Politecnico di Milano

Software Engineering 2 Project
a.y. 2018-19

Data4Help, AutomatedSOS

Design Document

Version 1.0

Comolli Federico, 920258
Corda Francesco, 920912

Referent Professor: Di Nitto Elisabetta

December 10, 2018

Table of Contents

1 Introduction 3

1.1 Purpose . . . 3
1.2 Scope . . . 3
1.3 Definitions, Acronyms, Abbreviations . . . 4
1.3.1 Definitions . . . 4
1.3.2 Acronyms . . . 4
1.3.3 Abbreviation . . . 5
1.4 Revision history . . . 5
1.5 Reference Documents . . . 5
1.6 Document Structure . . . 5

2 Architectural Design 6

2.1 Overview . . . 6
2.2 High Level Components and their interactions . . . 7
2.3 Component View . . . 9
2.4 Deployment View . . . 13
2.5 Runtime View . . . 15
2.6 Component Interfaces . . . 24
2.7 Selected Architectural Styles and Patterns . . . 27
2.8 Other Design Decisions . . . 28

3 User Interface Design 29

3.1 Mock-up Interfaces . . . 29
3.2 User Experience Diagrams . . . 31
3.3 Boundary Control Entity Diagrams . . . 34

4 Requirements Traceability 38

4.1 Functional Requirements . . . 38
4.2 Quality Requirements . . . 41

5 Implementation, Integration and Test Plan 42

5.1 Implementation Plan . . . 42
5.2 Integration and Testing . . . 45
5.2.1 Entry Criteria . . . 45

5.2.2 Elements to be Integrated 46
5.2.3 Integration Testing Strategy 48
5.2.4 Sequence of Component/Function Integration 48

6 Effort Spent 55
6.1 Comolli Federico 55
6.2 Corda Francesco 56

7 References 57

Section 1

Introduction

1.1 Purpose

The purpose of the Design Document is to give a detailed description of how Data4Help and AutomatedSOS are designed. This document is addressed to the developer of the system and aims to identify the high level architecture and to describe the behaviour of its components.

1.2 Scope

Data4Help is an application-to-be that allows third parties to monitor the location and the health status of its users. It is projected to be installed on different wearable electronic devices (smartphones, smartwatches, smartbands, etc.).
Data4Help collects different parameters provided by sensors on smart-watches or health tools (such as heart rate, skin temperature, blood glucose level, weight, etc.) and also acquires the position of its users through GPS technology.
Third parties interested in data acquired by Data4Help can make a request using the functionality provided by the application. In addition it's possible to make requests referred to group of people. Data4Help accepts or declines these requests using a defined privacy criteria. Users who receive a request for their personal information can respond using the provided functionality.
Furthermore, TrackMe wants to build a new application on top of Data4Help, with the purpose of exploiting its functionalities to provide an emergency service.
AutomatedSOS analyzes real-time data, provided by Data4Help technology. If the software detects an anomaly in the parameters it notifies an emergency to the Operations Center of the National Health Service within 5 seconds. The intervention is carried on by the NHS that determines the emergency level and sends an ambulance to the location of the user.
In the following pages we will present the design decisions made to achieve all the functionalities that the system has to offer.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Users: people who use the services provided by TrackMe.
- Third Parties: entities that are interested to data provided by TrackMe.
- National Health Service: the national institution that provides health care to citizens.
- Social Security Number: a nine-digit-number that identifies uniquely a citizen.
- Fiscal Code: synonym to Social Security Number for Italian people.
- Credentials: username, password, SSN or Fiscal Code.
- Anomaly: when the parameters are below a defined threshold.
- Emergency: it occurs when an anomaly is detected.
- System: defines the set of software components that implement the required functionalities.
- Real-time acquisition: the interval of time between two different acquisitions is less than 2 seconds.
- Operations Center: the public authority that coordinates and manages the first aid operations.

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document.
- DD: Design Document.
- SSN: Social Security Number.
- GPS: Global Position System.
- API: Application Programming Interface.
- NHS: National Health Service.
- OC: Operations Center.
- TP: Third Party.
- SOA: Service Oriented Architecture.
- UX: User Experience.
- BCE: Boundary Control Entity.

1.3.3 Abbreviation

- [Gn]: n-th goal
- [Rn]: n-th functional requirement
- [Dn]: n-th domain assumption

1.4 Revision history

- Version 1.0 delivered on date 10/12/2018.

1.5 Reference Documents

- Specification Document: "Mandatory Project Assignment AY 2018-19".
- 1016-2009 - IEEE Standard for Information Technology, Systems Design, Software Design Descriptions.
- Software Engineering II course slides, AY 2018-19, prof. Di Nitto.
- Software Engineering II projects examples, AY 2017-18 and 2016-17.

1.6 Document Structure

Section 1 gives an introduction to the problem and describes the purpose of the applications Data4Help and AutomatedSOS. The scope of the two applications is defined.

Section 2 gives an Overview of the system and illustrates the main components and their relations. The Component View section deepens the analysis for each subsystem and the Deployment View describes how the system will be implemented in the architectural infrastructure. The Runtime View shows the interaction sequences between the modules. Then, the Interfaces provided by the components are showed. Moreover, a description of the main Architectural Styles and Patterns adopted in the design is given.

Section 3 presents mockups and further details about the User Interfaces using UX and BCE diagrams.

Section 4 maps functional requirements with the components responsible for their realization and quality requirements with the design decisions made during the development of the system.

Section 5 shows the effort spent by each group member while working on this document.

Section 6 includes the reference documents used to compose the project analysis.

Section 2

Architectural Design

2.1 Overview

This chapter presents a general overview of the system architecture. The description will concentrate on the logical and physical components that we choose in order to build our system.
We will start by illustrating the *High Level components* of the system and their interactions.
Then, the section *Components view* will focus on describing the single components and their interfaces.
The *Deployment view* will provide the strategy we adopted to deploy the system, highlighting the load division between different tiers of the system.
In the *Runtime view* we will illustrate different examples of interaction between the components with the use of Sequence Diagrams.
The *Component interfaces* section contains a description of the interfaces provided by the logical components.
The chapter ends with a selection of *Architectural styles and patterns* and *Other design decisions* chosen while composing the application.

2.2 High Level Components and their interactions

The following image illustrates the general architecture on which the system will be based on. Every component can be associated with one of three logical tier: Data, Application, Presentation.
This is a standard and well accepted architecture, that will promote the independence between logically separated components.
During the design phase of the application, our focuses are the ones illustrated in the RASD document: maintainability, security, reliability and availability.
Given the nature of the two applications, a robust design of the Data tier is fundamental for the success of the project.
All the information related to the system will be stored in multiple databases. These will be protected from external attacks by a firewall and will be regularly subjected to back-up procedures.
The Application tier contains the logic that controls the core services of Data4Help and AutomatedSOS. The applications will be installed on different servers, located in a demilitarized zone.
To conclude the general description, the Presentation tier contains the system interface to the external world. The applications will be accessible both by mobile applications and web browsers.
Users accessing the applications via internet browsers will interact with a web server that will provide the web interface. The mobile applications instead will communicate directly with the application server.

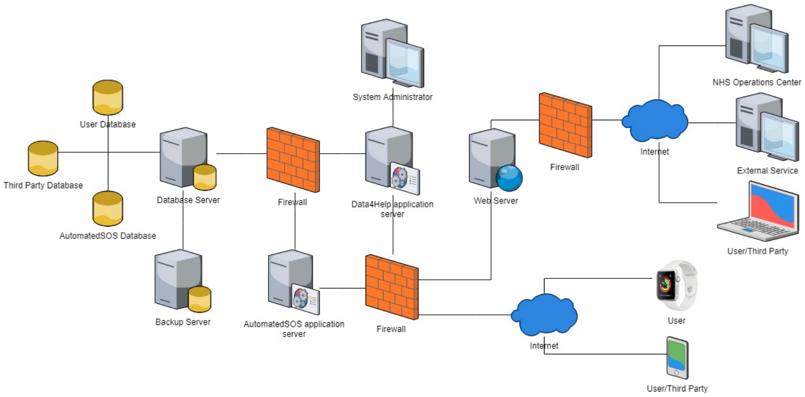


Figure 2.1: Overview.

The UML diagram below describes the main logical components that compose the system and the connections between them. Every component enclose a specific group of services offered by the system and is not intended as a node of the physical architecture.
Even if this is a very high-level representation of the system it's possible to identify a general subdivision in client-related services, back-end logic and data management components.
A more detailed description of each component will be given in the next section.

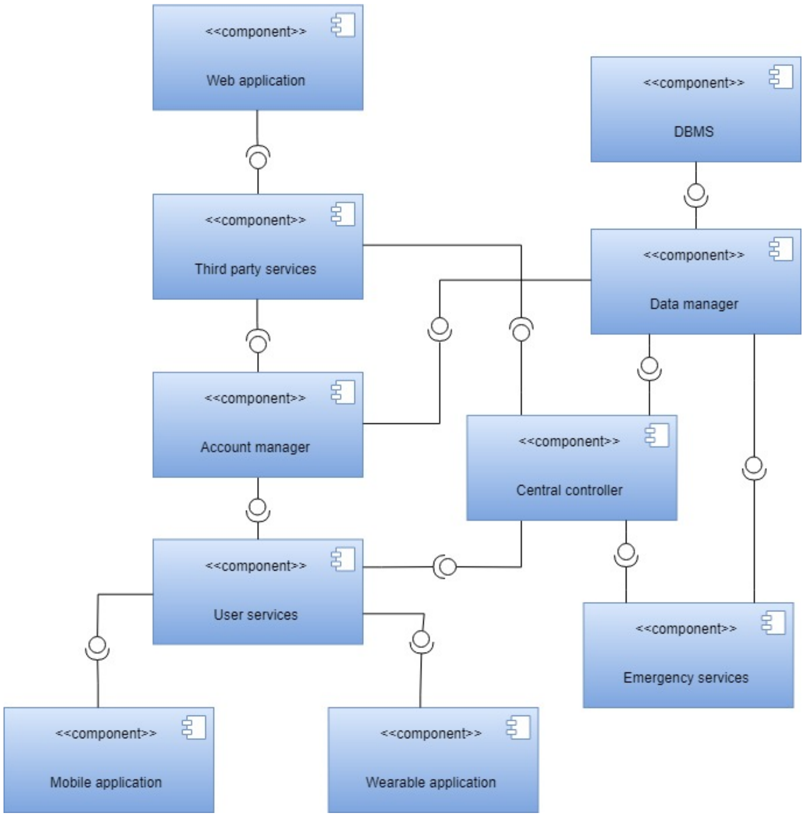


Figure 2.2: High Level Components.

2.3 Component View

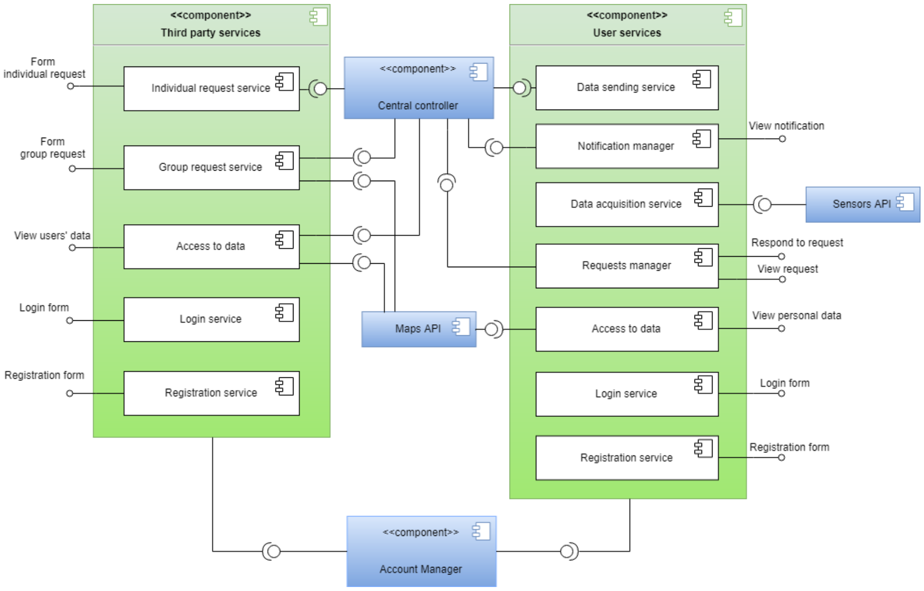


Figure 2.3: High Level Components.

The image above contains an expanded version of the two components User Services and Third Party Services. We chose to show these components together because of their similarities and common interactions. These components connect the client application with the internal logic. *Third Party services* provide an interface for every functionality offered to the presentation tier:

- **Individual request service**: module that receives individual requests from a third party.
- **Group request service**: module that receives group requests from a third party.
- **Access to data**: provides the possibility to access and download users' data.
- **Login Service**: lets the third party sign in to the application.
- **Registration Service**: lets the third party register to the application.

User services it's the counterpart component for monitored users:

- **Data sending service**: it's the module responsible for sending the user's collected data to Data4Help back-end.
- **Notification Manager**: sends notifications to the user.

- **Data Acquisition service**: the module that interacts with the sensors API to collect the health data and the positions.
- **Requests Manager**: module that shows incoming requests to the user and receives his/her replies.
- **Access to data**: lets the user access and download their data.
- **Login Service**: lets the user sign in to the application.
- **Registration Service**: lets the user register to the application.

Both these components are also connected to the Central Controller and the Account Manager modules.

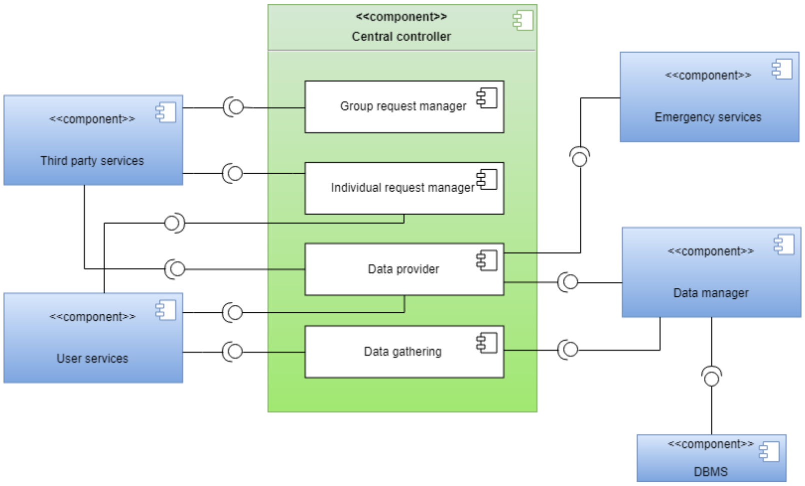


Figure 2.4: High Level Components.

The Central Controller is the core of the system, it's modules manage the acquisition, storing and exchange of data between users and third parties:

- **Group request manager**: this module contains the logic concerning group requests. After receiving a request from a third party, it checks that the privacy condition is respected and authorizes or forbids the access to data.
- **Individual request manager**: this module receives the individual requests and forwards them to the corresponding user. After receiving the response from the recipient it authorizes or forbids the access to data.
- **Data provider**: service that provides users' data to authorized applicants.
- **Data gathering**: module that receives users' data and stores them by using the Data Manager component.

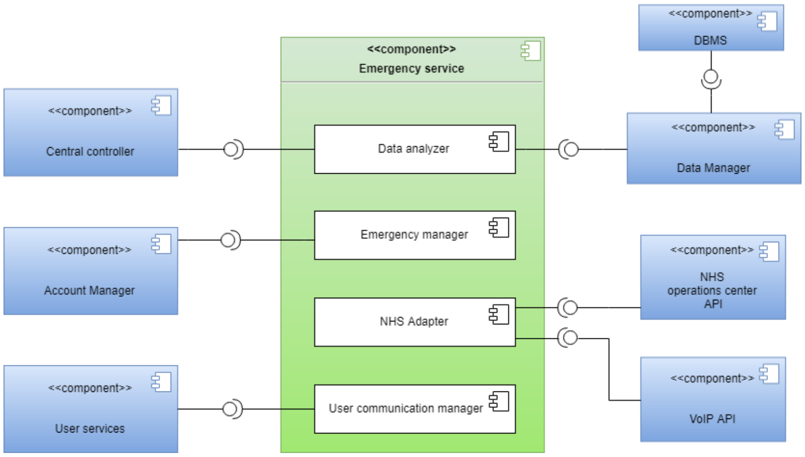


Figure 2.5: High Level Components.

The Emergency Service is the component that manages the logic behind AutomatedSOS. Since this application is built on top of Data4Help, the emergency service is conceived as a stand-alone module inside the back-end logic of the system. AutomatedSOS exploits Data4Help technology to obtain users' data from the Data Provider of central controller. The two systems share the same Account Manager, so every user that registers to AutomatedSOS is also a Data4Help user.

- **Data analyzer**: connects to the central controller to acquire users' data and analyzes them in real time. This module also connects to the Data Manager to obtain the parameters thresholds.
- **Emergency Manager**: if the data analyzer detects an anomaly, it alerts the Emergency Manager. This module creates an emergency message containing the user personal credentials, its current position and most recent health parameters.
- **NHS Adapter**: this module sends the message generated by the Emergency Manager, adapting it to the interface provided by the National Health Service or making an automatic call to the Operations Center using the VoIP protocol if the NHS doesn't provide an API in the hosting country.
- **User Communication manager**: this module is responsible for the communication between AutomatedSOS and the user. If the user is having a health emergency, the Communication Manager notifies him about the ambulance arrival.

2.4 Deployment View

In this section we illustrate the deployment of the system components on the hardware infrastructure. The design of the hardware architecture was made keeping in mind the non-functional requirements illustrated in the RASD document. The objective was to obtain a reliable and scalable system, but also to avoid an over-complicated disposition, that would increase the costs of construction and maintenance.
We chose to deploy the system using a four tiers architecture:

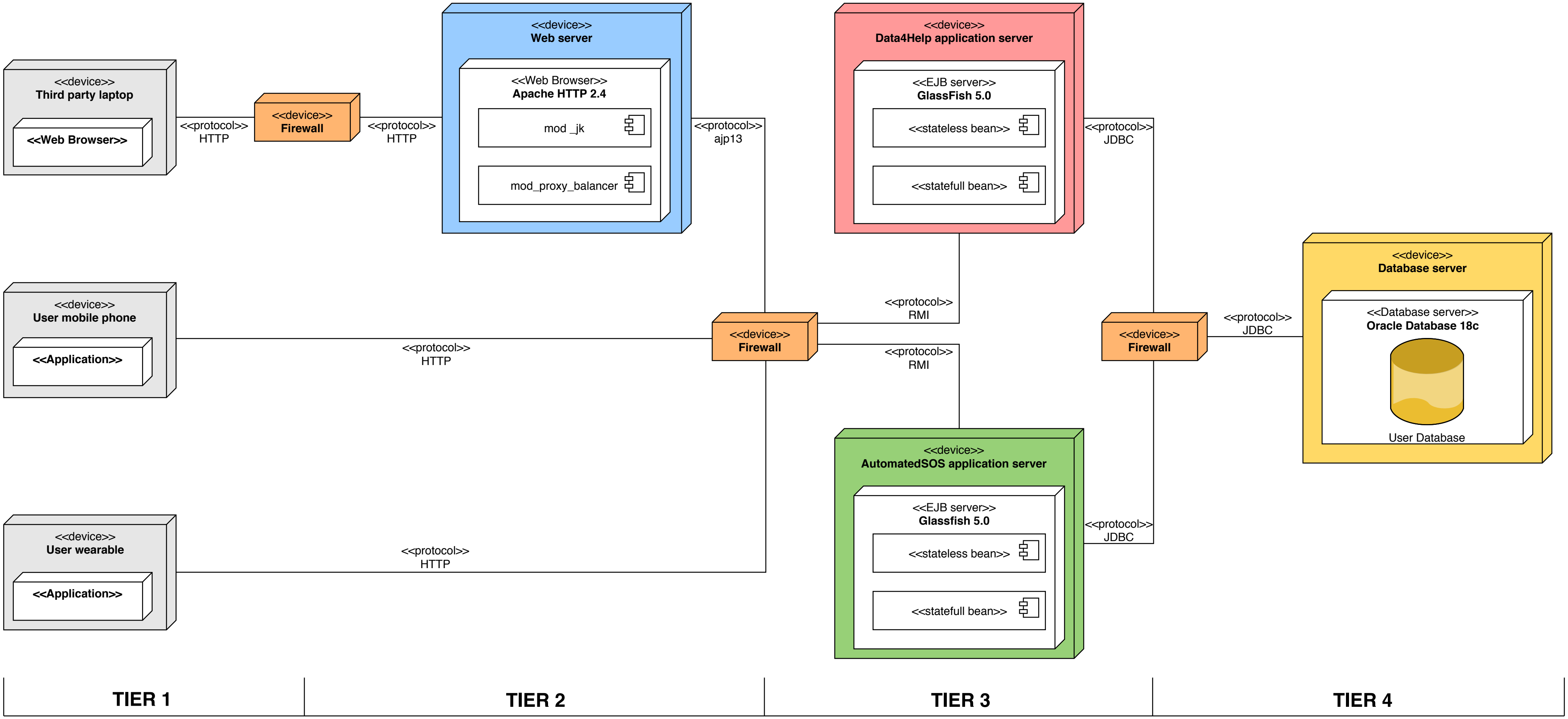
- the **first tier** represents the client applications:
the system will be accessible by smartphones, wearables and web browsers. All these platforms will exchange messages using the HTTP protocol. There will be two different apps for Android and iOS, implemented using the native frameworks of the operating systems. We made the decision to avoid cross platform development solutions to build the client apps, because they will be a very small portion of the code base and will provide only presentation services. Both applications will connect to the same interfaces provided by the system. We also chose to avoid other mobile platforms, since the current scenario in mobile OS clearly shows a duopoly of two competitors that will unlikely change in the next years. The web client will be developed using HTML5 and CSS for the design and JavaScript for the web page logic.

- in the **second tier** will be placed the web server:
implemented using an Apache HTTP architecture, it will be responsible for the communications with web browsers.

- the **third tier** will be the core of the system:
two application servers will contain Data4Help and AutomatedSOS central logic. They will be based on Glassfish[6], the reference implementation by Oracle of Java EE[7] specifications. The use of JEE promotes the creation of a multi-platform ("Write Once, Run Anywhere"), portable and scalable central system. In particular, Glassfish offers complete support to Enterprise JavaBeans, JMS, RMI and other JEE key features. Furthermore, this platform provides robust off-the-shelf mechanisms to support the interoperability with non Java systems (JAX-WS, JAX-RS).

- the **fourth tier** will be occupied by the Databases:
all the the data will be stored using Oracle DBMS and backup database servers. The application servers and the database will interact using the JDBC protocol.

Every tier will be isolated from the others with the use of a firewall. The application servers, that are the most sensitive components of the architecture, will be deployed inside a demilitarized zone.
In the next page is presented a deployment diagram that models the proposed architecture.



<<device>>
Third party laptop
<<Web Browser>>
<<protocol>>
HTTP
<<device>>
Firewall
<<protocol>>
HTTP
<<device>>
Web server
<<Web Browser>>
Apache HTTP 2.4
mod _jk
mod_proxy_balancer
<<protocol>>
ajp13
<<device>>
Data4Help application server
<<EJB server>>
GlassFish 5.0
<<stateless bean>>
<<statefull bean>>
<<protocol>>
JDBC
<<device>>
User mobile phone
<<Application>>
<<protocol>>
HTTP
<<device>>
Firewall
<<protocol>>
RMI
<<protocol>>
RMI
<<device>>
Firewall
<<protocol>>
JDBC
<<device>>
Database server
<<Database server>>
Oracle Database 18c
User Database
<<device>>
AutomatedSOS application server
<<EJB server>>
Glassfish 5.0
<<stateless bean>>
<<statefull bean>>
<<protocol>>
JDBC
<<device>>
User wearable
<<Application>>
<<protocol>>
HTTP
TIER 1
TIER 2
TIER 3
TIER 4

2.5 Runtime View

This section illustrates the phases of the main interactions that happen during the use of the system. Using several Sequence Diagrams, we illustrated the communications between the logic components.
Even though we tried to describe the processes in the most clear and detailed way, these diagram remains a high-level description of the interactions flows. Therefore, all of the parameters and methods that appear in these Sequence Diagrams may do not correspond exactly to the ones in the actual implementation of the components.
In the following Sequence Diagrams we cover all the possible interactions between each component. We grouped all the Sequence Diagram that belong to the same flow of activities, since for obvious reasons we had to decomposed them in sub-sequence to fit them into this document. The diagrams presented below cover all these situations:

- Registration of a user (which is identical to the registration of a third party, so we omit this last one) : Figure 2.6
- Login of a user (which is identical to the login of a third party, so we omit this last one) : Figure 2.7
- A third party makes an individual request :
  Figure 2.6, Figure 2.8, Figure 2.9, Figure 2.10, Figure 2.11, Figure 2.12
- A third party makes a group request : Figure 2.13
- AutomatedSOS analyzes health parameters and detects an anomaly:
  Figure 2.14, Figure 2.15, Figure 2.16

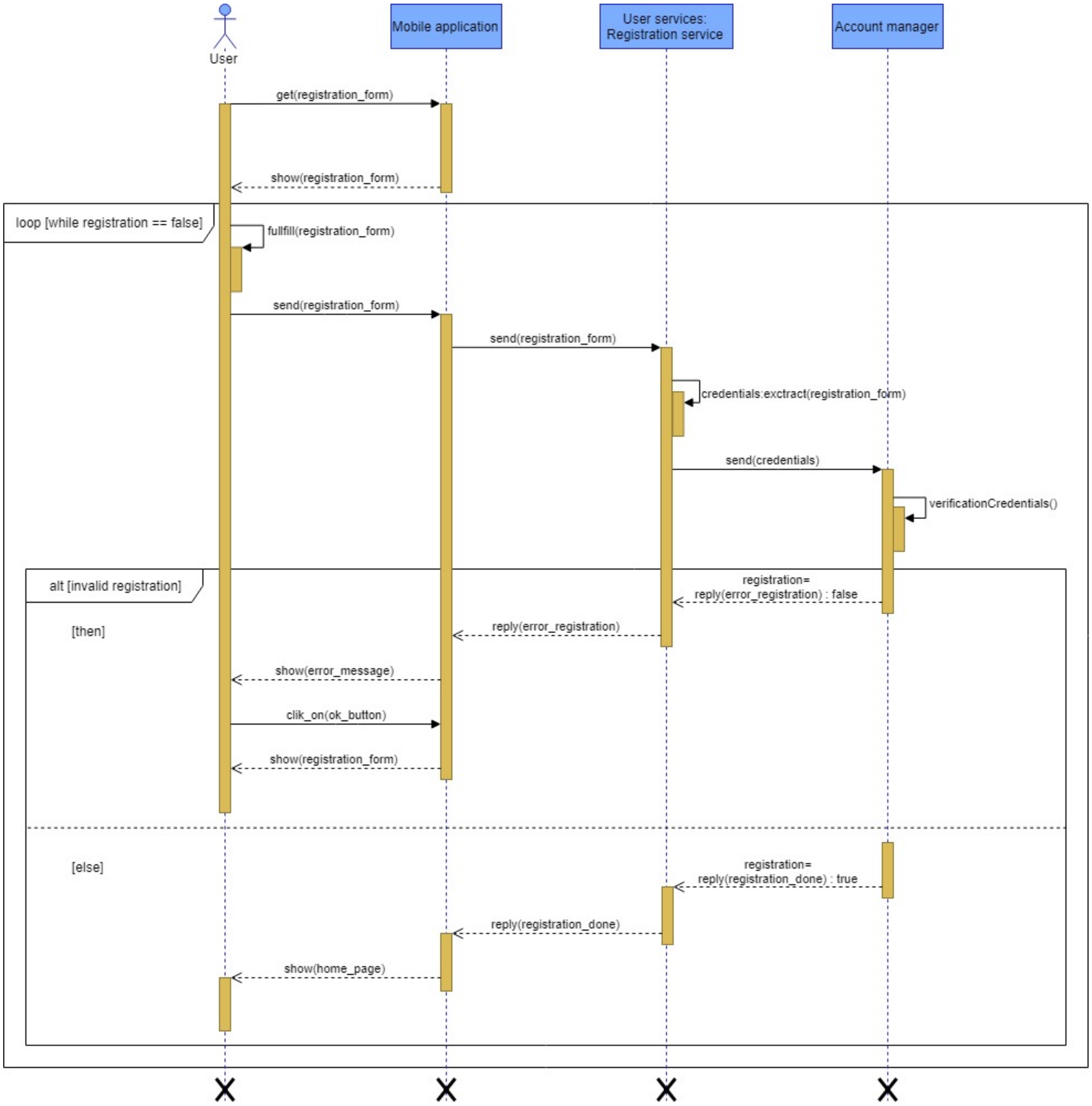


Figure 2.6: Data4Help - Registration of a user.

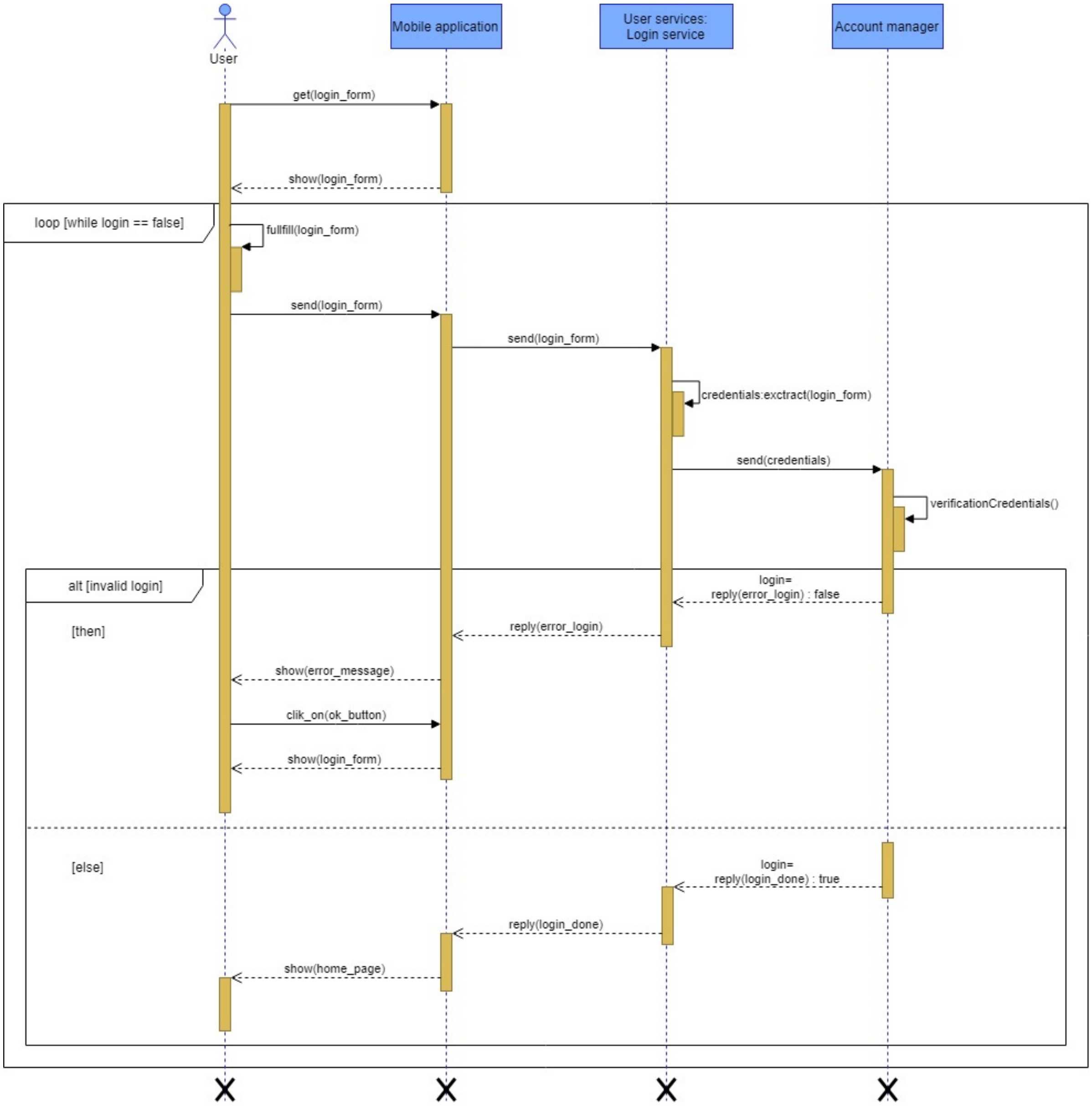


Figure 2.7: Data4Help - Login of a user.

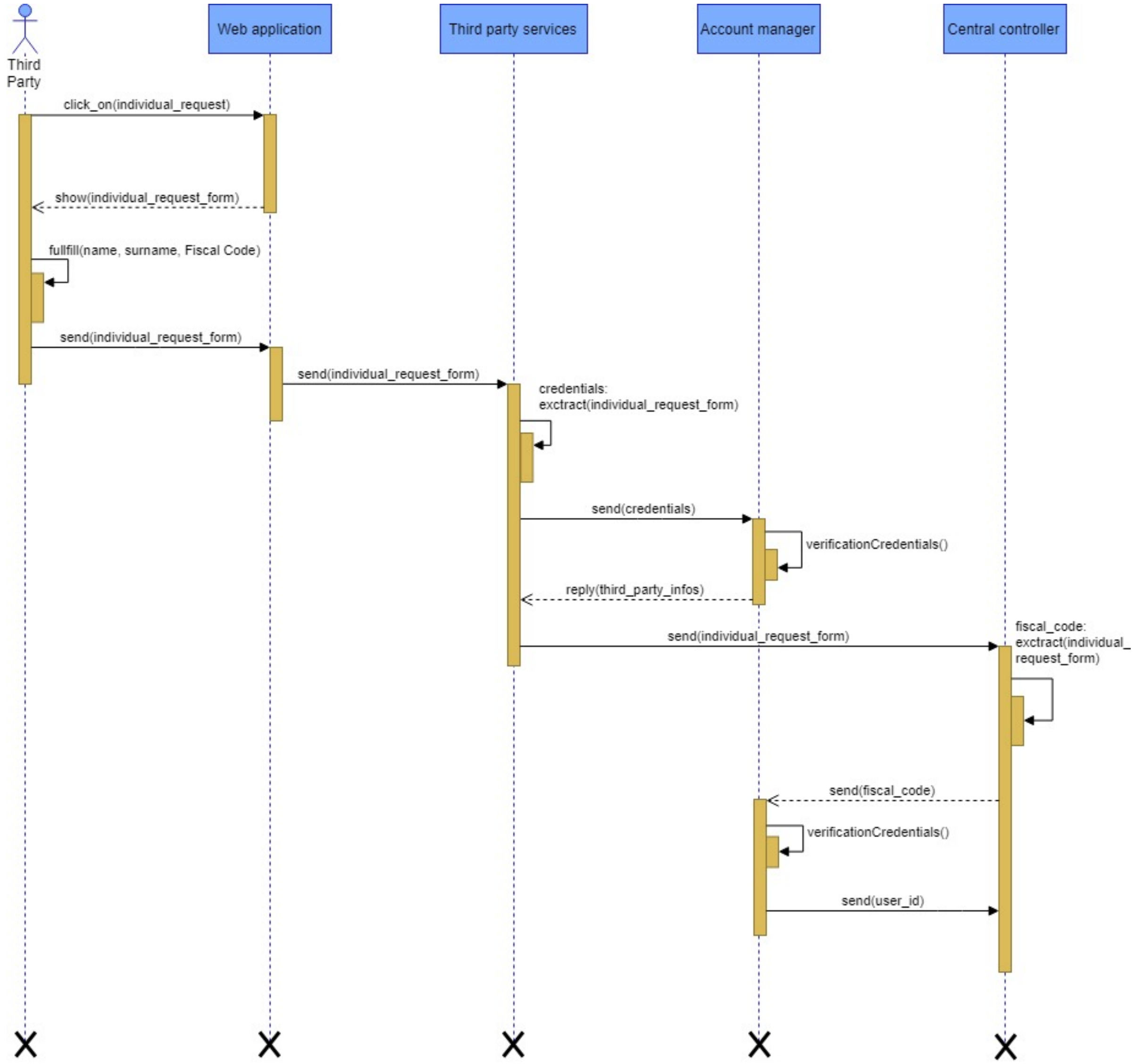

Third Party
Web application
Third party services
Account manager
Central controller
click_on(individual_request)
show(individual_request_form)
fullfill(name, surname, Fiscal Code)
send(individual_request_form)
send(individual_request_form)
credentials:
exctract(individual_request_form)
send(credentials)
verificationCredentials()
reply(third_party_infos)
send(individual_request_form)
fiscal_code:
exctract(individual_
request_form)
send(fiscal_code)
verificationCredentials()
send(user_id)

Figure 2.8: Individual request - A third party makes an individual request.

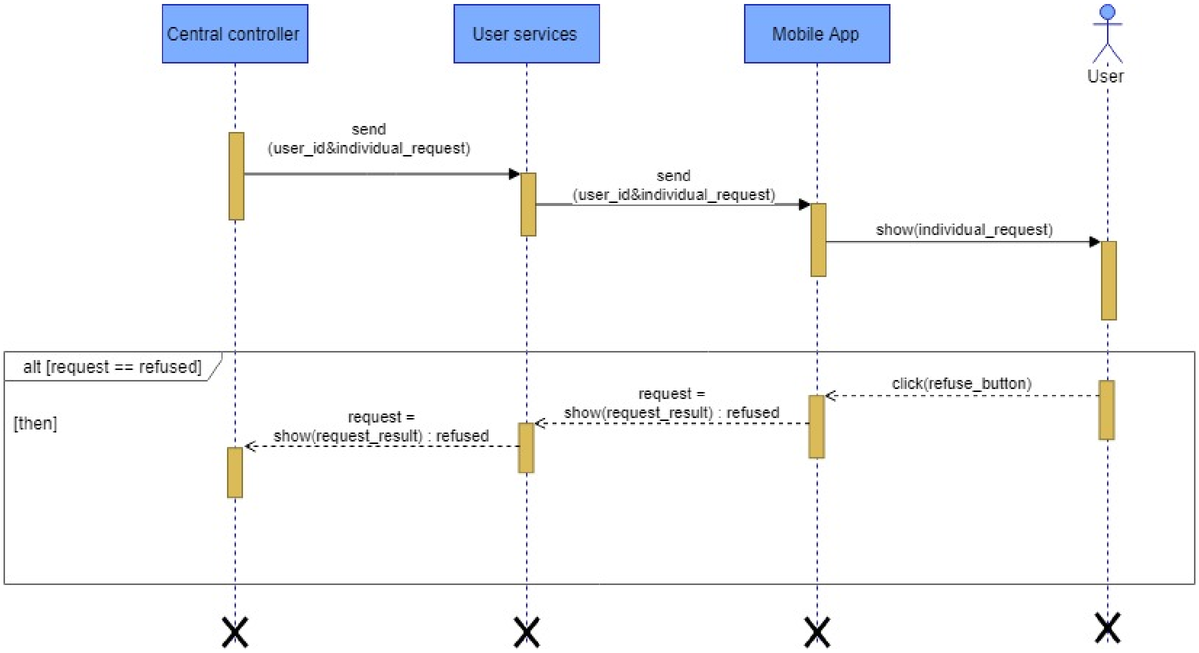


Figure 2.9: Individual request - The individual request is forwarded to the user and he/she refuses it.

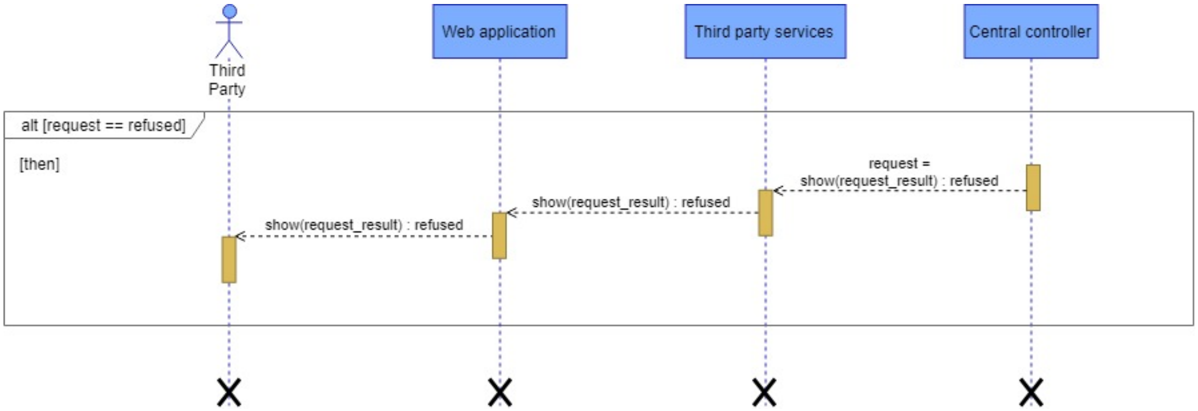


Figure 2.10: Individual request - The reply is sent to the third party.

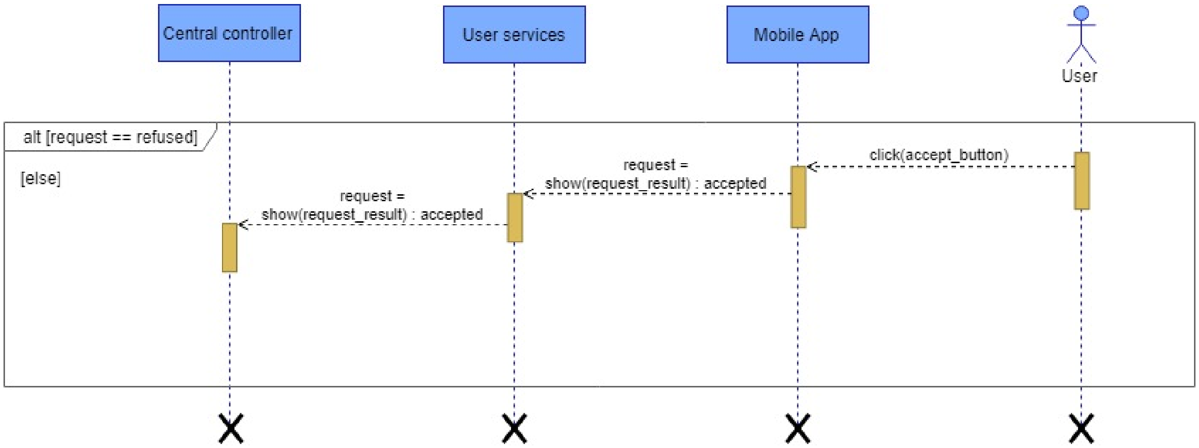


Figure 2.11: Individual request - The user accepts the request.

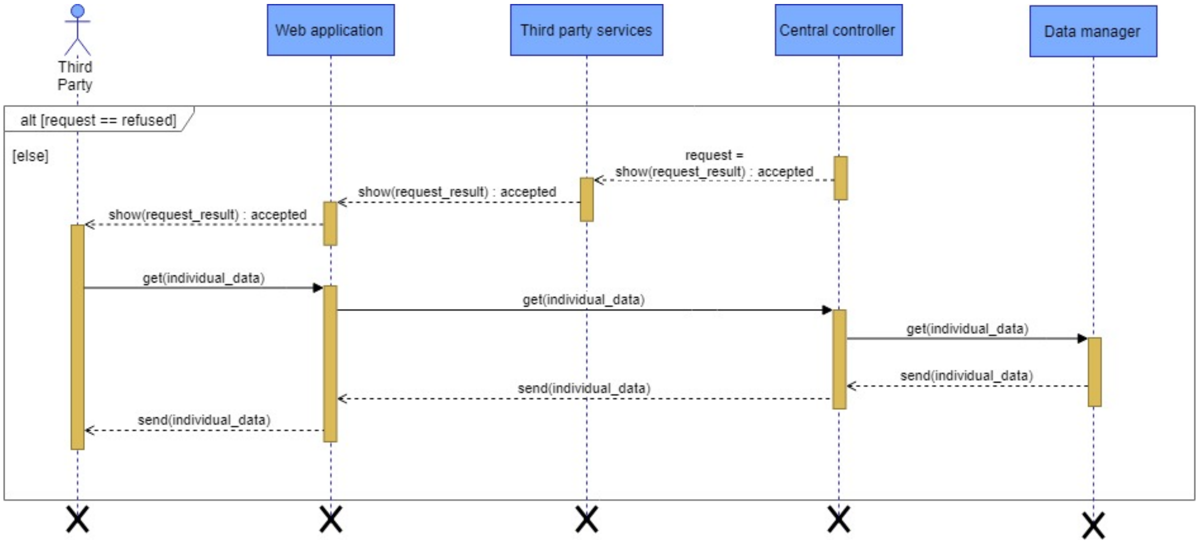


Figure 2.12: Individual request - The reply and the individual data are sent to the third party who requested them.

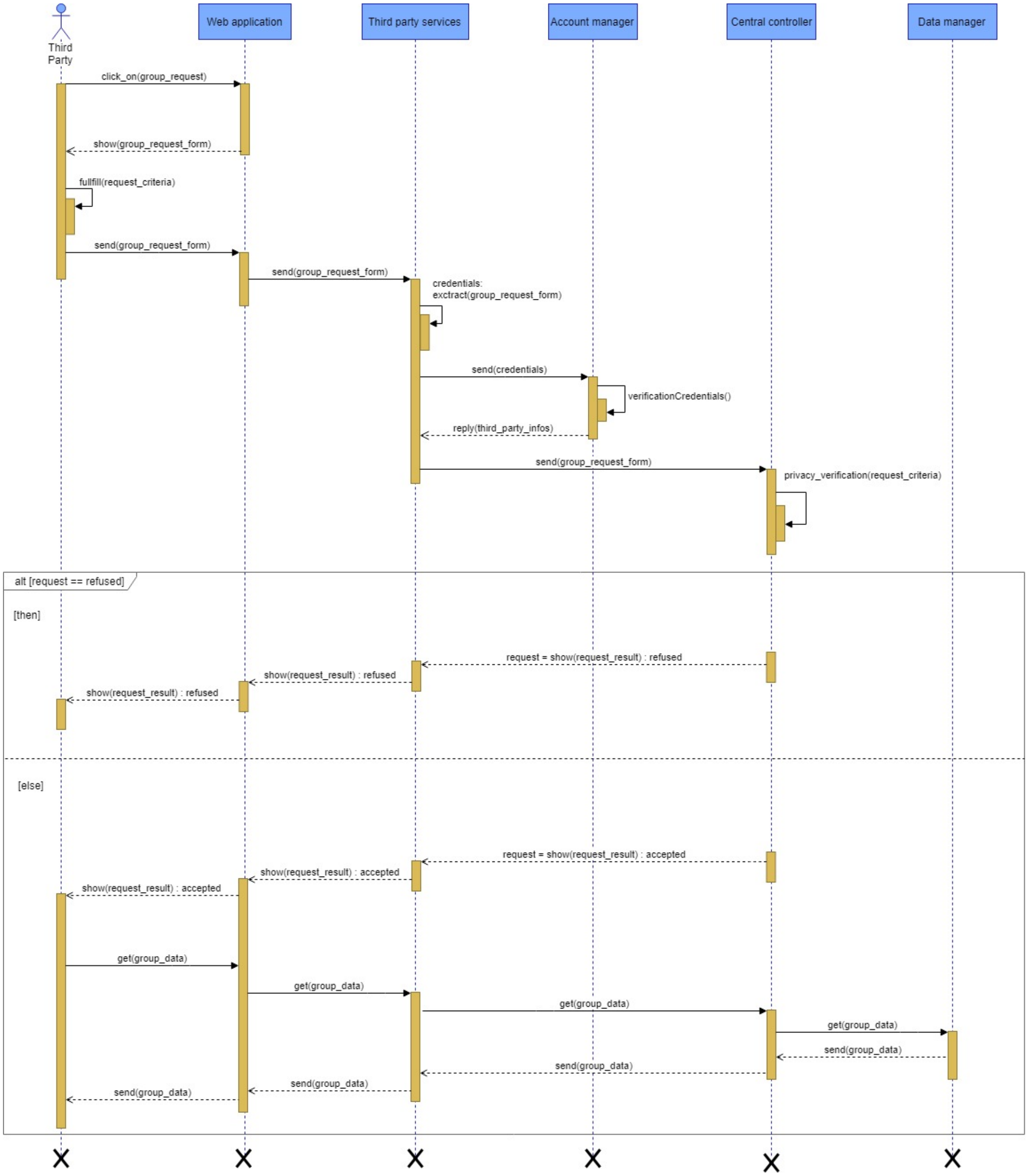


Figure 2.13: Data4Help - A third party makes a group request.

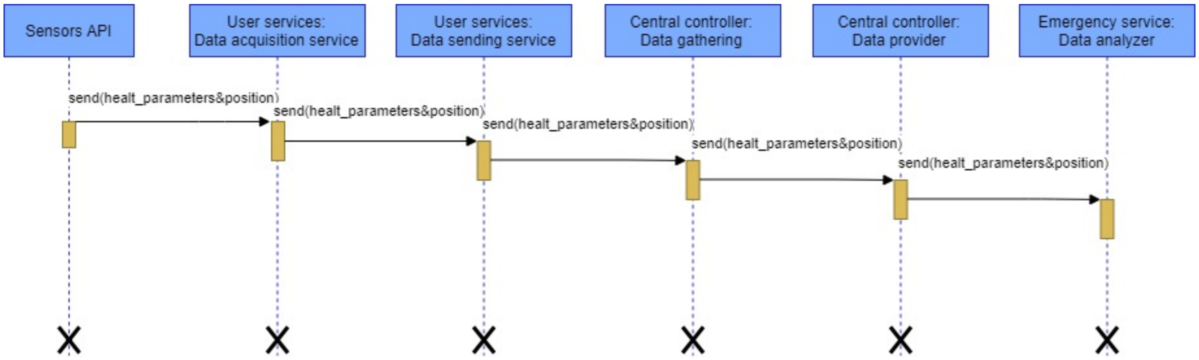

Sensors API
User services: Data acquisition service
User services: Data sending service
Central controller: Data gathering
Central controller: Data provider
Emergency service: Data analyzer
send(healt_parameters&position)
send(healt_parameters&position)
send(healt_parameters&position)
send(healt_parameters&position)
send(healt_parameters&position)

Figure 2.14: AutomatedSOS - Real-time data acquisition.

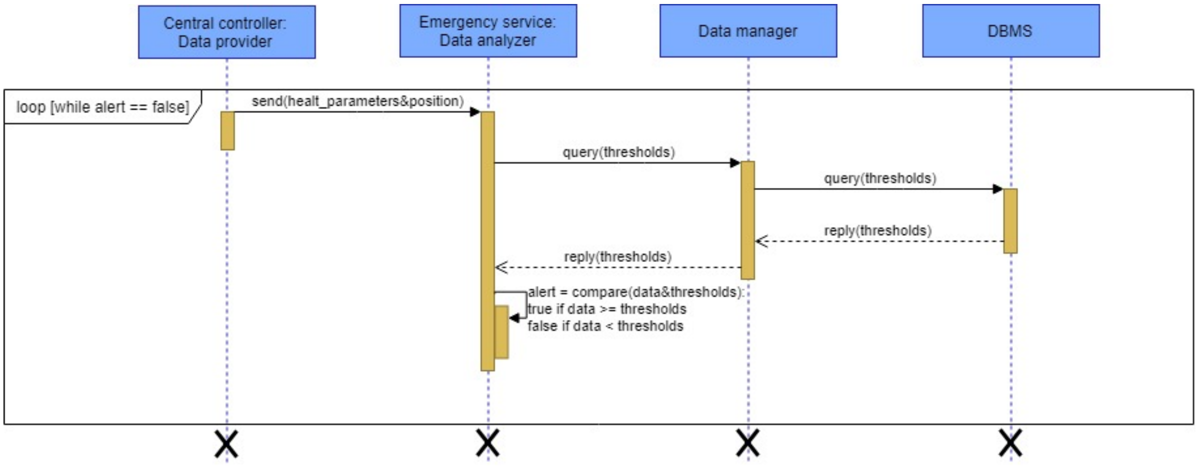

Central controller: Data provider
Emergency service: Data analyzer
Data manager
DBMS
loop [while alert == false]
send(healt_parameters&position)
query(thresholds)
query(thresholds)
reply(thresholds)
reply(thresholds)
alert = compare(data&thresholds):
true if data >= thresholds
false if data < thresholds

Figure 2.15: AutomatedSOS - Real-time data analysis.

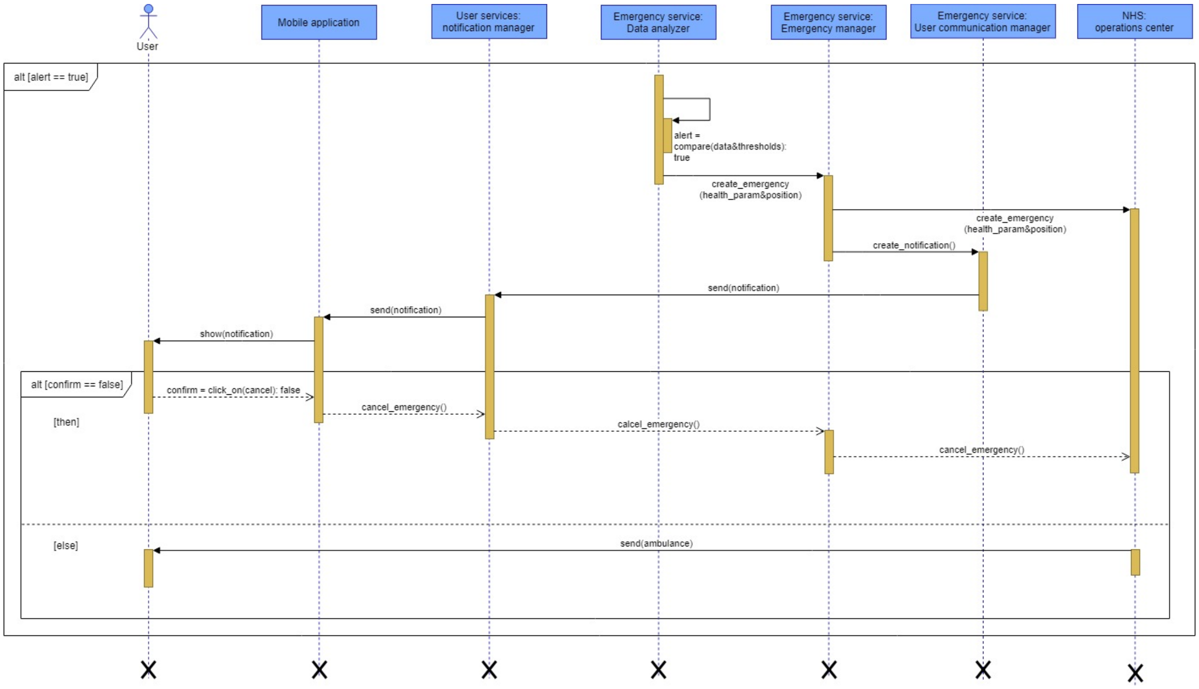


Figure 2.16: AutomatedSOS - An emergency is detected.

2.6 Component Interfaces

The diagrams showed below list the main methods that form the components interfaces.

Since the Central Controller behaves as an intermediary between users (collection of data) and third parties (transfer of these data), the component provides interfaces that serve the communication with both parties. In particular, the Individual Request Manager offers methods for the addition of a request by a third party and the response to that request by the user.

The Third Party and User services offer all the methods responsible for the interaction between the client applications and the back-end. Thanks to these components, all the different implementations of the presentation tier can interact with the same interface.

Examples of methods offered by the Third Party Services regards the sending of a request and the access to users' data. Examples of methods offered by the User Services regards the response to an incoming request and the data acquisition from the device.

The emergency service provides the methods to collect and analyze health data, generate and send emergency notification to the NHS and communicate with the user.

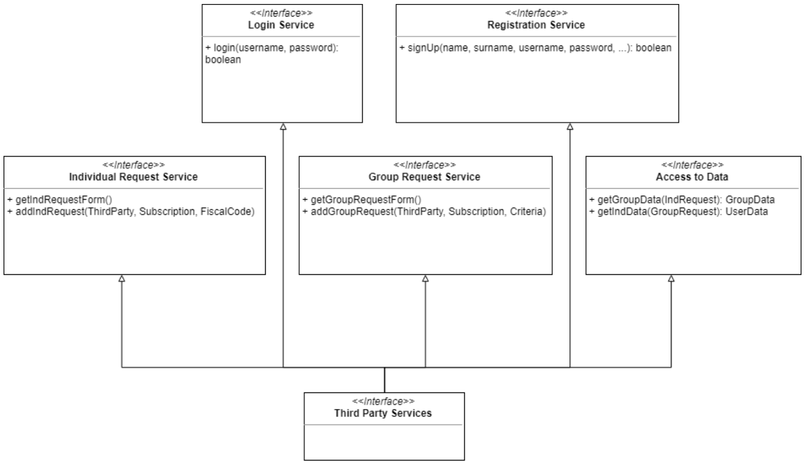

<<Interface>>
Login Service
+ login(username, password):
boolean
<<Interface>>
Registration Service
+ signUp(name, surname, username, password, ...): boolean
<<Interface>>
Individual Request Service
+ getIndRequestForm()
+ addIndRequest(ThirdParty, Subscription, FiscalCode)
<<Interface>>
Group Request Service
+ getGroupRequestForm()
+ addGroupRequest(ThirdParty, Subscription, Criteria)
<<Interface>>
Access to Data
+ getGroupData(IndRequest): GroupData
+ getIndData(GroupRequest): UserData
<<Interface>>
Third Party Services

Figure 2.17: Component Interface

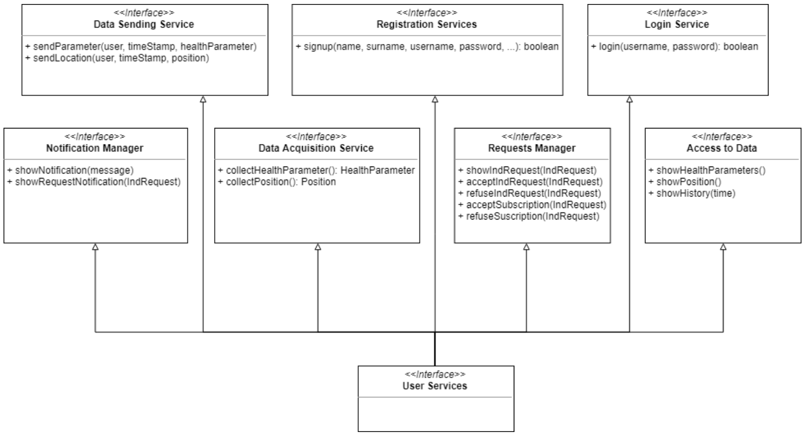

<<Interface>>
Data Sending Service
+ sendParameter(user, timeStamp, healthParameter)
+ sendLocation(user, timeStamp, position)
<<Interface>>
Registration Services
+ signup(name, surname, username, password, ...): boolean
<<Interface>>
Login Service
+ login(username, password): boolean
<<Interface>>
Notification Manager
+ showNotification(message)
+ showRequestNotification(IndRequest)
<<Interface>>
Data Acquisition Service
+ collectHealthParameter(): HealthParameter
+ collectPosition(): Position
<<Interface>>
Requests Manager
+ showIndRequest(IndRequest)
+ acceptIndRequest(IndRequest)
+ refuseIndRequest(IndRequest)
+ acceptSubscription(IndRequest)
+ refuseSuscription(IndRequest)
<<Interface>>
Access to Data
+ showHealthParameters()
+ showPosition()
+ showHistory(time)
<<Interface>>
User Services

Figure 2.18: Component Interface

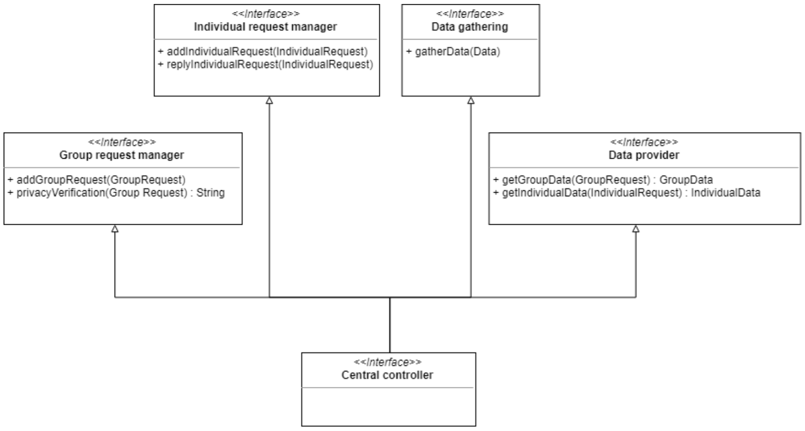

<<Interface>>
Individual request manager
+ addIndividualRequest(IndividualRequest)
+ replyIndividualRequest(IndividualRequest)
<<Interface>>
Data gathering
+ gatherData(Data)
<<Interface>>
Group request manager
+ addGroupRequest(GroupRequest)
+ privacyVerification(Group Request) : String
<<Interface>>
Data provider
+ getGroupData(GroupRequest) : GroupData
+ getIndividualData(IndividualRequest) : IndividualData
<<Interface>>
Central controller

Figure 2.19: Component Interface

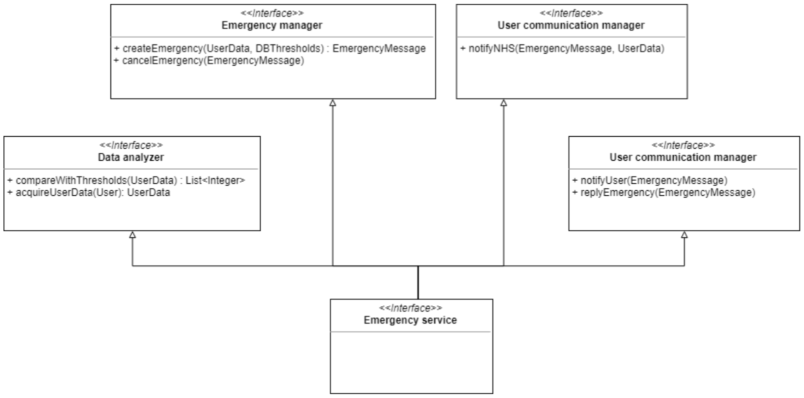

<<Interface>>
Emergency manager
+ createEmergency(UserData, DBThresholds) : EmergencyMessage
+ cancelEmergency(EmergencyMessage)
<<Interface>>
User communication manager
+ notifyNHS(EmergencyMessage, UserData)
<<Interface>>
Data analyzer
+ compareWithThresholds(UserData) : List<Integer>
+ acquireUserData(User): UserData
<<Interface>>
User communication manager
+ notifyUser(EmergencyMessage)
+ replyEmergency(EmergencyMessage)
<<Interface>>
Emergency service

Figure 2.20: Component Interface

2.7 Selected Architectural Styles and Patterns

- **Model-View-Controller:** it's a very common design pattern, that separates the architecture into three communicating components. The Controller performs actions on the Model, using inputs from the View. The Model contains the application data structures and the View gives an external representation of them.
  The MVC pattern promotes the code reuse and avoids the coupling between data and their representation. This pattern is particularly appropriate for complex systems such as Data4Help and AutomatedSOS, that will be deployed in multiple physical nodes and will provide several types of views to their users.

- **Service Oriented Architecture:** we designed the system components as a set of services that interacts with each other through a communication protocol. Viewing each component as a service gives the advantages of abstracting its functions to the highest level and see the module as a independent black box.
  A SOA enhances the modularity and maintainability of the system. It also promotes the creation of robust and complete interfaces for each component.

- **Proxy Pattern and Facade Pattern:** we used these patterns mainly for the interactions between the back-end and the client applications. The proxy pattern provides an intermediate object that substitute for the real subject of a communication, while controlling that the input is valid and the access is appropriate. The facade pattern implements a simple interface for the access of the back-end system and also acts as an adapter for the information exchanges in the messages. We used these patterns to create an interface between clients and the central system that could be simple to access, secure and that could minimize the dependencies between modules.

- **Publish/Subscribe Pattern:** it is closely related to the MVC pattern. This architectural style offers great scalability and it's fundamental to ensure satisfying performances in an event-based system. This pattern allows to send users' data as soon as they are collected and to notify an emergency as soon as it's detected.

- **Factory pattern (software design pattern):** a factory object provides a public method for creating other objects, while hiding the actual creation process. The factory pattern promotes the overall encapsulation in the code base.

- **Strategy pattern (software design pattern):** allows the creation of a set of encapsulated algorithms and the selection of one of them at run-time. This pattern is one of the most common in OOP and promotes encapsulation and polymorphism.

- **Adapter pattern (software design pattern):** used to adapt two different interfaces. This pattern is particularly useful for the integration between AutomatedSOS and the NHS of the hosting country. A more detailed description of its usage will be given in the next paragraph.

2.8 Other Design Decisions

- **Google Maps API:** to represent current positions and daily routes we will use the maps API offered by Google. This API is the common standard because of its quality and current updates. Google Maps offers an interface for JavaScript[4], Android[2] and iOS[3] development.
- **Sensors API and healthKit:** to collect data from users' device we will exploit the APIs offered by Android and iOS platforms: Sensors API[5] and healthKit[1].
- **AutomatedSOS and Data4Help integration:** when designing AutomatedSOS, we followed two main guide lines: promoting the code reuse as much as possible and keeping the central logic of the application independent from Data4Help's components. Given these objectives, we designed AutomatedSOS as an independent service, built on-top of Data4Help platform. The two apps share the same account management system, so whenever a user registers to AutomatedSOS he/she also becomes a Data4Help user.
  AutomatedSOS core module (Emergency Service) exploits Data4Help's Data Provider to obtain users' data in real time. Placing the logic of the two systems in distinct modules favors a distributed deployment of the back-end and a correct load balance. In terms of reliability, a failure in AutomatedSOS does not affect Data4Help operations (the vice versa is not true, for obvious reasons).
- **Interaction with the NHS:** AutomatedSOS will be hopefully adopted in several countries, that have significantly different National Health Institutions, First Aid services and unit of measurement for the health parameters.
  An efficient integration between the system and the local NHS is a fundamental requirement for the success of an application that deals with its users' lives.
  The existence of many differences between the countries forces the design to be adapted to each specific NHS. In order to do so, the NHS Adapter is the interface component responsible for the translation between the system and the local Operations Center. To give an high-level description: if a country offers an API to send a digital message to the Operations Center, the NHS Adapter will connect to it; if such an API does not exists, the NHS adapter will call 911 (112 in Italy) telephone number and read the emergency message to the operator with an artificial voice.

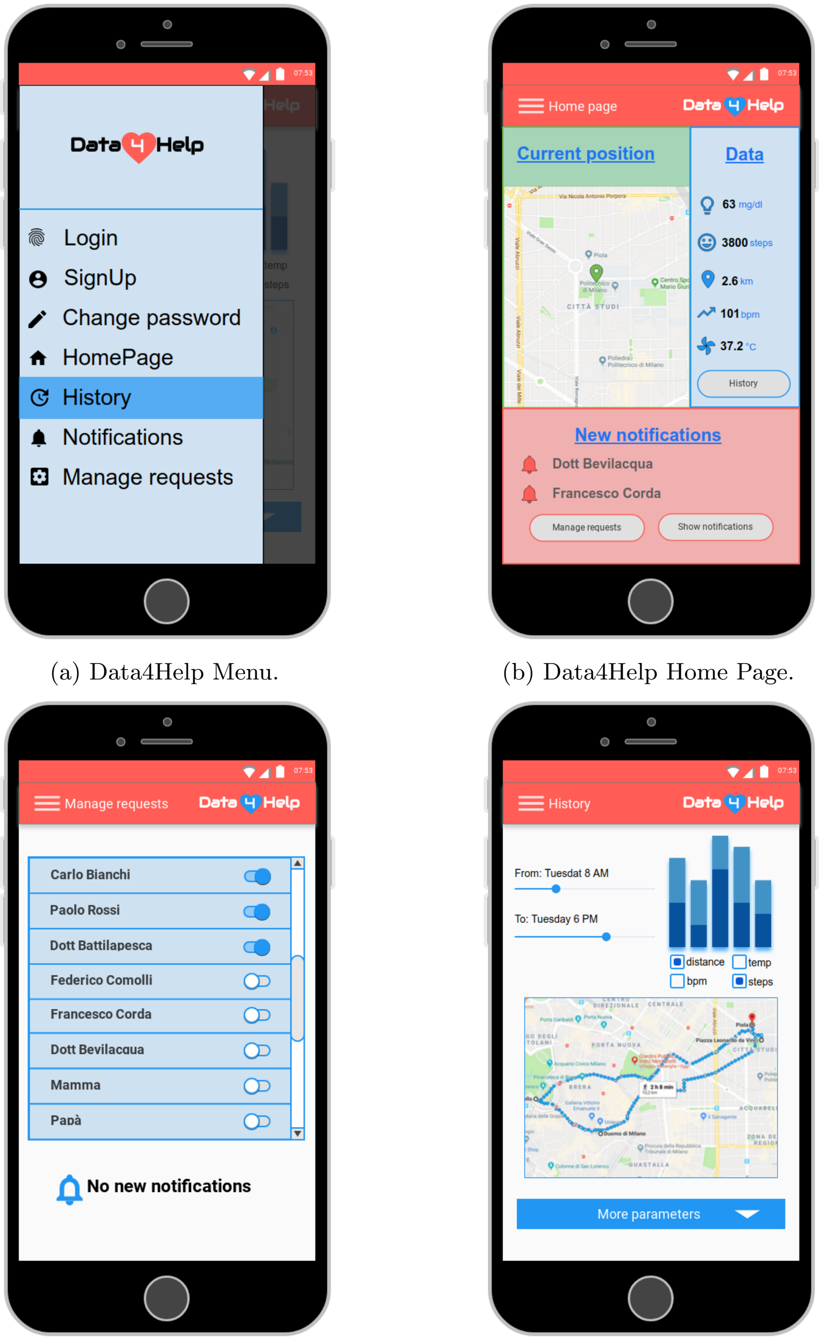
Section 3

User Interface Design

3.1 Mock-up Interfaces

We think that the success of one application strongly depends on his user-friendly graphic design. According to this we propose, both in the RASD and in the DD documents, some mockups trying to be as intuitive and satisfactory as possible.
We also want to offer the possibility to run the two applications on a large number of devices. In order to facilitate the diffusion of the Data4Help and AutomatedSOS applications, we want them to run properly on different hardware platforms: personal computers, mobile phones and also wearable devices. In the RASD document we have already presented some UIs that accomplish this multi-platform requirement. Moreover, we designed our application with the purpose to be resizable, so that it can adapt to screens of different dimension.
In the RASD document are illustrated 12 different mockups that represent different parts of Data4Help and AutomatedSOS applications. Here we want to introduce 4 new mockups, to increment and complete the overview of the graphic appearance of the two applications.
In particular, in this document, we focused on:

- **Menu:** from this page, the user can reach all the pages of Data4Help application by clicking on the respective item.
- **Home page:** here the user can see his real-time position on a map(based on Google Maps) and his real-time health parameters. Moreover he/she can see the notifications for the new requests.
- **Manage requests:** in this page the user can accept or decline the requests for his/her data and he/she can interrupt the subscriptions to updated data previously accepted.
- **History:** by selecting a time interval, the user can see the average value of his/her health parameters and he/she can also observe his historic route.



(a) Data4Help Menu.

(b) Data4Help Home Page.

(c) User manages his requests.

(d) User observes his historical data.

3.2 User Experience Diagrams

We have already presented how the different screens of our application will look like, showing some comprehensive mockups in the section above and in the RASD document. However they are in some way "static", and just by observing all the presented interfaces it's not possible to understand how they interact with each other. For this purpose, here are presented three different User Experience diagrams with the intention to cover all the possible applications of the system (web, wearable or mobile). The diagrams presented here and in the following pages are preceded by a short introduction.

The first one describes the User experience with the mobile application (and also partly with the wearable application). The first screen that User has to face is the Login page. From here the user can reach the Registration page or, if it has already an account, the Home page. From this last page, he/she finally can reach the Manage Requests, the Accept/Decline Requests or the History screen.

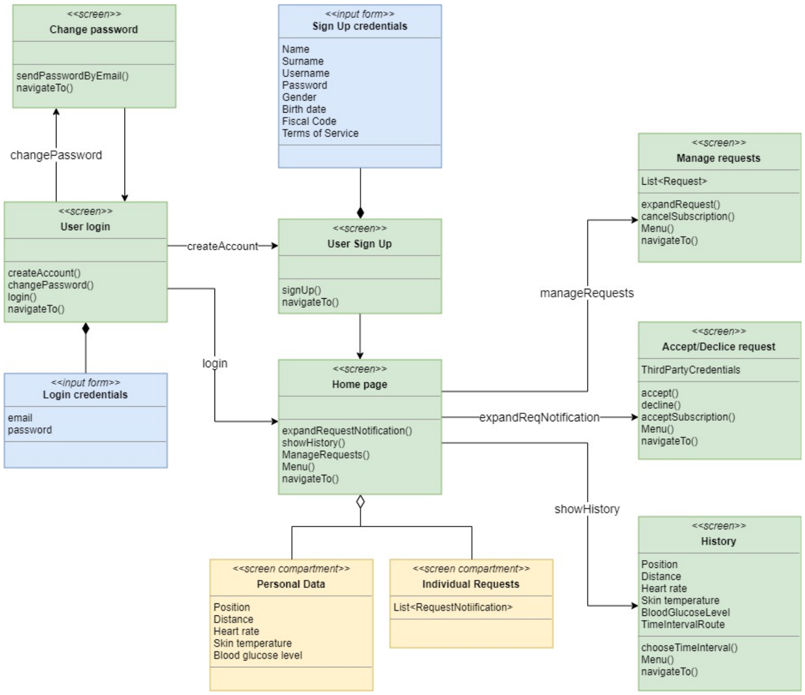


Figure 3.2: Data4Help - User experience

The second diagram describes the experience of the third party through the mobile application(or maybe also through the Web application). The first page that he/she has to face is obviously the login screen, from which he/she can reach the Registration or the Home pages. Finally, from this last screen the third party can reach the Create Group Request, Create Individual Request and the Requested Data pages.

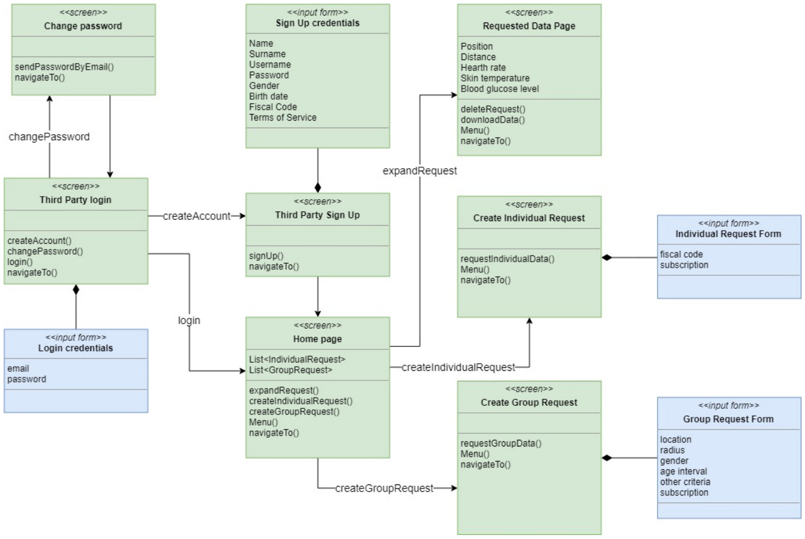


Figure 3.3: Data4Help - third party experience

This last diagram illustrates the experience of a User through the AutomatedSOS mobile application (and also partly through the wearable application). As in the other UXs, the first page that the user has to face is the Login page, followed by the Registration or Home pages. The last available page of this application is the Emergency screen, that appears instead of the home page when an emergency is detected.

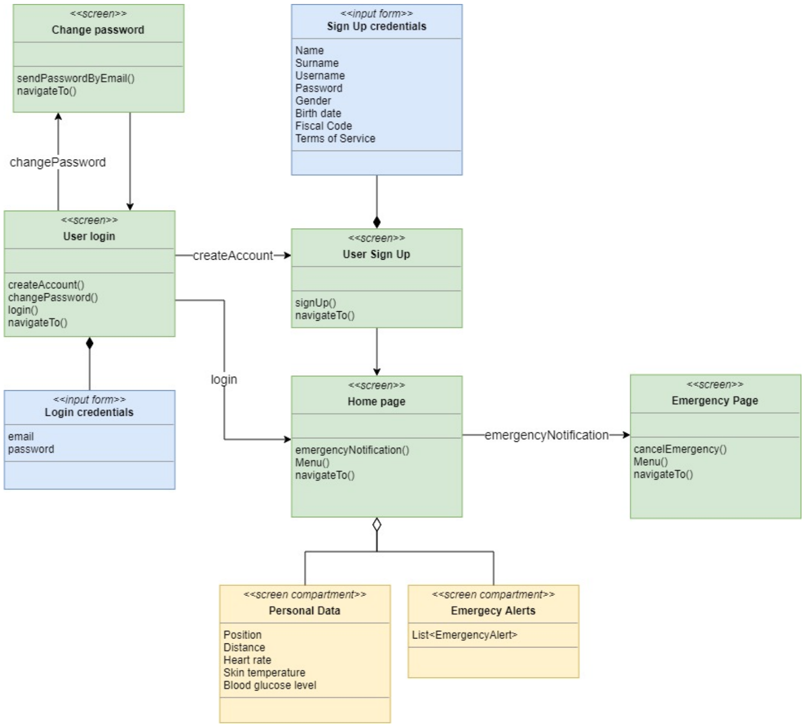


Figure 3.4: AutomatedSOS - User experience

3.3 Boundary Control Entity Diagrams

In the last section of the chapter focused on the User Interfaces we want to reach a further level of detail.
We showed all the available pages of our applications-to-be and how the user and the third party can navigate through them.
Here we want to illustrate how the interactions of a User/Third Party with respective applications are managed by the "Controller" and how they affect the "Model". In the following diagrams we refer to the Model-View-Controller design pattern, which guarantees the correct separation between the presentation of the application and its core logic.
The entities on the left are intended as part of the View of our applications-to be, the entities on the right are part of the Model and, for last, the ones at the center represent the Controller.

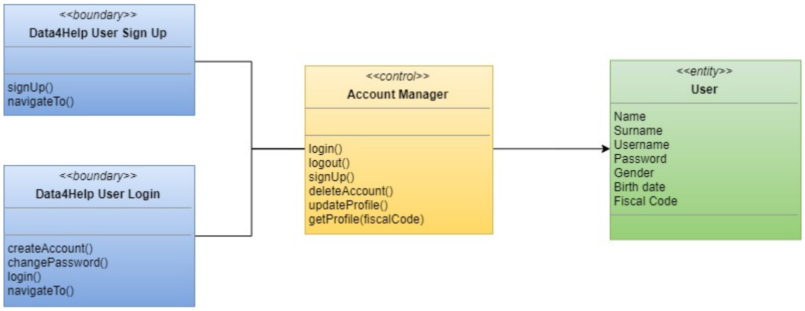


Figure 3.5: User registration and login to Data4Help application.

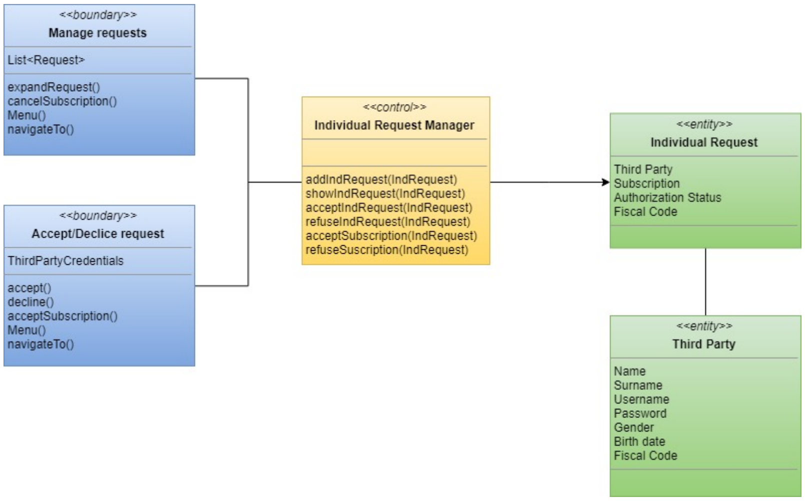


Figure 3.6: The user responds to a request and manages previous requests.

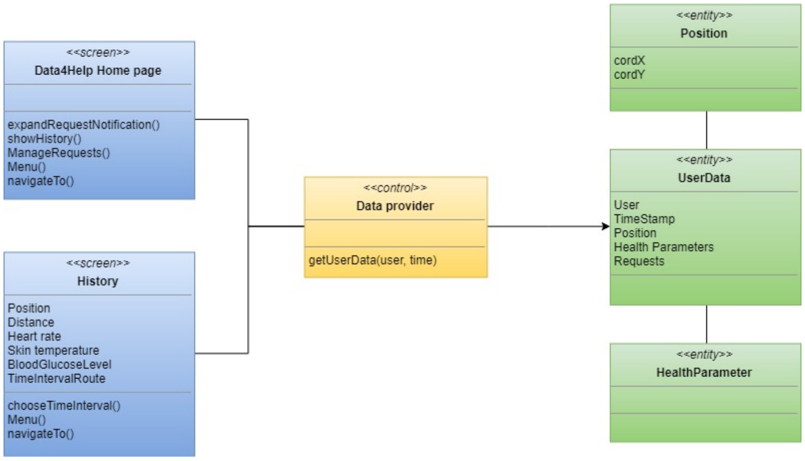


Figure 3.7: The user accesses current or historical personal data.

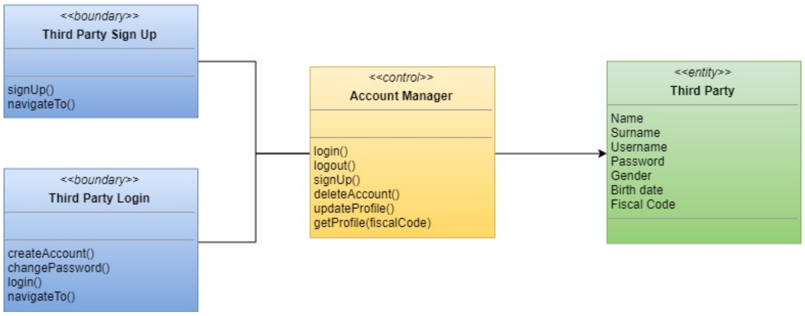


Figure 3.8: Third party registration and login to the Data4Help application.

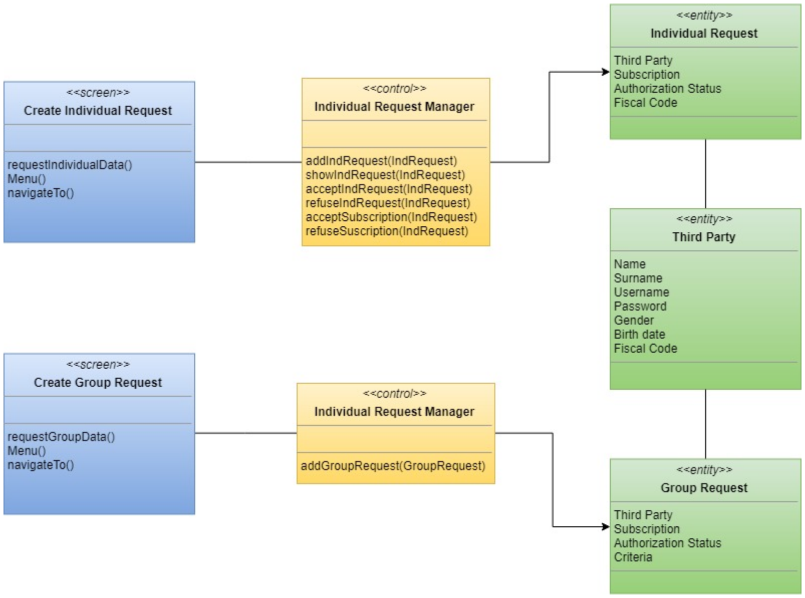


Figure 3.9: A third party creates an individual or group request.

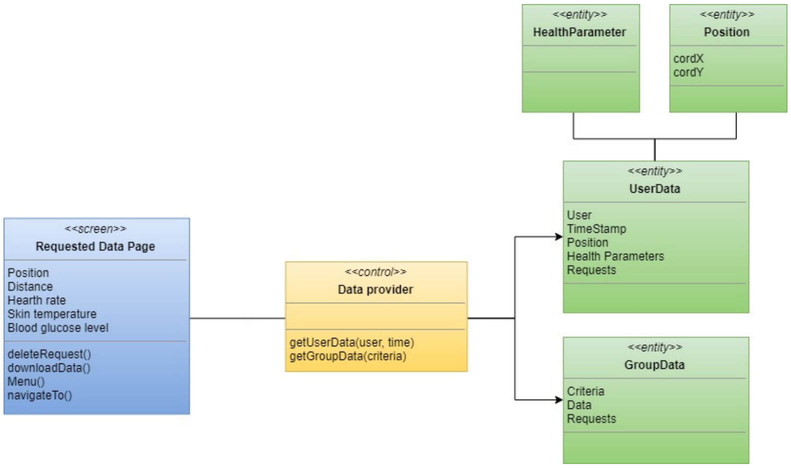


Figure 3.10: A third party obtains the requested data.

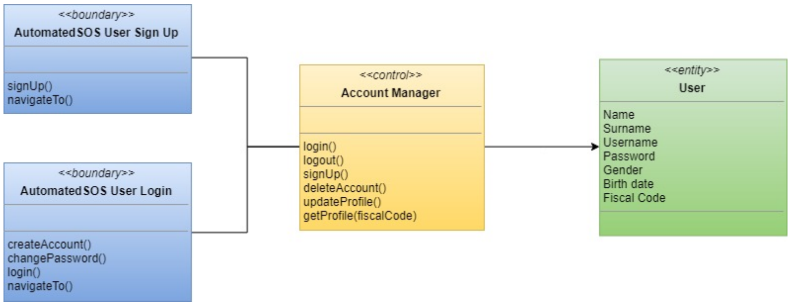


Figure 3.11: User registration and login to AutomatedSOS application.

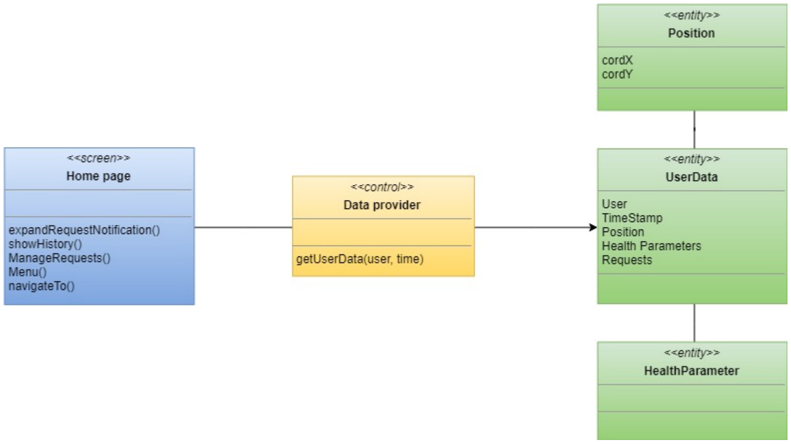


Figure 3.12: A user accesses current personal data.

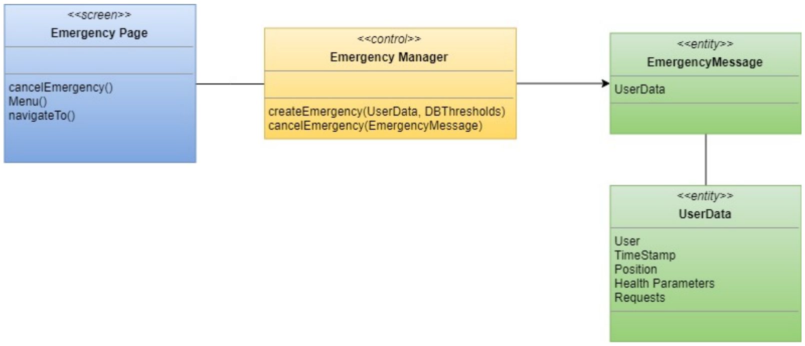


Figure 3.13: Emergency Page activated in case of a health crisis.

Section 4

Requirements Traceability

4.1 Functional Requirements

The design decisions illustrated in this document have been made with the purpose of satisfying all the functional requirements listed in the RASD document. In this section, each requirement is mapped to the corresponding design components. The mapping can be better understood by observing the Runtime view, section 2.5, in which is illustrated the sequence of the interactions between the design components, in order to accomplish the functional requirements.
Other important diagrams that integrate the following mapping, are in the Component view, section 2.3.

[R1] The system should provide a registration form offering the following mandatory fields: name, surname, SSN or Fiscal Code.

- Mobile application
- User services: Login service
- Account manager

[R2] The system must guarantee that the credentials are unique.

- Account manager

[R3] The system must acquire real-time users' positions from the GPS installed on the user's device.

- Sensors API (GPS API)
- User services: Data acquisition service
- User services: Data sending service
- Central controller: Data gathering

[R4] The system must collect users' data in specific databases.

- Central controller: Data gathering
- Central controller: Data provider
- Data manager
- DBMS

[R5] The system must acquire real-time users' health parameters from the sensors installed on the user's device.

- Sensors API
- User services: Data acquisition service
- User services: Data sending service
- Central controller: Data gathering

[R6] Every time that a third party inserts a request for data relative to a Fiscal Code, the system must ask the authorization to the corresponding user.

- Central Controller: Individual Request Manager
- User services: Request manager
- Mobile application

[R7] The system must provide a function to allow users to accept or refuse a request for personal data.

- Mobile application
- User services: Requests manager

[R8] The system should provide a registration form to third parties.

- Web application
- Mobile application
- Third party services: Registration service

[R9] The system must provide a function to allow third parties to request the access to individuals data. The form must require the filling of the SSN or Fiscal Code of the corresponding user.

- Web application
- Mobile application
- Third party services: Individual request service
- Account manager
- Central controller: Individual request manager

[R10] The system must provide a function to allow third parties to request the access to data of a group of people. The form must offer some selection criteria such as geographical, time, health parameters, movement habits, ecc.

- Web application
- Mobile application
- Third party services: Group request service
- Account manager
- Central controller: Group request manager

[R11] As soon as a request has been approved, the system must forward the most recent data to the third party applying for them.

- Web application
- Mobile application
- Third party services: Individual request service
- Third party services: Group request service
- Third party services: Access to data
- Central controller: Data provider
- Central controller: Data gathering
- Data Manager
- DBMS

[R12] Data4Help must provide a function to let third parties download the received data.

- Web application
- Mobile application
- Third party services: Access to data

[R13] The system accepts a request for anonymous data only if the group composing the data is formed by more than 1000 people.

- Central controller: Group request manager

[R14] The system must provide a function that allows to customize the requests for subscription to updated data. The requests must include how long the permission is still valid.

- Mobile application
- Web application
- Third party services: Individual request service
- Third party services: Group request service

[R15] The system must provide a function to the corresponding user that allows him/her to accept or refuse the subscription request.

- User services: Request manager
- Mobile application

[R16] The system must forward in real-time the most recent data to the third parties that subscribed to them.

- Web application
- Mobile application
- Third party services: Individual request service
- Third party services: Group request service
- Third party services: Access to data
- Central controller: Data provider
- Central controller: Data gathering
- Data Manager
- DBMS

4.2 Quality Requirements

To respond to a huge number of concurrent transactions we used the load balancing technique. Data provided by the users and their access by several third parties could represent a huge amount of transactions that the system must be able to correctly handle.
The security of the system is reached through at least four different mechanisms: the proxy, the Account Manager, the encryption of the data and the firewalls. The proxy is able to detect DDoS attacks and, as a consequence, can disable a user that is overloading the servers. This attacks are also prevented by the Account Manager, which guarantees that only the authorized users can interact with the servers. All the databases are encrypted to prevent attacks that aims to steal sensitive data from the system. Moreover, one of the purpose of the firewalls is to limit the number of ports that are exposed to the public.
The availability and the reliability of the system are ensured by the n-tier infrastructure. The system is based on several databases distributed on different data centers. The data are stored at least into two different databases so that when a server falls down, there is at least one available copy of them.
The maintainability of the system is guaranteed by the Model View Controller design pattern and by the Facade pattern.
In order to ensure the portability of the core service, the database and the back-end system are developed avoiding platform dependent programming languages. For this purpose the system is based on the Service Oriented Architecture.

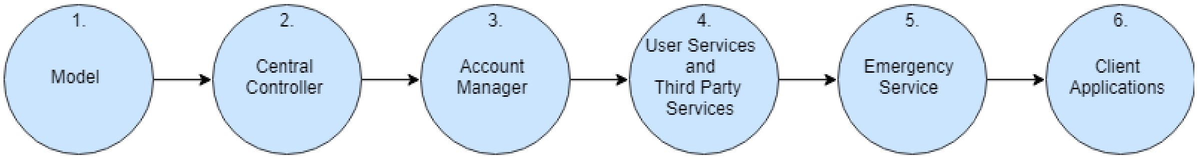
Section 5

Implementation, Integration and Test Plan

5.1 Implementation Plan

The implementation of Data4Help and AutomatedSOS systems is carried on taking into consideration some important factors, such as the complexity of each component and its relations with the others. Of course, the implementation plan takes into special consideration the core components which are essential for the whole system and are considered as "bottleneck". It is important to fix any flaw as soon as it is detected, so that the correction cost in terms of time is kept the lowest possible. The order in which our system is implemented is the following:

1. Model
2. Central Controller
3. Account Manager
4. Third party services, User Services
5. Emergency Service
6. Web, Mobile and Wearable Applications



Model

The Model is the first step of the implementation plan, according to its vital role for the whole application. Since the classes of the Model are used by all the components, it is important that they are ready as soon as possible to be used as a reference. Otherwise each team member would implement the classes of the Model "on the fly" when he/she needs them, introducing possible implementation flaws or not using abstraction in a proper way.

Central Controller

This component is the logic core of the application. According to this, it requires lot of time to be implemented and, consequently, it is a risky component. It is important that the least possible flaws are inserted during its implementation, since they may have catastrophic consequences in terms of time costs.
An important non functional requirement for the Central Controller is the fault-tolerance and the high throughput. This component may be stressed by lots of requests from users and third party, so it has to react in a short time. For this reason, this component is consider as a "bottleneck".

Account Manager

This module allows the sign-up and login of the users and third parties and is also used by the Central Controller when a third party composes a request for data of a specific user. As shown in the Runtime view, section 2.5, the third party fills the request form specifying the user's SSN or Fiscal Code and, thanks to the Account Manager, the Central Controller converts it into the corresponding user ID. This module is used by different kind of clients and, consequently, it has to use abstraction to interact with all of them.

Third party services, User Services

These two components allow the interaction between the user application (mobile or wearable) and the third party application (mobile or web), passing through the Central Controller. They can be implemented together, since they are quite similar to each other. As shown in the Component view, section 2.5, the sub-components that allows the login, the registration and the access to data are present in both modules. Regarding to the Individual and the Group requests, the Third party services component offers the possibility to create them while the User services component allow the user to manage them. According to what just said, they are implemented just after the Central Controller and the Account Manager.

Emergency Service

This component has a vital role for the AutomatedSOS system. It analyzes real-time health parameters by comparing them with specific thresholds. It is strictly dependent on the User services component and on the Central Controller, so it has to be implemented after they are ready to use. Of course, the Emergency Service has to be fault tolerant since it deals with human lives. In case of failure of this component, the error must be repaired as soon as possible since the system must result available 24/7. This component must guarantee an high throughput since it receives a large amount of real-time health data from all the users at the same time.

Web, Mobile and Wearable Applications

The remaining elements of the system to be implemented are the client applications. They allow users and third parties to interact with the system, but they don't contain core logic functions of the system. They represent only a minimum part of the code and they require a short time to be implemented. The web apps must be compatible with all the major browsers, while the mobile apps must be developed for Android and iOS platforms. The applications must properly run on different devices both for the users (smart-phones and wearable-devices) and the third parties (personal computers, tablets and smart-phones).
For these reasons, client components are not considered "risky" and they are taken into account as the last part of the implementation plan.

5.2 Integration and Testing

5.2.1 Entry Criteria

Integration testing is an important process that has to start as soon as every component is almost complete, in order to promptly fix any flaw that could eventually occur.
The purpose of this process is to test the integration between the components of our system, according to all the functionalities listed in the RASD and the DD documents that our application will be able to offer.
To start the Integration each component has to correctly pass its unit test. All the services provided by external entities, including each APIs (Google Maps API and the APIs of the health sensors), have to be completed and they have to properly work since our system will heavily depends on them. For the same reason, also the DBMS and the Data Manager service must be completed before starting all tests.
Regarding the AutomatedSOS system-to-be, we have already explained how the interaction between the system and the National Health Services can change according the host country. In the nations in which the NHS provides an API, the latter must be completed before starting the integration tests. Otherwise, there must be an operative VoIP API that allows AutomatedSOS to interact with the NHS.

The integration process between two components should start when both of them have reached a minimum level of completion, in order to make the integration tests meaningful.
To be detailed, the estimated completion of each component to enter the integration process should be:

- Data Manager, **100% of completion**
- Central Controller, **90% of completion**
- Account Manager, **70% of completion**
- Third Party Services, **60% of completion**
- User Service, **60% of completion**
- Emergency Services, **80% of completion**
- Third Party Client Applications, **40% of completion**
- User Client Applications, **40% of completion**

In the bulleted list above, the percentages reflect the minimum number of functions that each component must offer in order to enter the integration process.

5.2.2 Elements to be Integrated

As already mentioned in the Architectural Patterns section, the system is based on a Service Oriented Architecture, which guarantees the modularity and the maintainability of the software. Each component offers some interfaces to communicate with the others. It's important to test the integration of the modules to ensure the reliability of the system.
Moreover, each requirement listed in the RASD document is achieved through the numerous interactions between the interfaces of all the components.
We identify four different macro-categories in which components can be grouped, according to their role inside the system. Referring to the high-level components in the Figure 2.2, these categories are:

- Client components: third party web application, user mobile application, user wearable application.
- Logic components: Third party services, user services, account manager, central controller, emergency services.
- Data components: data manager, DBMS.
- External components: Maps API, VoIP API, Sensors API.

This separation facilitates us in the process of the integration. The Logic category represents the core of the application, so the majority of the interactions takes place inside this group. On the contrary, there aren't interactions between components inside the Client categories.
In the following lines we list all the elements that need to be integrated together, belonging to the same or to different categories.

Integration of Logic Components with Data and External components:

- Central Controller
  - Data Provider, Data Manager
  - Data Gatherer, Data Manager
- User Services
  - Data Acquisition Manager, Sensors API
  - Access to Data, Maps API
- Third Party Services
  - Group Request Service, Maps API
  - Access to Data, Maps API

- Emergency Service
  - Data Analyzer, Data Manager
  - NHS Adapter, NHS API

Integration between Logic Components:

- Central Controller
  - Individual Request Manager, Requests Manager (User Services)
  - Individual Request Manager, Notification Manager (User Services)
- User Services
  - Data Sending Service, Data Gathering (Central Controller)
  - Login Service, Account Manager
  - Registration Service, Account Manager
- Third Party Services
  - Individual Request Service, Individual Request Manager (Central Controller)
  - Group Request Service, Group Request Manager (Central Controller)
  - Access to Data, Data Provider (Central Controller)
  - Login Service, Account Manager
  - Registration Service, Account Manager
- Emergency Service
  - Data Analyzer, Data Provider (Central Controller)
  - Emergency Manager, Account Manager (Central Controller)
  - User Communication Manager, Notification Manager (User Services)

Integration of Client Components with Logic Components:

- User Web, Mobile and Wearable Applications, User Services
- Third Party Web and Mobile Applications, Third Party Services

5.2.3 Integration Testing Strategy

According to the implementation plan stated before, we decided to integrate the components using a mixture of two strategies: the **Bottom Up** and the **Critical Module First**.
The **Bottom Up** approach allows us to test the ready components directly in a real world scenario, avoiding the creation of stubs. It's important that the external services are available for the integration process from the beginning.
As soon as a component or a part of it has been implemented, it can be integrated with the already existing ones which lies in the lower levels of the hierarchy. Using this method we are able to find out the behavior of the system in the real world usage, and we can promptly intervene if something goes wrong. Also, we chose this integration approach to optimize the efficiency of the development, since it reflects the same approach used during the implementation phase. Components that lies in the same hierarchy level can be integrated in an arbitrary order, maybe integrating the ones that are ready first.
The Logic components represent the core of the entire system, so it is necessary to start integrating them from the beginning. For this reason we also decided to use the **Critical Module First** approach to start integrating the riskiest components first. Since a fault in these components may have tragic consequences for the entire system, this approach allows us to fix the bugs earlier.
Components provided by external entities, such as **Maps API**, **health sensors APIs**, **NHS API** and **VoIP API** can be immediately used in the integration since we assume that they are ready before all the others. Also the **Data Manager** and the **DBMS** components have to be completed before starting the integration, since the lower level of the hierarchy strongly depends on them.

5.2.4 Sequence of Component/Function Integration

This section illustrates the integration of components in our systems. The following diagrams represent the order that will be followed during the process. An arrow going from component X to component Y means that Y is necessary for X functioning, so it must have already been implemented before the integration happens.

Software Integration Sequence

Following a bottom-up approach, the integration process will start by connecting the submodules of the system. In the first step we will focus on the integration of logic components with data and external components, then we will proceed with the integration of subcomponents in the central logic.

Integration of Logic Components with Data Components: the first step in the sequence will concern the integration of the back-end submodules with the data management system. We chose to start from here because of the data-centric nature of the applications. As it's possible to see in the diagrams below, several submodules rely on a correct communication with the Data Manager.

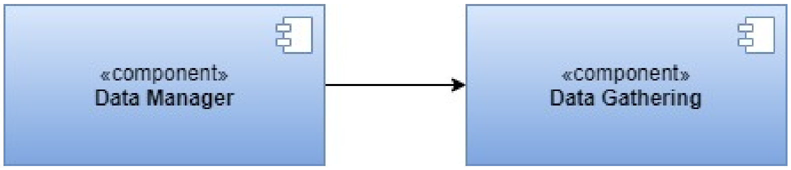


Figure 5.1: Integration of the Data Gathering and the Data Manager.

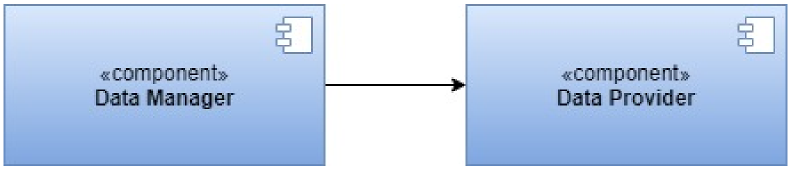


Figure 5.2: Integration of the Data Provider and the Data Manager.

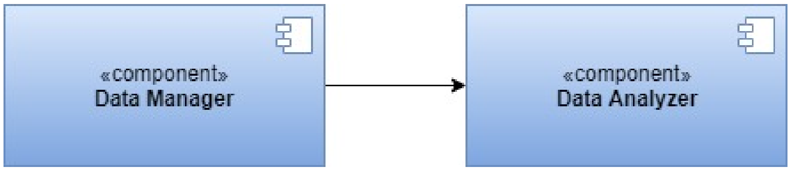


Figure 5.3: Integration of the Data Analyzer and the Data Manager.

Integration of Logic Components with External Components: the second step will be devoted to the integration with APIs provided by external entities. These components often play an essential role in the success of the system and it's important to test their functioning as soon as possible, in order to be able to react to eventual faults.

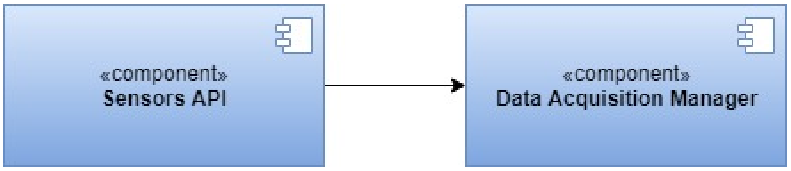


Figure 5.4: Integration of the Data Acquisition Manager and the Sensors API.

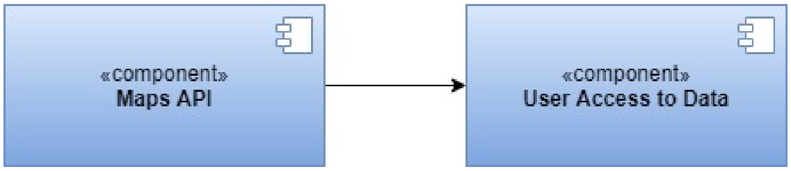


Figure 5.5: Integration of the User Access to Data and the Maps API.

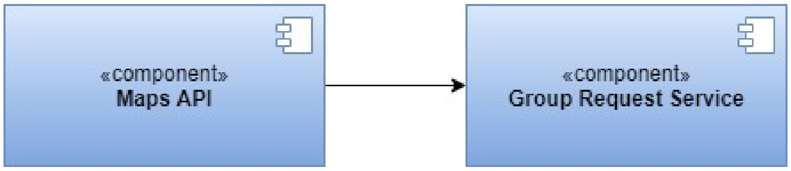


Figure 5.6: Integration of the Group Request Service and the Maps API.

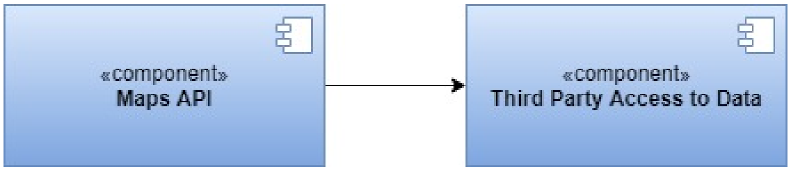


Figure 5.7: Integration of the Third Party Access to Data and the Maps API.

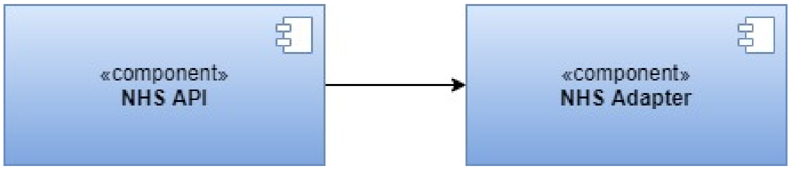


Figure 5.8: Integration of the NHS Adapter and the NHS API.

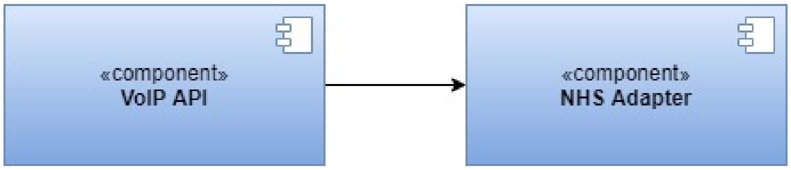


Figure 5.9: Integration of the NHS Adapter and the VoIP API.

Integration between Logic Components: the third step will be devoted to the integration of all the subcomponents in the back-end. This phase will be the longest and most complex of the process, given the great number of interactions that happens between the logic components.
For these reasons, the sequence of integration will follow a Critical Module First approach: initially, we will integrate the subcomponents regarding the collection and the exchange of data between the parties; then we will move on to connect the modules that deal with the account credentials and the dispatch of notifications.

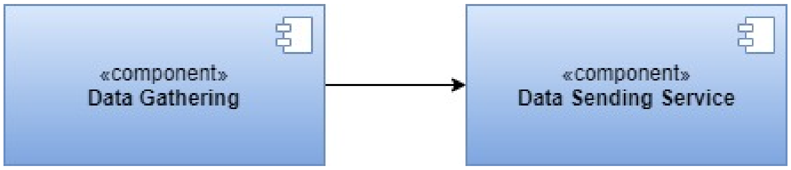


Figure 5.10: Integration of the Data Sending Service and the Data Gathering.

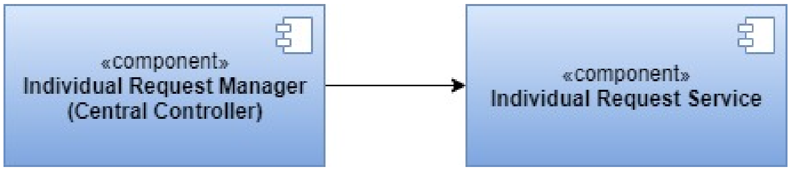


Figure 5.11: Integration of the Individual Request Service and the Individual Request Manager.

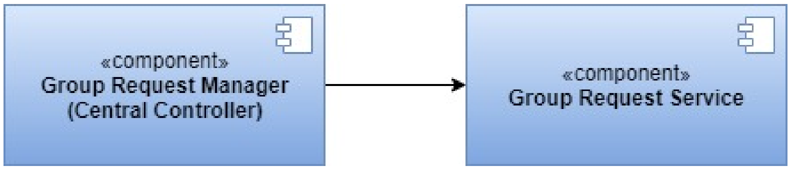


Figure 5.12: Integration of the Group Request Service and the Group Request Manager.

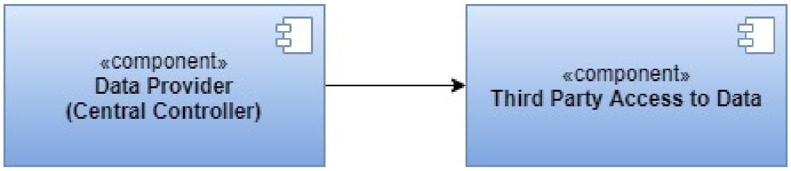


Figure 5.13: Integration of the Third Party Access to Data and the Data Provider.

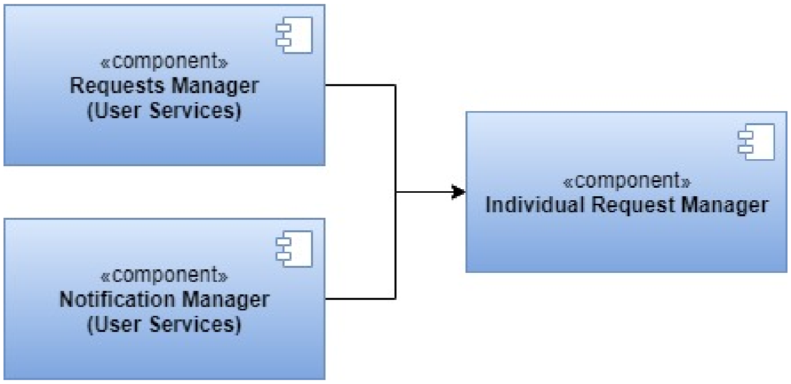


Figure 5.14: Integration of the Individual Request Manager and the Requests and Notification Managers

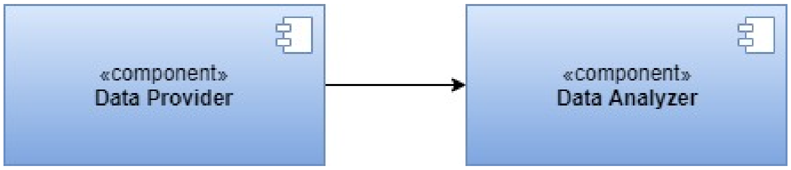


Figure 5.15: Integration of the Data Analyzer and the Data Provider.

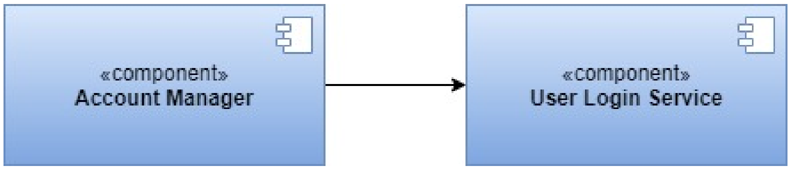


Figure 5.16: Integration of the User Login Service and the Account Manager.

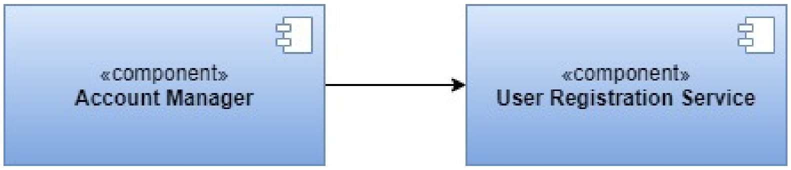


Figure 5.17: Integration of the User Registration Service and the Account Manager.

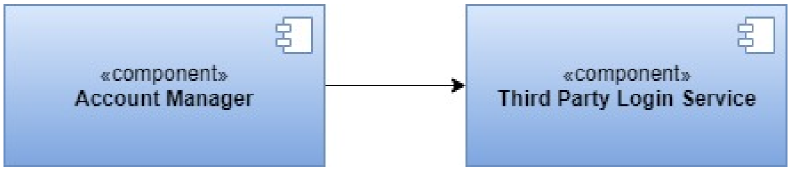


Figure 5.18: Integration of the Third Party Login Service and the Account Manager.

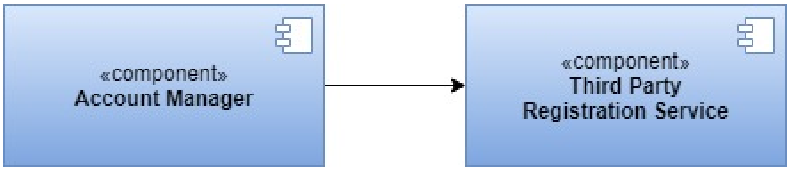


Figure 5.19: Integration of the Third Party Registration Service and the Account Manager.

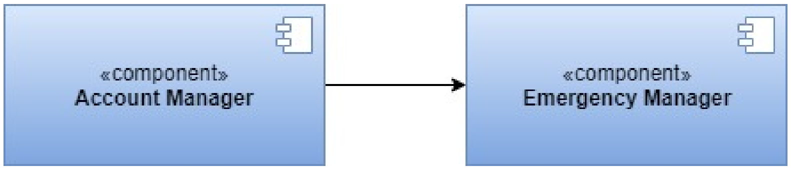


Figure 5.20: Integration of the Emergency Manager and the Account Manager.

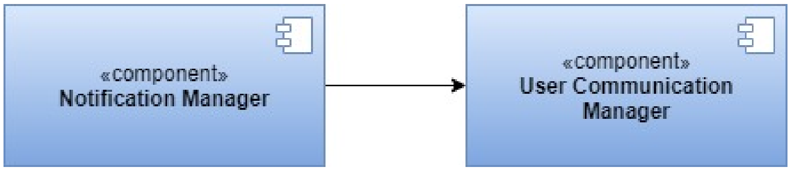
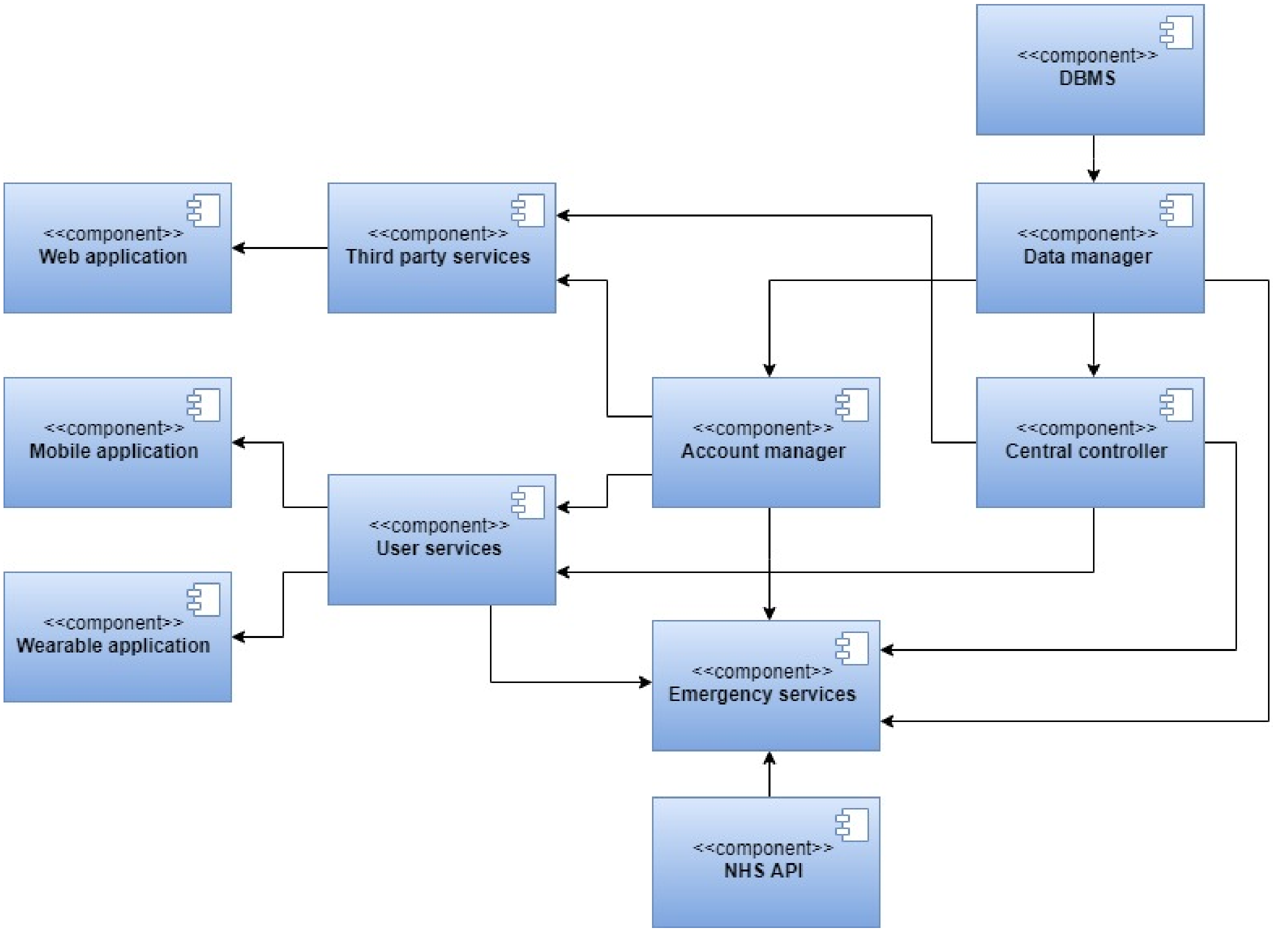


Figure 5.21: Integration of the User Communication Manager and the Notification Manager.

Subsystem Integration Sequence

Once all the submodules will be connected, we will proceed with the integration of the high level components, resulting in the final and complete system. In the final phase we will integrate the back-end system to the various instances of the presentation layer: web, mobile and wearable applications.
The following diagram gives an overview of the dependencies between the high level components, highlighting the sequence of integration that composes the system from the data layer to the client applications.



Section 6

Effort Spent

6.1 Comolli Federico

Description of the task	Date	Hours
Overview diagram	**23/11/2018**	2
High Level Components	**24/11/2018**	2
Component view	**25/11/2018**	3.5
Deployment view	**25/11/2018**	2
Runtime view	**27/11/2018**	3.5
Component interfaces	**29/11/2018**	2
UX and BCE diagrams	**01/12/2018**	3.5
Requirements traceability	**02/12/2018**	2
Section 2 review	**03/12/2018**	2
Implementation plan	**04/12/2018**	2
UX and BCE sections	**05/12/2018**	2.5
Integration and Testing	**06/12/2018**	4
Integration diagrams	**07/12/2018**	2
Introduction	**08/12/2018**	1.5
Review	**09/12/2018**	4.5
Review	**10/12/2018**	2
Total Effort Spent		41

6.2 Corda Francesco

Description of the task	**Date**	**Hours**
Overview diagram	**23/11/2018**	2
High Level Components	**24/11/2018**	2
Component view	**25/11/2018**	3.5
Deployment view	**25/11/2018**	2
Runtime view	**27/11/2018**	3.5
Component interfaces	**29/11/2018**	2
UX and BCE diagrams	**01/12/2018**	3.5
Requirements traceability	**02/12/2018**	2
Section 2 review	**03/12/2018**	2
Implementation plan	**04/12/2018**	2
Architectural Styles and Patterns	**05/12/2018**	2.5
Integration and Testing	**06/12/2018**	4
Integration diagrams	**07/12/2018**	2
Introduction	**08/12/2018**	1.5
Review	**09/12/2018**	4.5
Review	**10/12/2018**	2
Total Effort Spent		41

Section 7

References

[1] Apple Inc. healthkit. Technical report, Apple Inc., 2018.

[2] Google Inc. Google maps api android. Technical report, Google Inc., 2018.

[3] Google Inc. Google maps api ios. Technical report, Google Inc., 2018.

[4] Google Inc. Google maps api javascript. Technical report, Google Inc., 2018.

[5] Google Inc. Sensors api. Technical report, Google Inc., 2018.

[6] Oracle Inc. Glassfish jee implementation. Technical report, Oracle Inc., 2018.

[7] Oracle Inc. Java enterprise edition. Technical report, Oracle Inc., 2018.