays whether this same implementation happens to require more time than expected. ClientApplication, on the other hand, represents the other crucial module in the core functionality and, since it has a great synergy with the previous component, should be implemented very early, even if with a slightly lower priority.

3. Request, AnonymousRequest, SpecificRequest

In order of priority and according to the integration of the different functionalities, all the features regarding the requests should be implemented right after the already mentioned components. In particular, since the Request module heavily depends on the AnonymousRequest and SpecificRequest and there is an inherent similarity between the latter two, their implementation should happen at the same time, in order to correctly maintain the same coherence in the components.

4. DecideRequest, BasicRTPServices, WebApp

Since the SpecificRequest module naturally recalls the DecideRequest, in order to grant or deny to the asking RTP the access to the user's data, the latter module should be implemented right after the previous one, in order to complete the relative functionality. The same reasoning applies to BasicRTPServices, that allow the RTP to access the correct data, according to its Requests: it should be implemented here, in order to complete this side of the service.

Then, in a similar way as it was discussed in the ClientApplication module, the WebApp should be developed in order to complete the RTP's end part of the system, even if with a lower priority.

5. Introduction, Sign Up, BasicRUServices

The last components, in order of importance, are the one regarding the login, sign in, and the side functionalities offered by the system, such as the possibility to modify the password. Being them also relatively easy to implement, it was decided to keep them for last.

5.1.2 AutomatedSOS implementation plan

Regarding the implementation of the AutomatedSOS service, there are only two main modules that need to be implemented, namely **DataSOS** and **AmbulanceCommunication**. Since these modules are only two and are greatly interconnected, it is suggested to implement them at the same time, giving slightly more importance to the DataSOS module, since the latter component heavily relies on it.

5.2 Integration and testing

The integration of the different components of the system is one of the highest priorities during the development of the platform, therefore it should begin as early as possible.
However, in order to facilitate the integration and to make sure that there are no flaws in the system, the modules should be promptly tested, both alone and in their composition. Since the two processes of integration and testing are inherently connected, they are both considered in this same section. Even if two separated paragraphs are dedicated too integration and testing strategy it should be remembered that the two processes should proceed hand in hand, and that said division in only for clarity's sake.

5.2.1 Integration strategy

The way the different modules should be integrated has already been shown in the system. In this paragraph no integration with the model is considered, since the model is strictly related to the implementation of the modules themselves, and then the integration between components and model comes naturally with the implementation of the single modules.
The process of integration can be divided in the following groups:

5.2.1.1 Integration with the DMBS.

Most of the components belonging to the Application layer need an integration with the database.
For the Data4Help system the complete list is the following:

- Introduction
- SignUp
- BasicRUServices
- DataGathering
- DecideOnRequest
- AnonymousRequest
- SpecificRequest
- BasicRTPServices

While for the AutomatedSOS service the following modules need an integration with the DBMS:

- DataSOS
- AmbulanceCommunication

5.2.1.2 Integration between components of the system in the Application Server.

Of course, some components of the system need to be integrate one another.
For the Data4Help service these components are:

- SignUp and Introduction
- Request and SpecificRequest
- Request and AnonymousRequest
- SpecificRequest and DecideOnRequest

With the inclusion of the AutomatedSOS service, the following components need integration one another:

- DataSOS and DataGathering
- AmbulanceCommunication and DataSOS

5.2.1.3 Integration with external services.

For the Data4Help the components that need an integration with some external services are:

- ClientApplication and the smartwatch device, as described in the architecture of the system

Regarding the AutomatedSOS service, the component needing an integration with external services is:

- AmbulanceCommunication with the Ambulance manager itself, considered as an external service.

5.2.1.4 Integration between Application Server and user-related devices

For the Data4Help the components that need an integration with some external services are:

- Request with the WebApp
- BasicRTPServices with the WebApp
- Intro with the WebApp
- Intro with the ClientApplication

- DecideOnRequest with the ClientApplication
- DataGathering with the ClientApplication
- BasicRUServices with the ClientApplication

There are not AutomatedSOS related modules that need an integration with the ClientApplication.

5.2.2 Testing strategy

This test plan describes the testing approach that will follow the testing of the TrackMe's new system. In particular it will be here considered the test strategy, regarding the rules the tests will rely on.

The objective of the test is to verify that the functionality of TrackMe's new system, according to the specifications. The test will execute and verify the test cases, with the outcome of each one notified to the development team right away.

The final product of the test is twofold:
- A production-ready software;
- A set of test results to provide quality assurance of the product.

5.2.2.1 Testing assumptions

The following assumptions are considered to be always valid:

- Exploratory Testing would be carried out on each module as soon as they are built correctly.
- All the defects would come along with a detailed format specified in the paragraph related to Test Execution.
- There will be cooperation between the Development Team and the Test Team, during the execution of the test cases.
- Performance testing is not considered for this estimation.
- The testing environment will be always online. (Downtime of the server will allegedly create problems like: Is the functionality not available because of errors in the integration, or the testing environment missed something itself?)
- The system will be treated as a black box; if the information shows correctly online and in the reports, it will be assumed that the database is working properly.

5.2.2.2 Testing principles

- Testing environment and data will emulate a production environment as much as possible.
- There will be entrance and exit criteria, namely Pre-Conditions and Post-Conditions.
- The entry criteria refer to the desirable conditions in order to start test execution.
- The exit criteria are the desirable conditions that need to be met in order proceed with the implementation.

- Entry and exit criteria are flexible benchmarks. If they are not met, the test team will assess the risk, identify mitigation actions and provide a recommendation. All this is input to the project manager for a final “go-no go” decision.
- Test environment will be with application configured and ready to use
- Testing will be focused on meeting the security objectives and quality of service.
- Final testing will be performed by a top-down approach

5.2.2.3 Data approach

In functional testing, TrackMe's system will use pre-loaded test data, which is used for testing activities.

5.2.2.4 Additional details

Regarding exploratory tests, their purpose is to make sure that critical defects are removed before the next levels of testing are reached.

Regarding functional tests, the following key functionalities should be tested:

- One by one request of data to one single user by one single third party
- Multiple requests of data to one single user by one single third party
- Multiple requests of data to one single user by many third party
- Multiple requests of data to many user by many third party
- Multiple tracking of many users concurrent to many requests to many users by many third party.
- All the functionality running simultaneously.
- Requests to user violating some of the privacy policy of the system
- Each time parameters drop below the threshold the alarm is sent to the server.

6. Appendix

6.1 Used tools

In the redaction of this document the following software tools were utilised:

- OpenOffice Writer
- Microsoft Word
- Microsoft OneDrive
- https://github.com
- https://www.draw.io

6.2 Hours of work

In this section are reported the hours of work spent by the single group members in order to redact this document, with a brief description of the activities done during each session.

6.2.1 Alberto Tiraboschi

DATE	***TASK***	***HOURS***
26/11	Global idea Organization of work	4
27/11	Algorithms paragraph start Interfaces start	3
02/12	Algorithms paragraph end Paragraph 6 start	3
04/12	Test paragraph completed with additional material Interfaces completed	5
09/12	Last overall check Trimmed excessive deep parts Thorough revision	10

6.2.2 Giorgio Polla

DATE	***TASK***	***HOURS***
17/11	Introduction	1
18/11	Introduction Architectural overview	2
24/11	Component view	3
28/11	Runtime view	4
01/12	Component view Component interfaces	3
02/12	Component view Deployment view	3
05/12	Architectural styles, patterns Requirements traceability	4
07/12	Architectural styles, pattens Implementation plan	3
08/12	Implementation plan Integration plan	4
09/12	Merge of the work done Integration plan	5
10/12	Final review Testing DD layout	6

7. References

- Specification document: “Mandatory Project Assignment AY 2018-2019”
- Related RASD: “Requirements Analysis and Specification Document”, version 1
- Slides from the Software Engineering II course, A.Y. 2018-2019, PoliMi
- UML diagrams: uml-diagrams.org