POLITECNICO
MILANO 1863

Politecnico di Milano
AA 2018-2019
Computer Science and Engineering
Software Engineering 2 Project

Dalle Rive Fabio - 920082
Di Giacomantonio Marco - 846515

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to give more technical details than the RASD about TrackMe system. It provides an overall guidance to the architecture of the software product and therefore it is primarily addressed to the software development.

1.2 Scope

The project Data4Help aims to build a system that allows third parties to monitor the position and health status of users. All the data are collected by TrackMe, the company that wants to develop Data4Help, and are shared with other companies which are interested in those data. Furthermore, TrackMe wants to develop AutomatedSOS, a system build on top of Data4Help, designed for elderly people. It is able to intervene by calling an ambulance if the health parameters of the user are below some fixed thresholds. The main target group of the application are the companies interested in users' data. After registration, these companies can request:

- Access to the data of some specific user.
- Access to anonymized data of groups of users.

Another terget group are elderly people, to whom is addessed AutomatedSOS, a non-intrusive SOS service. It is build on top of Data4Help. This service is designed to monitor health status of users and to send an ambulance to the location of the user if some parameters are below some specified thresholds.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Acronyms

- API: Application Programming Interface
- DB: Database
- DBMS: Database Management System
- DD: Design Document
- GPS: Global Positioning System
- GUI: Graphical User Interface
- MVC: Model View Controller
- RASD: Requirements Analysis and Specifications Document
- UI: User Interface
- SSN: Social Security Number

1.3.2 Abbreviations

- [Gn]: n-th goal
- [Rn]: n-th functional requirement

1.4 Document structure

- Introduction: this section introduces the design document. It explains the utility of the project, text conventions and the framework of the document.
- Architecture Design: this section illustrates the main components of the system and the relationships between them, providing information about their operating principles and deployment. This section will also focus on the main architectural styles and patterns adopted in the design of the system.
  This section is divided into different parts:
  1. Overview: this sections explains the tiers division of our application;
  2. High level architecture: this sections gives a high-level view of the components of the application and how they communicate;
  3. Component view: this sections gives a more detailed view of the components of the applications;
  4. Deploying view: this section shows the components that must be deployed;
  5. Runtime view: sequence diagrams are represented in this section to show how our application deals with different tasks;
  6. Component interfaces: the interfaces between the components are presented in this section;
  7. Selected architectural styles and patterns: this section explains the architectural choices taken during the creation of the application and the design pattern used;
  8. Other design decisions.
- User Interface Design: this section presents mockups about the User Interface.
- Requirements Traceability: this section aims to explain how the decisions taken in the RASD are linked to design elements.
- 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.

2 Architectural Design

2.1 Overview

The application uses a three layers architecture.

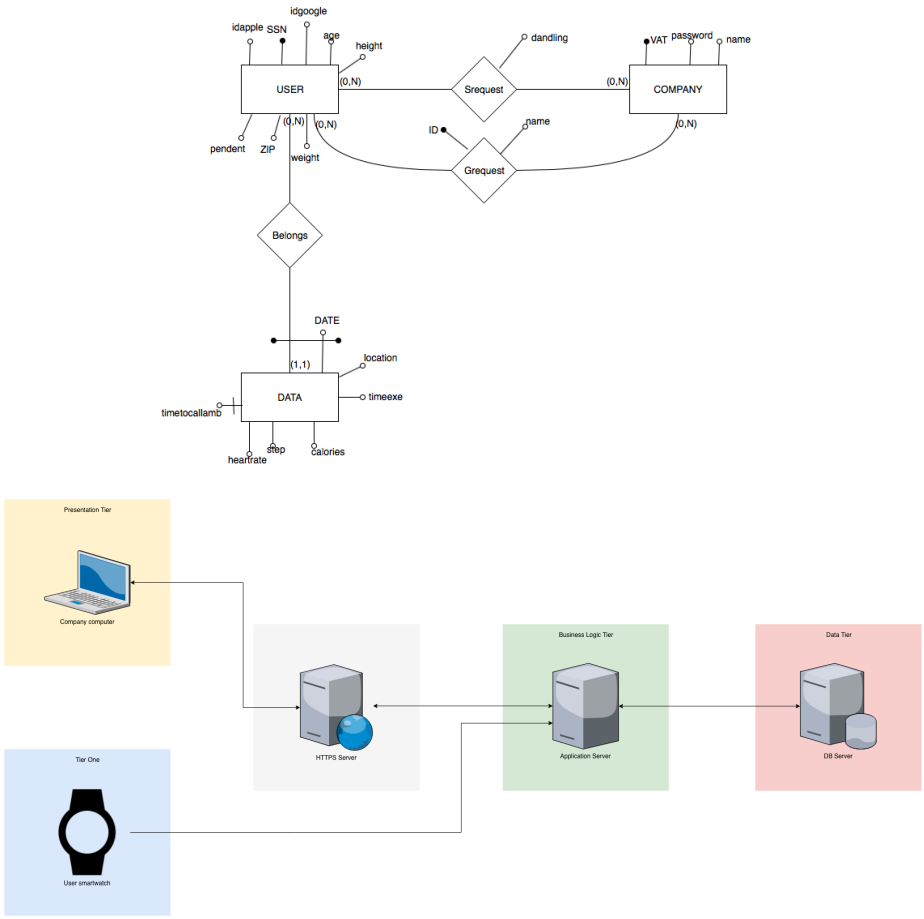
- **Presentation tier**: This layer is used only by companies that want to access to users' data. The user interface consists of a website accessible from a browser. Each web page contains some JavaScript element in order to enable client-side interaction;

- **Tier One**: This layer is only used from the background app running on the smart watch. The function of this layer is to communicate with the server all the data gathered form the sensor present on the smart watch. Data are then communicated to the AppServer every 30 seconds. Since this is the first time we present this concept of data transferring in the project let's look at it in more detail. The system-to-be need to operate as a collector of data gathered by the smart watch. Since it is not feasible that a server asks each smart watch to communicate the data, we need a software module, in the client side, that every *tot* seconds (here 30s) sends data to the AppServer. This module is developed in Tier One. In the rest of the project we will not develop the client side of the application, and we will focus on the server side, nevertheless the presence of this module must be kept in mind.

- **Business Logic tier**: This second layer interacts with the Presentation tier and with Tier One. As regards the interaction with the Presentation tier it enables all the functionality listed in the RASD document, while the interaction with Tier One consists of the processing of the data coming from the user. This layer also communicates with the database in order to permanently store data and with the external services.

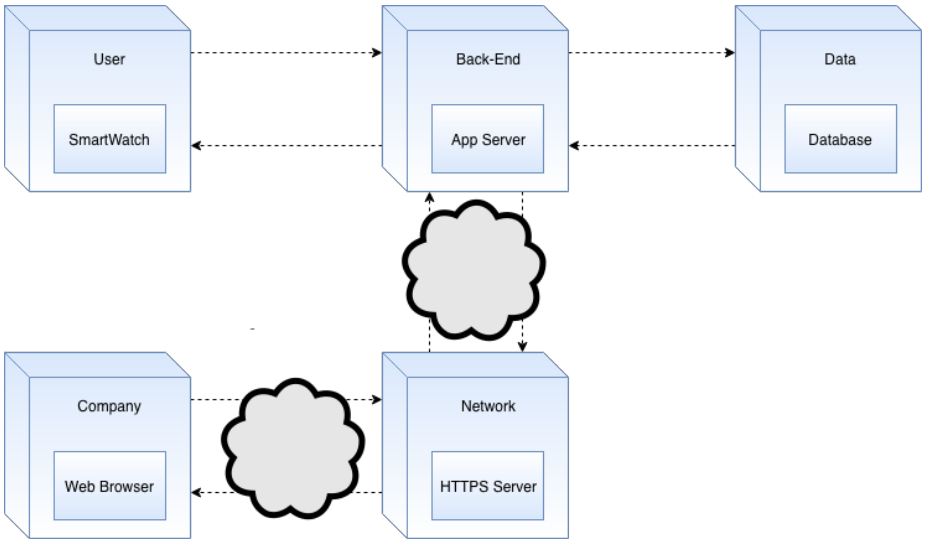
- **Persistence tier**: This layer purpose is to save all the users' (e.g. current position, heartbeat...) and companies' (single user and group user request) data. It shows some interfaces that enable communication with the business logic tier, letting the second level access data.

Here is presented a possible entity-relationship model for the persistence tier.



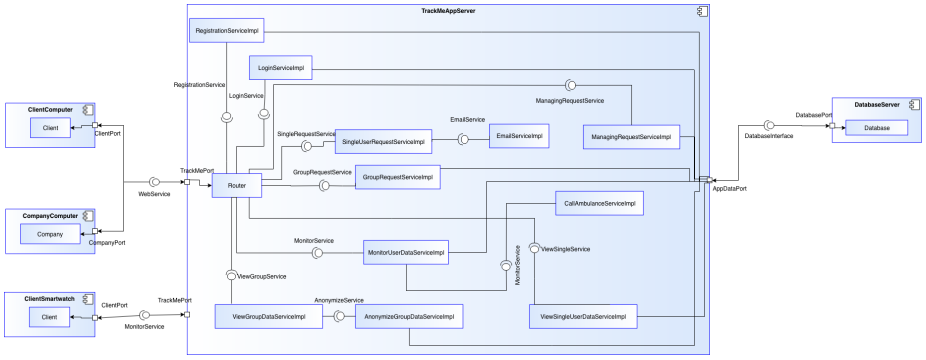
2.2 High Level Architecture

The figure above describes the high-level architecture of the system. All the actors are represented and how they interact with each other. Each tier is mapped with a high level architecture block, as represented in the table above.



	User smart watch	Company Web Browser	BackEnd App Server	Data Database
Presentation Tier		X		
Tier One	X			
Business Logic Tier			X	
Persistence tier				X

2.3 Component View



This diagram shows the components and how they interact with each other. All of them are part of the system, apart from the email service that is external. The email service is only used to send emails between the company and the user and viceversa.
The main component is the application server and it consists of the following

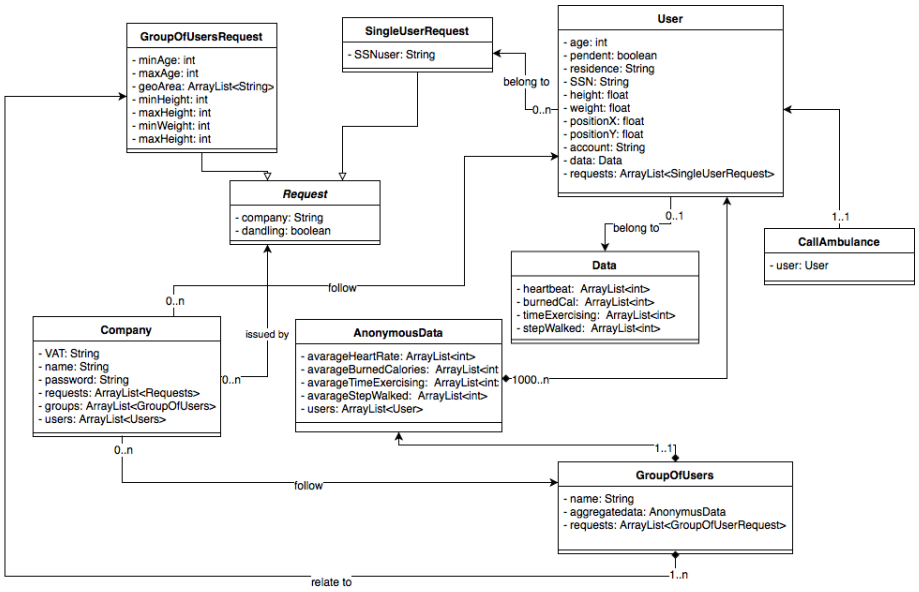
parts:

- LoginService: responsible for authentication of the companies;
- Registration Service: responsible for registration of companies and of single users by using their Apple or Google credentials;
- Single User Request Service: enables companies to formulate a request based on the user SSN in order to access data of a single user;
- GroupRequestService: enables companies to formulate a request in order to access data of a group of users with some specific traits. Furthermore it is responsible for showing an answer to the company request, positive or negative;
- EmailService: external service used to forward companies request to single users and to let them answer these requests;
- ManagingRequestService: enables single user to accept or decline a request by clicking on the corresponding link received in the single user request email that redirects him to our website;
- CallAmbulanceService: external service responsible for calling an ambulance if users heart rate is below *Threshold*. With the term *calling an ambulance* we indicate all the process that this module must follow in order to contact the ambulance and share the user's current location. Due to the fact that this is an external service, that is offered to the system-to-be, the only things that can be said at this point is that during the development of this module, programmers will be provided with all the materials needed to communicate with this module(i.e *primitive calls,...*);
- ViewSingleUserDataService: shows data of a single user to a company. This module query the DB and create a user-friendly graphical representation of data obtained by DB;
- AnonymizeGroupDataService: responsible for anonymizing data of a group of users. This is the component that will actually carry out the process of building a possible group, based on the filter present in the GroupUserRequest issued by a company. All this process is based on the possibility to query the DB and build the group after having verify that more than 1000 user are part of it. This module must also be able to build a sort of intermediate representation of the data, which is sent to the ViewGroupDataServiceImpl, starting from the result of the queries;
- ViewGroupDataService: shows data of a group of users to a company. This module will get the data to display directly from AnonymizeData;
- MonitorUserDataService: responsible for receiving data from the client every 30 seconds. If the user is subscribed to AutomatedSOS checks if the data are below a certain value and communicates with Call Ambulance

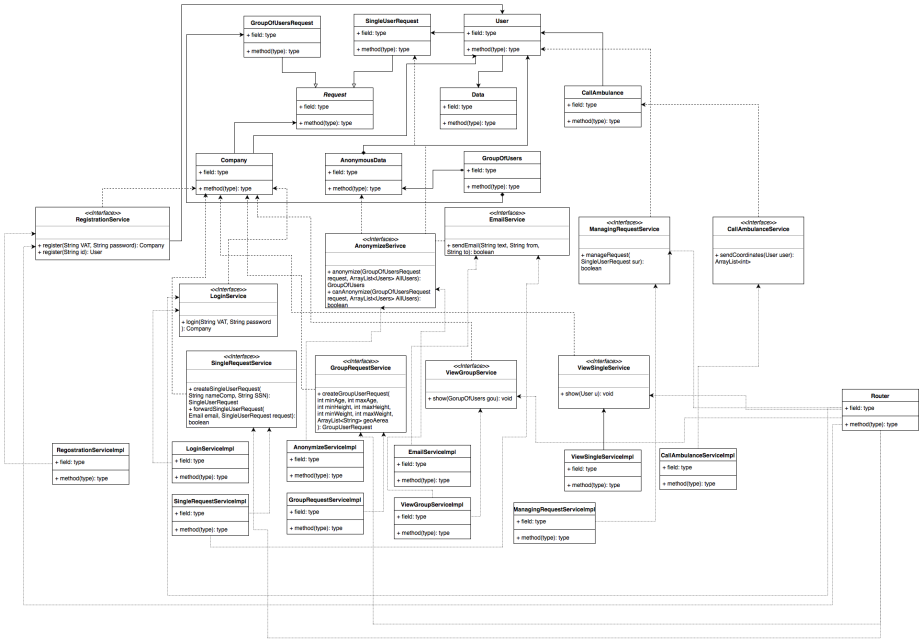
Service. Here it is not meant to go deeper in the development process, but due to the distributed nature of this application we suggest to use message queues to handle the incoming data;

- Router: dispatch a request to the relevant service component;

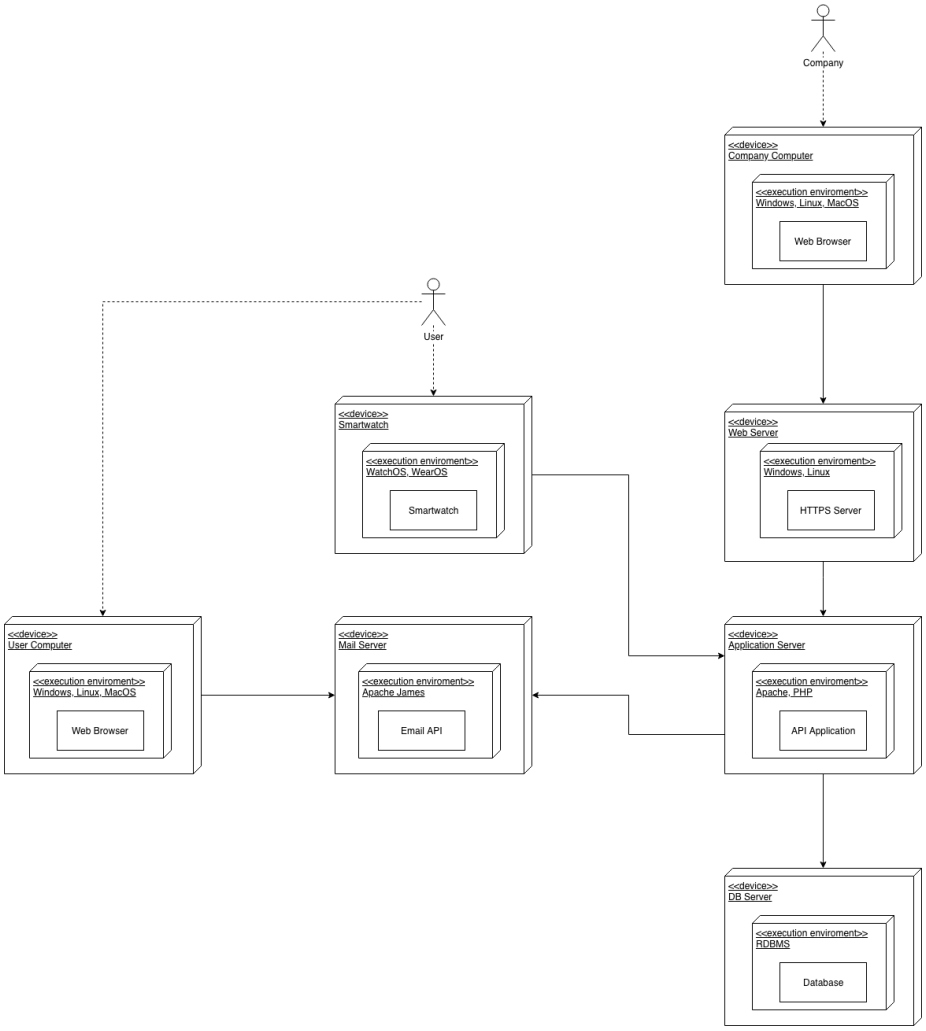
The following diagram shows the Model of the System-to-be.



The following diagram represents the UML of the whole system with emphasis on the View component and is shown the interaction between the result and the model.



2.4 Deployment View



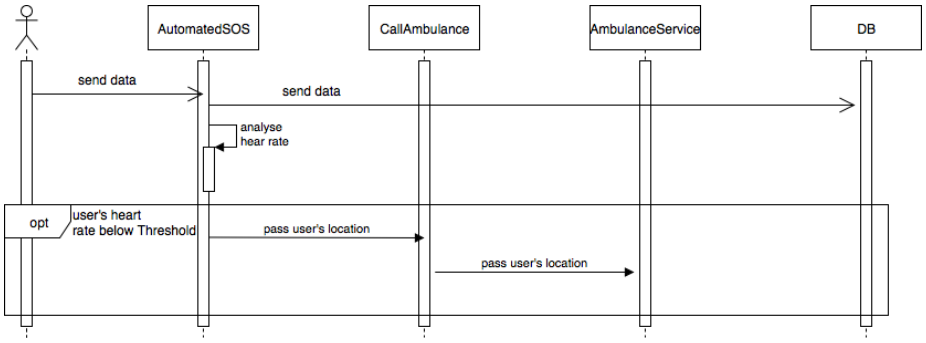
The diagram shows the architecture of the system-to-be. Data4Help requires the deployment of software on these nodes:

- **smart watch**: the application runs in background in order to collect user's data and communicates them to the Application Server;
- **User Web Browser**: the acceptance of any single user request from the user is made with his own mailbox. By clicking on the link the user is redirected on the website and the request can then be accepted or rejected;
- **Company Web Browser**: is used to access to single user data or to

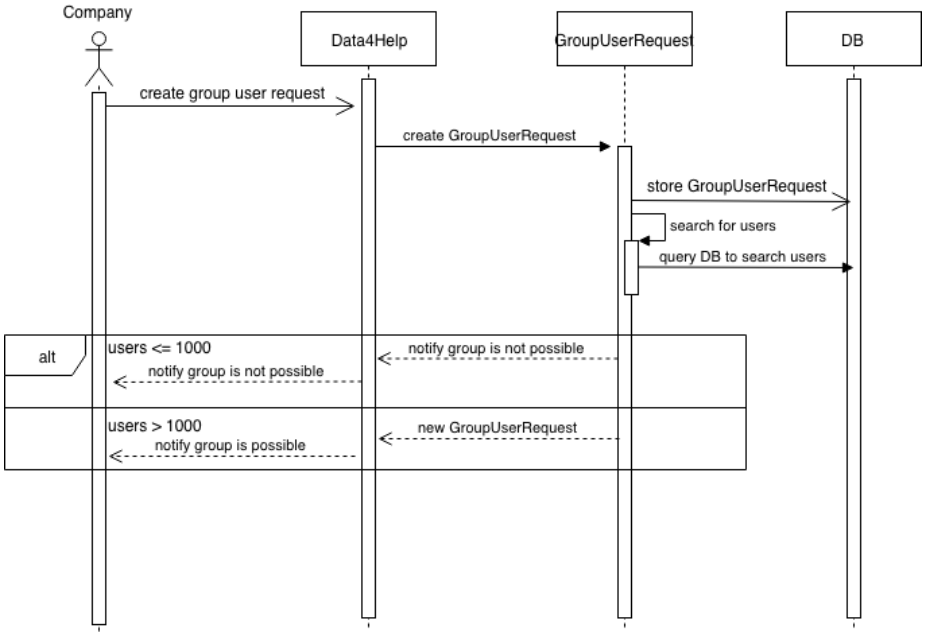
anonymous data of a group of users;

- **Application Server**: the main logic of the application will be deployed here. This server will communicate with all the other nodes - it will collect users' data and store them in the DB server, it will exploit the email server to forward companies requests and it will show to companies the users' saved data from the DB server.
- **DB Server**: it will store all the users' data such as heart rate, current position etc., as well as all the companies requests both for groups and single users;
- **Email Server**: provides the email service used both by the Application server to forward the companies requests and by users to accept them.

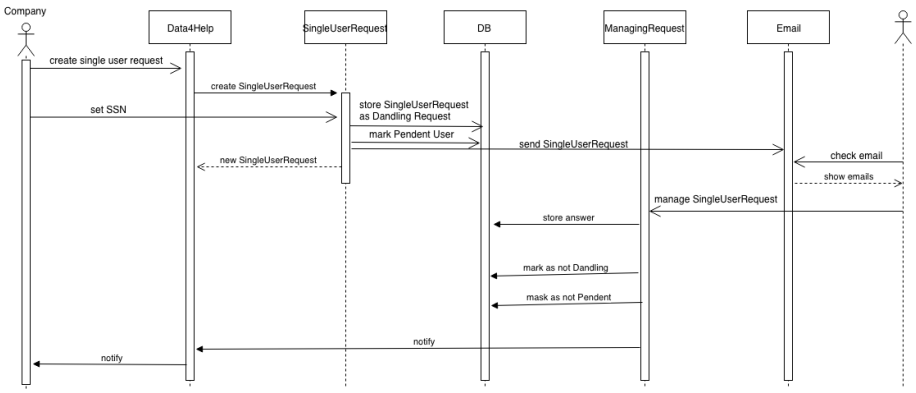
2.5 Runtime View



In this diagram a user, registered to AutomatedSOS, sends with his smart watch his bio-medical parameters and his current location to the System-To-Be. Since the user is already registered to AutomatedSOS there is no necessity to ask for his data, but can access them directly.
Every time that AutomatedSOS recevies new data it checks if they are below a threshold. If so creates an instance of a CallAmbulance class that is responsible for contacting the AmbulanceService and sending the user location.



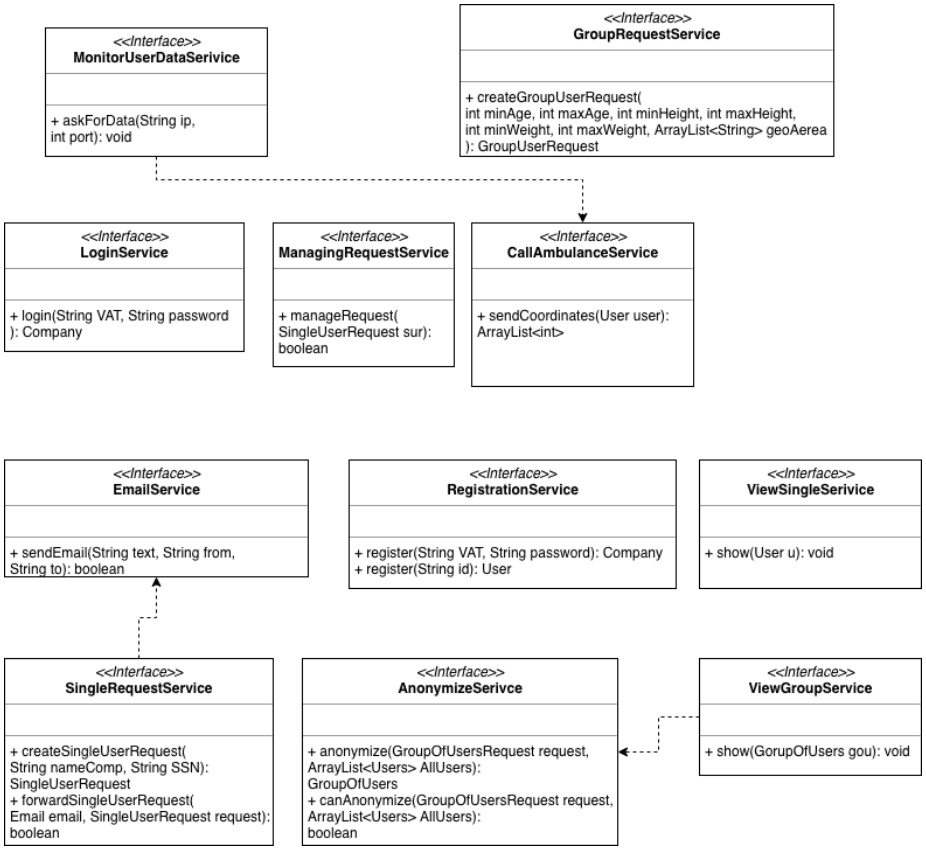
In this diagram a company creates a group user request, to identify the group they can use various filters (e.g. age, ZIP Code, etc.). The Group User Request is stored in the DB. The system searches for the users corresponding to the filters by querying the DB.
The outcome depends on the number of users found: if there are more than 1000 the outcome is positive, a group is created and the company is notified; otherwise the system is notified that a group doesn't exist and also the company is notified.



In this diagram a company creates a single user request by using a SSN. The single user request is stored in the DB as dandling request and the user is marked as a Pendent User. The request is sent to the user by email, he can choose whether accepting it or not by clicking the corresponding link in the email. The link redirects the user into a web page that interacts with the Managing Request Service. The user's answer is stored in the DB, he is marked as not pendent anymore and the request is marked as not dandling as well. In the end the system and the company are notified of the outcome.

2.6 Component Interfaces

This diagram shows the interaction between different component interfaces, this information was already present in the class diagram, but here it is shown more clearly.



2.7 Selected Architectural styles and patterns

2.7.1 Overall Architecture

Our application uses a three-tiers architecture.
The first tier is composed of the presentation tier and the "Tier One", respectively the first is relative to the company and the second is relative to the user.The presentation tier is the only layer that company can access by using the web browser that displays all the corresponding services; Tier One is used by the user in order to communicate his data to the App server. Both of them communicate only with the Business Logic Tier.
The second tier, the Business Logic one, controls the application's functionality by performing processing on users' data and is where lies all the logic that manages with the single or group requests.
The data tier, or Presistence tier, is where all the data are permanently stored.

2.7.2 Design Pattern

Recommended architectural pattern:

- Model - View - Controller (MVC): The model is the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application. The view is any output representation of information. The controller, accepts input and converts it to commands for the model or view. In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them. The model is responsible for managing the data of the application. It receives user input from the controller. The view means presentation of the model in a particular format. The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

- Client - Server: The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a service to many clients, which initiate requests for such services. The client only has to understand the response based on the well-known application protocol. Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response. All client-server protocols operate in the application layer. This pattern has been chosen for different reasons:

  - It's pratical.
  - Data synchronization: there is only one application that manage the data.
  - Having one unique server application improves the maintenability.

  - the application is independent from the number of clients connected.
  - improves the security between clients, that known only the server endpoint but not other clients.

Recommended behavioral pattern:

- Observer: lets one or more objects be notified of state changes in other objects within the system. It is useful for the AutomatedSOS external service, in order to observe the data coming from the user and in case instantiate the CallAmbulance class.

2.8 Other Design Desicions

The users data should be sampled at regular intervals, i.e. every 5 seconds.
The system uses the email external service in order to let the user accept or reject companies' requests.

3 User Interfaces Design

The mock-ups for this application have already been presented in the RASD Document.

4 Requirements Traceability

Here is presented the mapping between the RASD goals and software components:

- [G1] Visitor can become User after providing credentials. (Requirement **[R1]**)
  - RegistrationServiceImpl
- [G2] User can accept or reject the request of access to his data formulated by companies. (Requirements **[R2] - [R3]**)
  - ManagingRequestServiceImpl
  - EmailServiceImpl
- [G3] If user's parameters are below specified thresholds, an ambulance is called within 5 seconds. (Requirement **[R4]**)
  - CallAmbulaceServiceImpl
  - ViewSingleUserDataServiceImpl
- [G4] Company can sign up as Company to Data4Help and AutomatedSOS. (Requirement **[R5] - [R6]**)
  - RegistrationServiceImpl
- [G5] Company can be recognized providing a password and vat number. (Requirement **[R7]**)
  - LoginServiceImpl
- [G6] Company can formulate a request to see anonymized data of a group of users. (Requirement **[R8]-[R9]**)
  - GroupRequestServiceImpl
- [G7] Company can formulate a request to see data of a specific user providing his SSN. (Requirement **[R9]-[R10]**)
  - SingleUserRequestServiceImpl
- [G8] Company can see anonymized data of a group of users. (Requirement **[R11]-R[12]-R[13]**)

  - ViewGroupDataServiceImpl
  - AnonymizeGroupDataServiceImpl

- [G9] Company can see data of a specific user providing his SSN. (Requirement [**R14**])
  - ViewSingleUserDataServiceImpl

- [G10] Company can subscribe to users' new data. (Requirement [**R15**])
  - ViewSingleUserDataServiceImpl
  - ViewGroupDataServiceImpl
  - AnonymizeGroupDataServiceImpl

- [G11] Data4Help can forward companies' requests to users. (Requirement [**R9**]-[**R16**])
  - SingleUserRequestServiceImpl
  - EmailServiceImpl

- [G12] A user of Data4Help becomes a user of AutomateSOS if he is older than *Age* (Requirement [**R17**]-**R**[**18**]-**R**[**19**])
  - RegistrationServiceImpl

5 Implementation, Integration and Test Plan

5.1 Implementation Plan

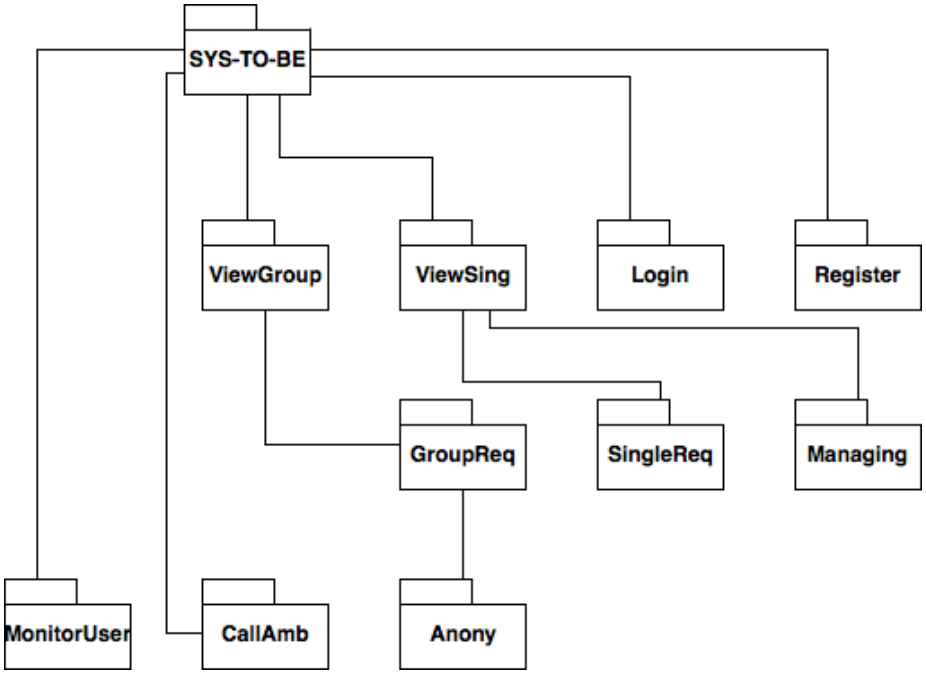
In the next pages a step-by-step plan of module development is presented. In order to achieve a complete test coverage, of all the functionality that the system-to-be must have, we suggest to adopt a bottom-up approach. This means that once a module is build, it has to be tested individually and then it can be integrated we other already tested modules. This methodology also drive the implementation plan, that is build looking at the way and order in which tests will be run. It turns out that the already presented components can be integrated following these list:

1. Model
2. MonitorUserDataServiceImpl
3. CallAmbulanceServiceImpl
4. AnonymizeGroupDataServiceImpl
5. SingleUserRequestServiceImpl, GroupUserRequestServiceImpl, ManagingRequestServiceImpl
6. ViewSingleUserDataServiceImpl, ViewGroupDataServiceImpl
7. RegistrationServiceImpl, LoginServiceImpl

The Model is the first component to be implemented. This is done to build the common ground over which the system-to-be will operate. Doing the Model as first component will force programmers to develop the project having in mind what are the data structures to deal with and their semantic in the system-to-be.
Once the Module is done, MonitorUserDataServiceImpl must be developed. This is the component that every 30 second receives data from all the smart watches and write them in the DB. Because all the other operation that the system-to-be needs to carry out are some how related to data, here is the reason why the first component to implement must be MonitorUserDataServiceImpl.
The next component it is suggested to implement is CallAmbulanceServiceImpl. This is the core component of AutomatedSOS. Since all the AutomatedSOS is basically done through this component, it must be developed and tested it in the very initial stages of the development process. This will let to show the stakeholders a product that has at least the core functionality.
The third block to develop is AnonymizeGroupDataServiceImpl. It is used in by the ViewGroupDataserviceImpl so it must be surely developed before this module, and it is used by the system-to-be, to asks GroupUserRequest, so it also might be developed before this component.
The fourth blocks of components that need to be developed are those related with the *requests* issued by companies. Suppose to have completed the previous

steps, these modules will let programmers test a product that almost have all the relevant functionality, since the visual display of the result is not a logical function per se.
The development of the remaining blocks can be done in parallel since they are not related.

5.2 Integration and Test Plan



Here is presented a representation of how the system-to-be must be integrated. How was previously stated in the implementation plan in the development of the modules programmers should be driven by the tests to be performed over the system-to-be. The test plan is carry out following the bottom-up approach. This methodology give the possibility to test each functionality in isolation and integrate it in the final system after have tested it with the modules that used it. This process is time consuming and also quite heavy for the programmers, but it force them to test each functionality as soon as it is produce. So far the Model was not mentioned, but this is just because it is not a functionality, nevertheless it is obvious that all the different modules can be tested *if and only if* the Model is implemented and tested.

6 Effort Spent

7 Resources

7.1 Used Tools

- Texmaker 5.0.3
- Adobe Photoshop
- https://www.draw.io

7.2 References

- Specification document "Mandatory Project Assignment AY 2018-2019"
- Software Engineering 2 Course Slides