POLITECNICO
MILANO 1863

SOFTWARE ENGINEERING 2 PROJECT

TRACKME

Design Document

Authors:
Matteo VELATI
Niccolò ZANGRANDO

Professor:
Matteo ROSSI

Delivery date:
10/12/2018

Table of Contents

1 Introduction

1.1 Purpose of this document

In this document we are going to describe software design and architecture of the TrackMe system.
The software architecture of a system is a group of structures of the system, which comprise software elements, the externally visible properties of the elements, and the relationships among them.

1.2 Scope

The main scope of TrackMe application is to allow third parties to monitor the health and positions data of individuals.

TrackMe offers two services: Data4Help and AutomatedSOS.
The former can be accessed by every successfully registered user, who can share its data or send a request to collect some data. The request must be approved by the individual user, in case of single request, or by the system itself, in case of group request, because it's needed to guarantee the anonymity of the data.
The latter is offered only to individual users: the system constantly monitors the vital parameters and, when it detects an anomaly on the fixed thresholds, it sends an help request in less than 5 seconds and notifies the user of the incoming ambulance at his current position.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Company user: a third party registered user

Individual user: an individual registered user

Single request: a request made by a Company user to a specific Individual user

Group request: a request made by a Company user for a group of people

Health data: the vital parameters of an individual

Location data: the coordinates of the position of an individual

Warning: a message sent by the AutomatedSOS service to an Individual user to advise him that an ambulance is arriving

System: the TrackMe software we are to develop

1.3.2 Acronyms

RASD: Requirements Analysis and Specifications Document

API: Application Programming Interface

GPS: Global Position System

GUI: Graphical User Interface

DB: DataBase

DBMS: DataBase Management System

MVC: Model View Controller

RASD: Requirements Analysis and Specification Document

DD: Design Document

1.3.3 Abbreviations

[Gn]: n-th goal

[Rn]: n-th requirement

1.4 Revision History

- **v1.0** December 10th, 2018

1.5 Document Structure

According to IEEE standard, this document is structured in six chapters:

1. *Introduction:* provides an overview of this entire document, explaining the utility of the project.

2. *Architectural design:* describes different views of components and their interactions.

3. *User interface design:* provides an overview on how the user interfaces of our system will look like.

4. *Requirements traceability:* explains how the requirements we have defined in the RASD map to the design elements that we have defined in this document.

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

6. *Effort spent:* reports a summary of the effort spent for each section by each team member.

7. *References:* includes the references of this document.

2 Architectural Design

2.1 Overview

The need to design a system which allows communications with agents such as users and external systems led us to decide to use a client-server architectural approach.
Specially, our choice falls on a four-tier architecture whose layers are: Presentation, Logic and Data.
On the client side is generated a dynamic GUI through a module that interacts with the Application Server through the Internet network.

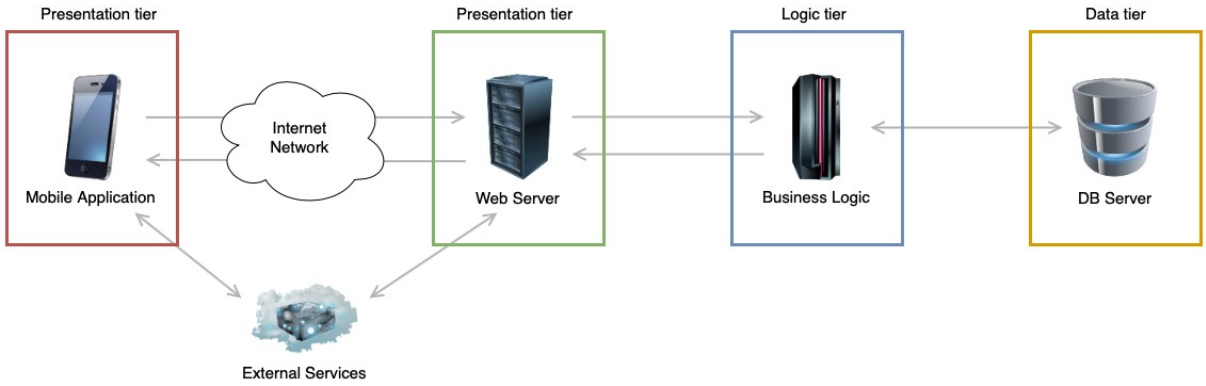

Figure 1: 4-tier architecture with internet layer

To clarify at a finer level how the Server components are mapped in the four tier layered architecture is now explained the division introduced in the diagram above:

- Mobile Application: components used by the users in order to access the functionalities offered by the system
- Web Server: components which provides interfaces to clients in order to allow them to use functionalities offered by the system
- Business Logic: components which realizes the functionalities offered by the system
- DB server: components which store and manage the access to the data produced and needed by the Business Logic

The high level components architecture is divided into five elements, which interact with each other through the main element: the Central. The central receives and manages all kind of requests made from the CompanyUser and the treatment of the data from the IndividualUser.

A communication can start from the mobile application of a CompanyUser: after sending a request, the Central checks the type of the request, then it forward the request to the IndividualUser, (in case of single request) who can accept or refuse. This kind of communication is made in a synchronous way since the IndividualUser can receive many requests at once.

In case of a group request, the Central communicates with another kind of component: the Data, in which are stored the information about all the people who have joined TrackMe services, that can be extracted at any time.

Finally, the Central is in contact with the AutomatedSOS, in order to send a request for help to a customer whose vital parameters are below the pre-fixed thresholds. The AutomatedSOS sends a warning on the mobile application of the IndividualUser.

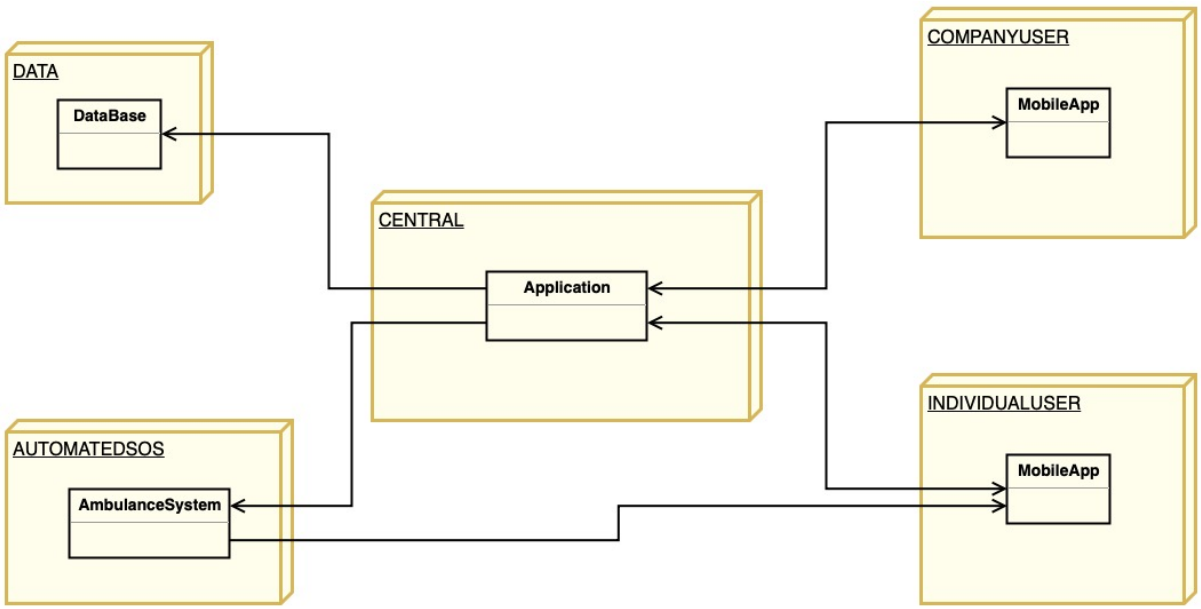

Figure 2: High Level components architecture

2.2 Component View

The following diagram shows the main component of the system and the interfaces through which they interact.

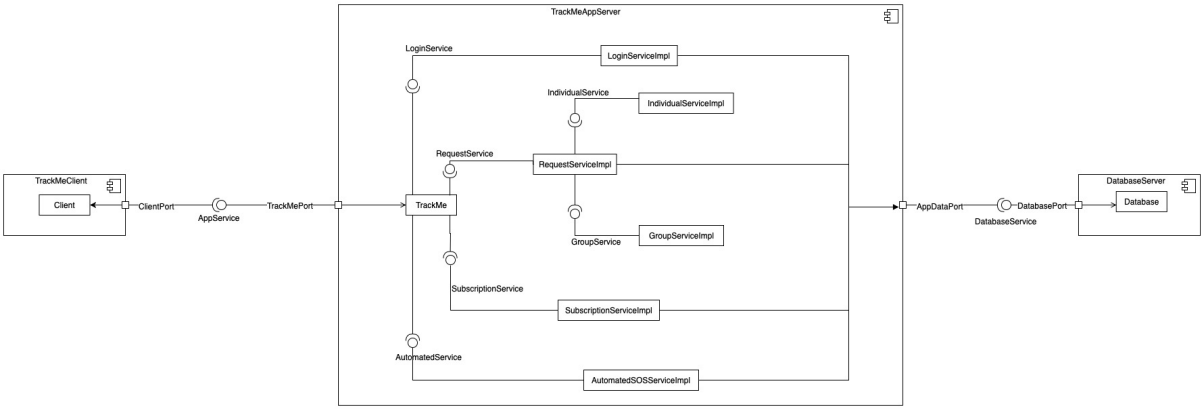

Figure 3: Component view

The application server consists of the following parts:

- LoginService: responsible for the authentication of the user
- RequestService: manages requests queue
- IndividualService: manages the individual requests
- GroupService: manages the group requests
- SubscriptionService: manages the subscription for new data
- AutomatedSOSService: handles the help requests
- AppService: responsible for the connection of user's device
- DatabaseService: responsible for the acquisition of data from the database

The following class diagrams illustrates in more detailed way the components of the application server and how they interact with each other through the interfaces.

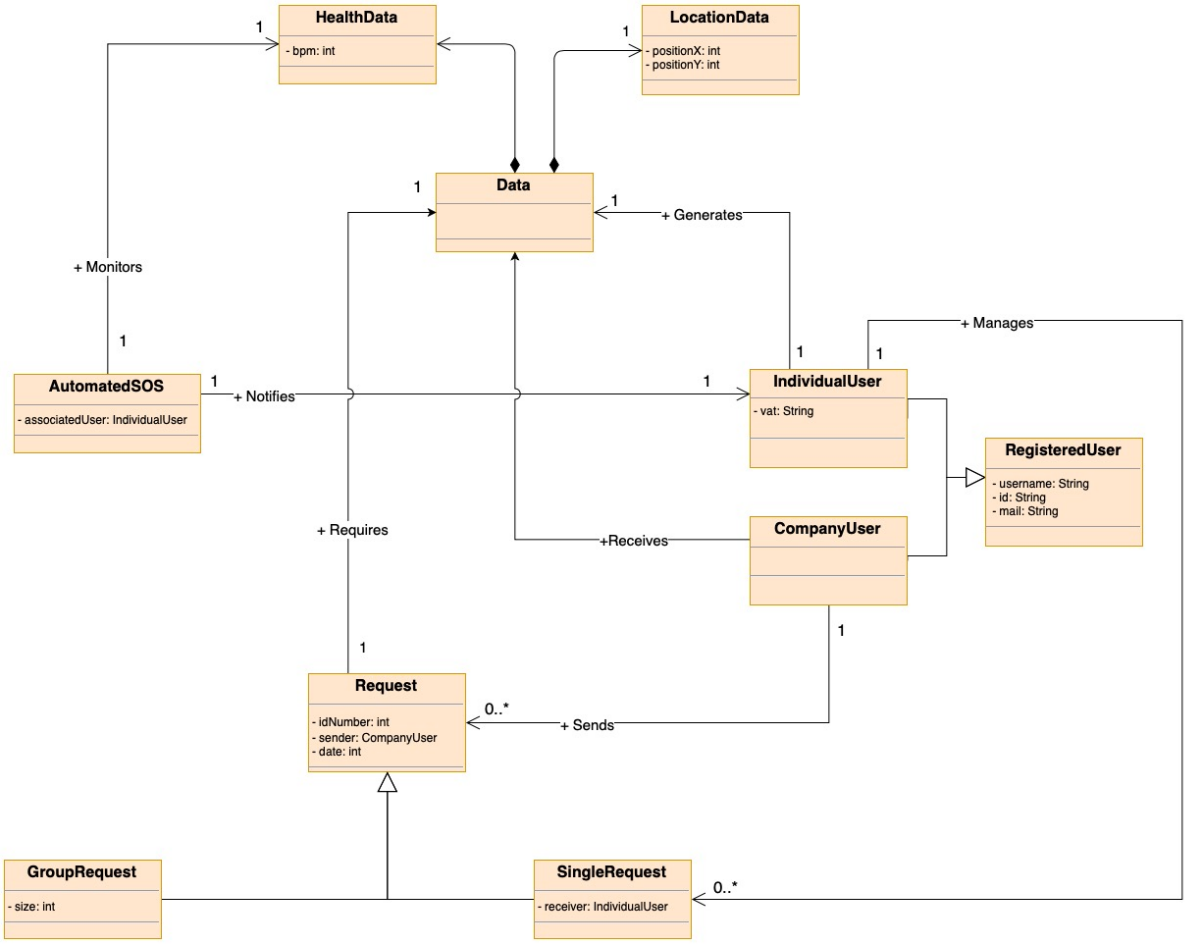

Figure 4: Class diagram

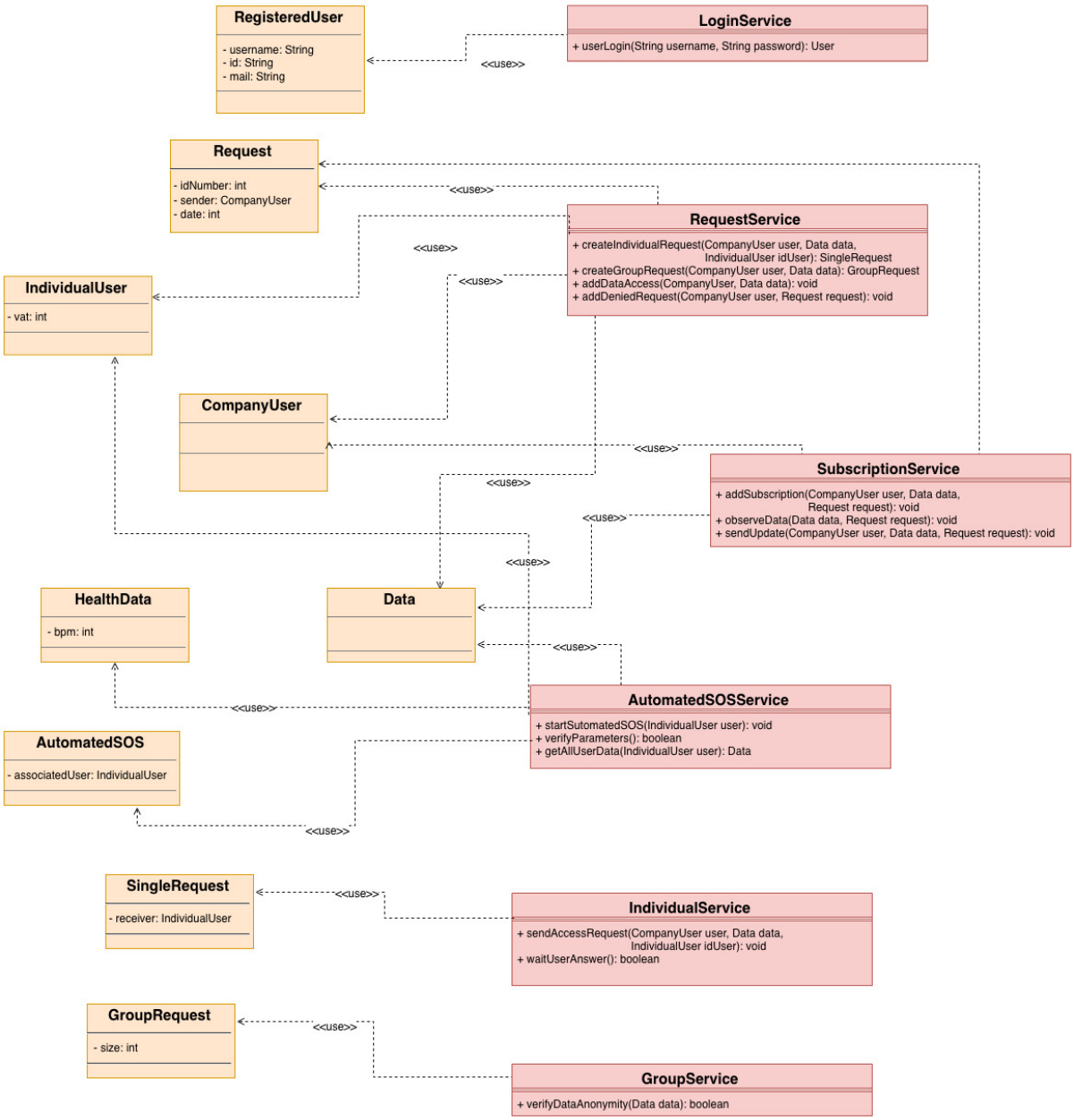

Figure 5: Class diagram (2)

2.3 Deployment View

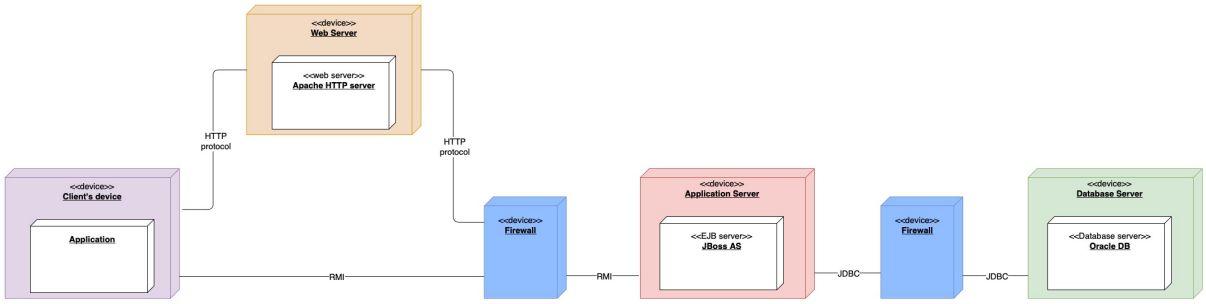

Figure 6: Deployment view

The system architecture is divided in 4 tier:

- The first tier is composed by the *Client's device*, which can communicate both to the application layer directly using RMI system or via HTTP protocol;
- The second tier contains the *Web server* implemented with Apache HTTP platform;
- The third tier consist of the *Application server* which handles all the business logic;
- The fourth and last tier is composed by the *Database server:* the communication between the last two tier is performed using JDBC protocol.

2.4 Runtime View

2.4.1 Create Single Request

In the following sequence diagram is shown the process of a new single request. First of all the Company user has to fill all the field of the request and then the RequestService creates the new single request.
After creating it, the RequestService sends to the IndividualService the request. At this point the IndividualService sends the request to the Individual user whose data are requested and waits the decision.
If the Individual user accepts, the RequestService add the data access to the Company user. Otherwise the RequestService add the request to the denied request of the Company user.

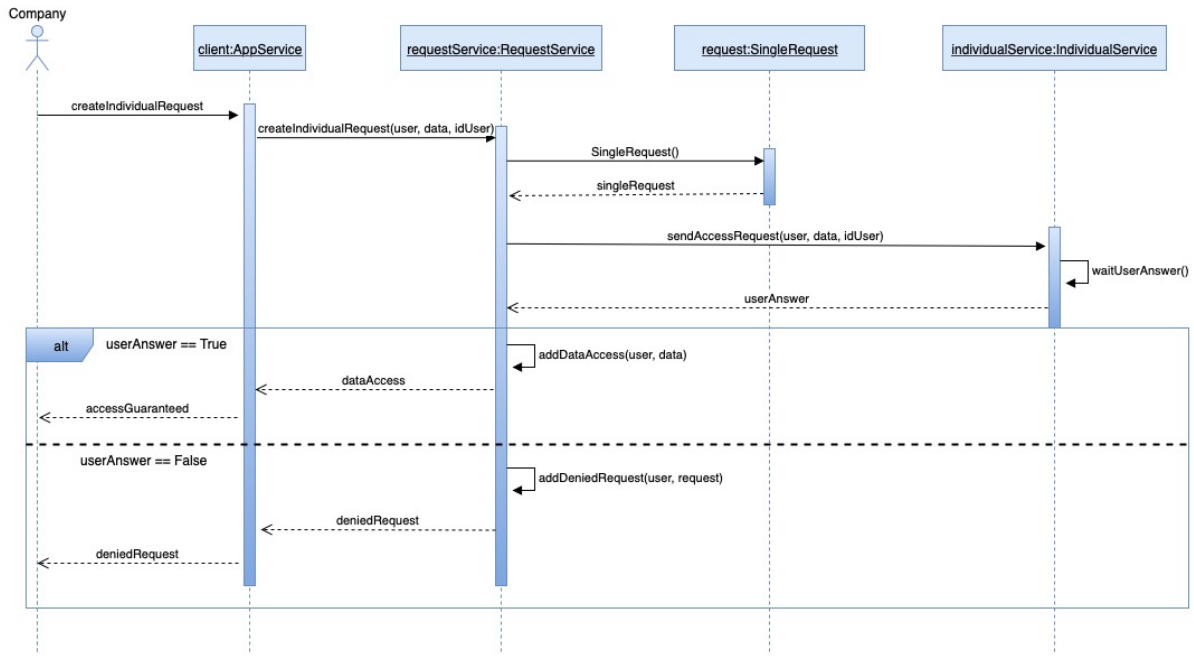

Figure 7: Create a single request

2.4.2 Create Group Request

In this sequence diagram is shown the process of a new group request. First of all the Company user has to fill all the field of the request and then the RequestService creates the new group request.
After creating it, the RequestService sends to the GroupService the request. The task of the GroupService is to verify the anonymity of the requested data.
If the anonymity of the data is respected the RequestService add the data access to the Company user who has sent the request. Otherwise the RequestService add the request to the denied request of the Company user.

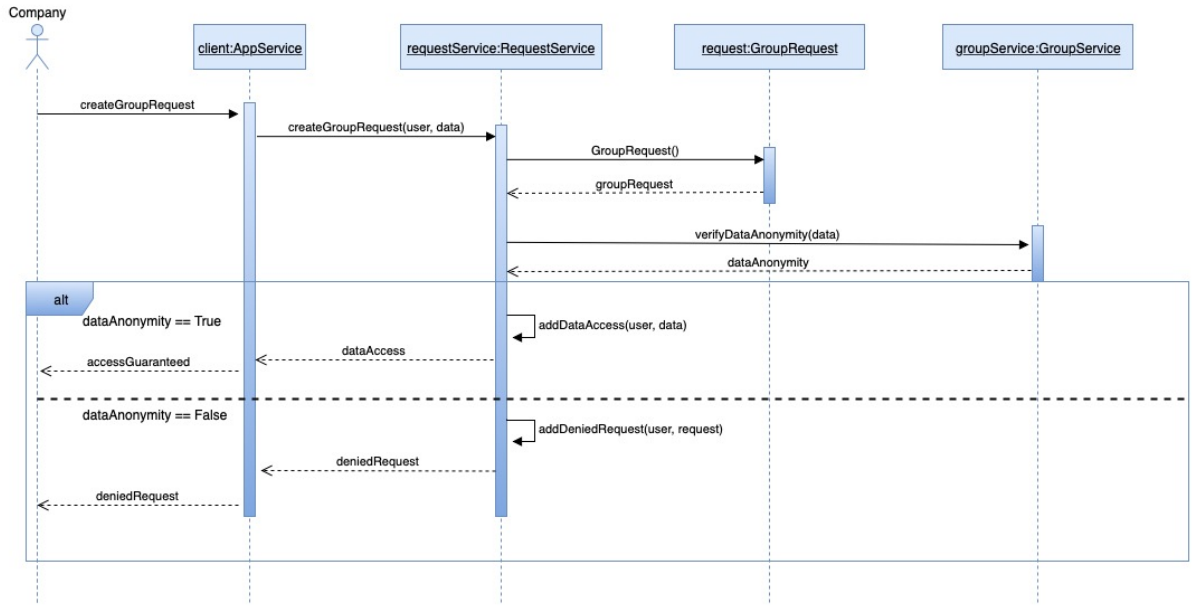

Figure 8: Create a group request

2.4.3 AutomatedSOS Detects Anomaly

In this sequence diagram is shown the process of a new SOS request. After the Individual user has logged in, the AutomatedSOSService starts. This service continues to request data from the wearable device of the Individual user and verifies his health parameters.
If the parameters are under a certain pre-fixed threshold, the AutomatedSOSService recovers all the Individual user data and sends the help request to the AmbulanceExternalSystem. After that notifies to the Individual user that an ambulance is arriving.

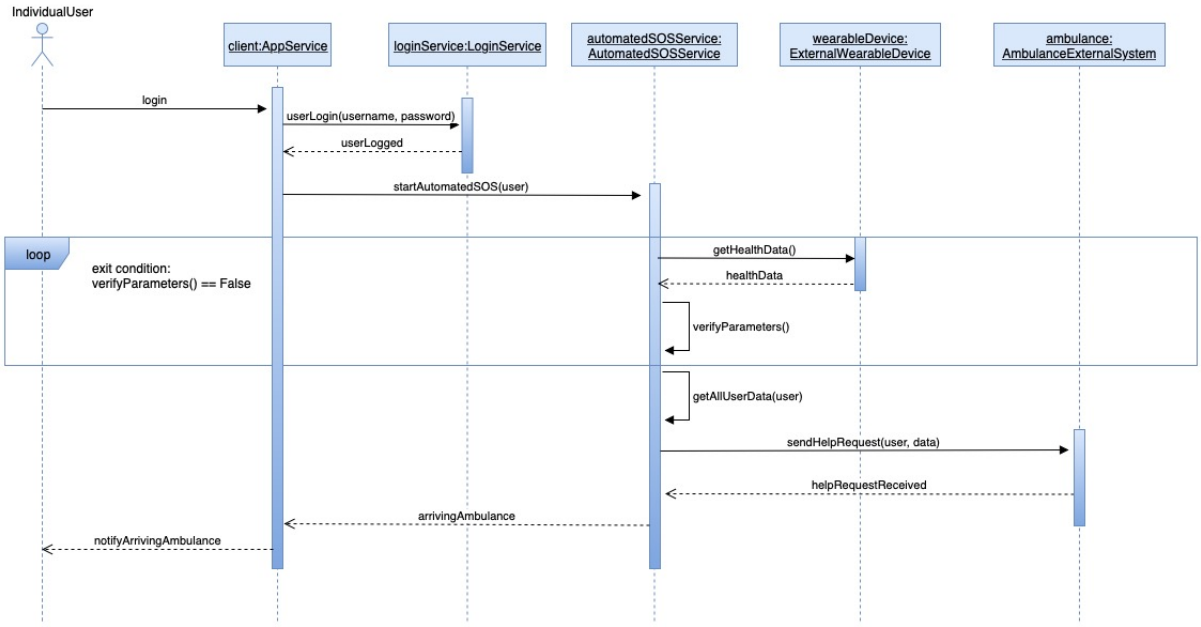

Figure 9: AutomatedSOS detects an anomaly

2.5 Component Interfaces

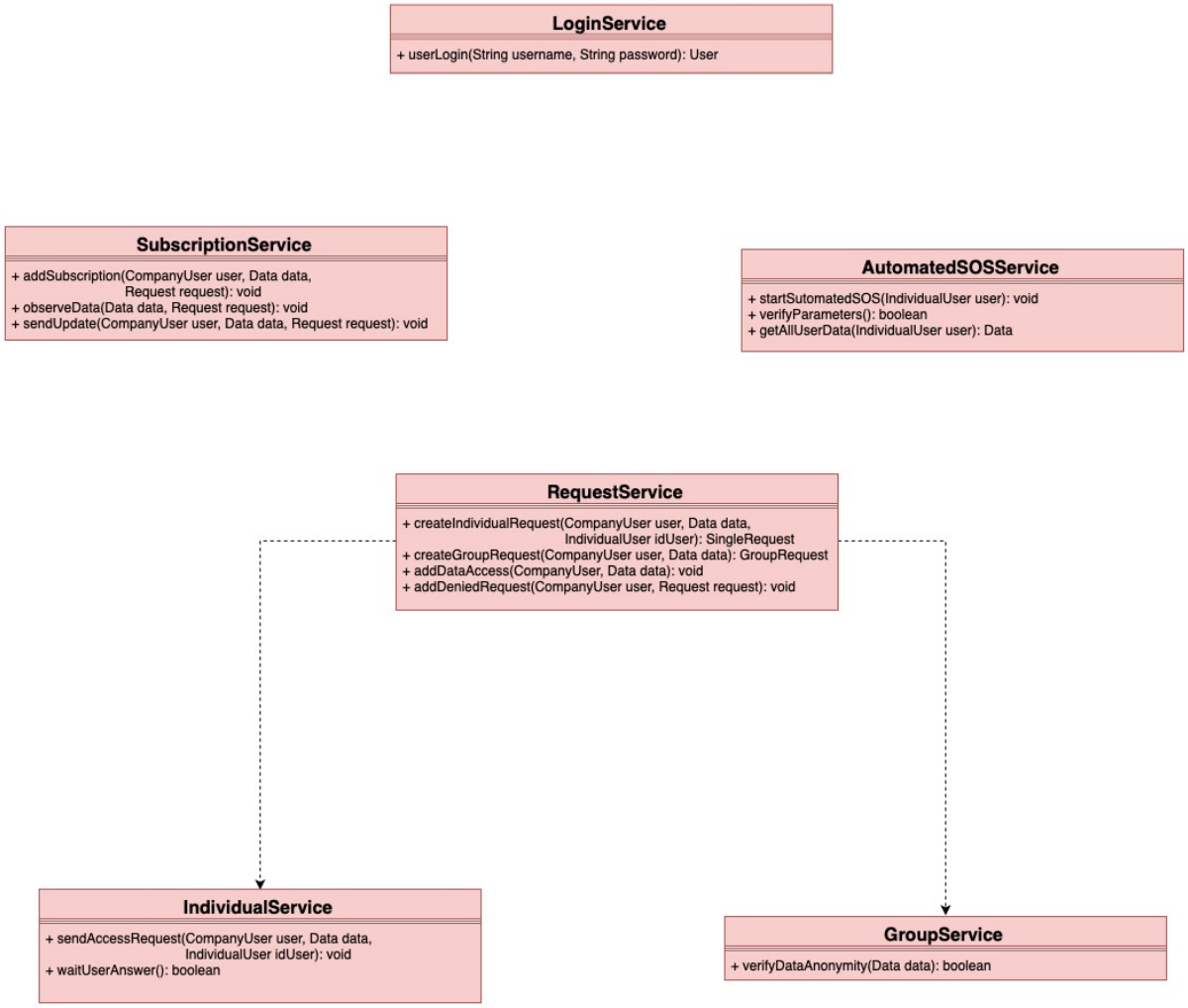

Figure 10: Component interfaces

2.6 Selected architectural styles and patterns

Selected design patterns:

- **Facade pattern** hides the complexities of the larger system and provides a simple interface to the client.

Recommended architectural patterns for implementation:

- **Model-View-Controller pattern** divides a given software application into three interconnected parts, so as to separate internal representation of information from the ways that information is presented to the user. This pattern allows an efficient code reuse and parallel development.

- **Client-Server** architectural pattern allows the application to be scaled up, so that it results independent from the number of clients connected. Moreover improves the security between clients, that knows only the server endpoint but not other clients.

Recommended design patterns for implementation:

- **Strategy pattern** defines a set of encapsulated algorithms that can be swapped to carry out a specific behavior at runtime. We suggest the usage of this pattern for the management of the different request types reducing code duplication.

- **Observer pattern** lets one or more objects be notified of state changes in other objects within the system. It is practically essential for the application of the MVC patter, moreover we suggest to use this pattern to handle the update of the data for the subscriptions.

3 User Interface Design

An idea of the layout of the application development has already been presented in the RASD section 3.1.1.
In addition, the mock-up of the settings management page of the AutomatedSOS service is reported below.

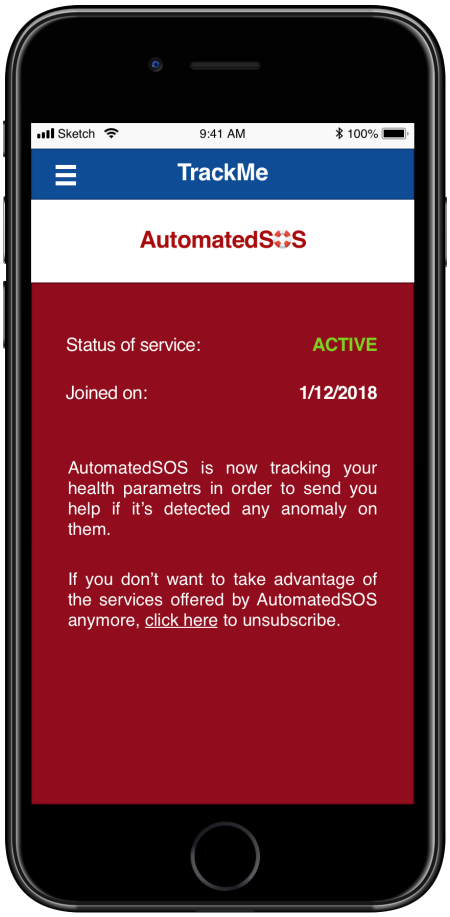

Figure 11: AutomatedSOS settings mock-up

4 Requirement Traceability

While making the design choices presented in this document, the main goal was to fulfill the goals specified in the RASD.
In the following list we present how we map the goals and the requirements defined in the RASD with the system components illustrated in this document.

[G1] Allow an individual to become registered user after providing credentials. (Requirements **[R1]** - **[R4]**)

- LoginServiceImpl

[G2] Allow third parties to become registered user after providing credentials. (Requirements **[R1]** - **[R4]**)

- LoginServiceImpl

[G3] Allow third parties to monitor the location and health status of individuals. (Requirements **[R5]** - **[R8]**)

- LoginServiceImpl
- RequestServiceImpl
- IndividualServiceImpl
- SubscriptionServiceImpl

[G4] Allow third parties to monitor the location and health status of groups of individuals. (Requirements **[R5]** - **[R10]**)

- LoginServiceImpl
- RequestServiceImpl
- GroupServiceImpl
- SubscriptionServiceImpl

[G5] A specific individual can accept or refuse the request of processing his/her personal data from a third party.
(Requirements **[R5]** , **[R6]**)

- LoginServiceImpl
- RequestServiceImpl
- IndividualServiceImpl

[G6] When health parameters are below certain threshold, the system sends to the location of the customer an ambulance.
(Requirements **[R11]** - **[R13]**)

- LoginServiceImpl
- AutomatedSOSServiceImpl

[G7] The AutomatedSOS must guarantee a reaction time of less than 5 seconds in dispatching an ambulance.
(Requirements **[R14]** , **[R15]**)

- AutomatedSOSServiceImpl

5 Implementation, Integration and Test Plan

5.1 Implementation Plan

In order to implement the TrackMe application, a bottom-up approach was selected, in which individual parts of the system are specified in detail, and then connected together to form larger components, which are in turn interconnected up to make a complete system.

Specifically, the order of implementation is as follows:

1. Model
2. IndividualService, GroupService
3. RequestService, SubscriptionService
4. AutomatedService, LoginService

In this way we proceed to implement the most complex components first, such as the management of requests, so as to be able to dedicate more time and resources to the main function of the application.

5.2 Integration Plan

The integration plan foresees a close correspondence with the plan of implantation, since that each component developed must be integrated with the subsequent one for the realization of the complete system.

In fact, the first type of integration required is that between the various components of the application server, following the order previously listed.
Later on will proceed to the integration of the individual components with the DBMS, to guarantee access to the database in which the data are stored.
Subsequently, external services will be integrated with the components that require it.
Finally, the integration between client and server will be performed, to allow the sending of requests to the server through the mobile application that the user will use.

The precise order of the four phases will be as follows:

Integration of components of application server

- IndividualService -> RequestService
- GroupService -> RequestService

Integration of components with DBMS

- RequestService -> DBMS
- SubscriptionService -> DBMS
- LoginService -> DBMS
- AutomatedService -> DBMS

Integration of components with external services

- AutomatedService -> AmbulanceExternalSystem
- AutomatedService -> WearableExternalSystem

5.3 Testing Plan

The use of a bottom-up approach to implementation promotes the start of testing from the first component developed. This involves a simplification in solving programming problems and a major security in making the integration between the various components.
Specifically, unit testing can be carried out as soon as the main functions of a single component have been developed.

An incremental strategy allows us to start the integration and its testing even if each component is not fully developed and tested. In this way the integration testing can start with the components of the application server, then the components with the DBMS and finally the components with external services.

6. Effort spent

6.1 Matteo Velati

TASK	HOURS
Introduction	2
Component View, Interfaces	5
Deployment View, Runtime View	5
Style and Patterns	3
Requirements Traceability	2
Implementation	2
Integration and Testing	3
Various	1
DD	2

Total: about 25 hours

6.2 Niccolò Zangrando

TASK	HOURS
Introduction	2
Component View, Interfaces	5
Deployment View, Runtime View	5
Style and Patterns	3
Requirements Traceability	2
Implementation	2
Integration and Testing	3
Various	1
RASD	2

Total: about 25hours

7. References

- Specific documentation ”AY 2018-2019 Project Assignment”
- Slides from Software Engineering 2 [AA 2018/2019] - Politecnico di Milano
- IEEE Recommended Practice for Design Document