



Data4Help  
Software Engineering II

## **Design Document**

*Eugenio Cortesi, Chiara Criscuolo*

Document version: 1.1  
December 10, 2018

# Contents

<b>1 Introduction</b>	<b>4</b>
1.1 Purpose	4
1.2 Scope	4
1.3 Definition, Acronyms, Abbreviations	5
1.3.1 Definitions	5
1.3.2 Acronyms	5
1.3.3 Abbreviations	6
1.4 Revision History	7
1.5 Reference Document	7
1.6 Document Structure	7
<b>2 Architectural Design</b>	<b>8</b>
2.1 High-level Components and Their Interaction	8
2.2 Component View	11
2.2.1 Relational Database	11
2.2.2 NoSQL Database	11
2.2.3 Application Server	13
2.2.4 Web Server	16
2.2.5 Mobile Application	16
2.3 Deployment View	18
2.4 Runtime View	20
2.5 Component Interfaces	27
2.5.1 Database - Application Server	27
2.5.2 Web Server - Web Browser	27
2.5.3 Application Server - Web Server and Mobile Application	27
2.5.4 Application Server - External System	27

2.5.5 Internal Interfaces for Application Server Components . . . . .	28
2.6 Selected Architectural Styles and Patterns . . . . .	34
<b>3 User Interfaces Design . . . . .</b>	<b>36</b>
<b>4 Requirements Traceability . . . . .</b>	<b>47</b>
<b>5 Implementation, Integration and Test Plan . . . . .</b>	<b>49</b>
5.1 Implementation Plan and Integration Strategy . . . . .	49
5.2 Test Plan . . . . .	51
5.2.1 Unit Tests . . . . .	51
5.2.2 Integration Tests . . . . .	53
<b>6 Effort . . . . .</b>	<b>56</b>
<b>Change-log . . . . .</b>	<b>57</b>

# 1 Introduction

## 1.1 Purpose

The design document aims to provide a deeper functional description of the system-to-be by giving the technical details and describing the main architectural components, their interface and their relationship between each other. The interactions are showed using UML diagrams and other similar ones to guide the development team to implement the architecture of the system identifying each component and its internal structure and highlighting all its relationships, defining how they work.

Two sections are also dedicated to the user interfaces and the implementation, integration and testing process.

## 1.2 Scope

Data4Help, as stated in the RASD document, is software-based service developed by TrackMe company that sends data from subscribed users to third parties that requested it. So it's necessary the registration to the third parties and individuals and the managing of this big data in a database. Users can see respectively their queries and answers or their health track.

AutomatedSOS is an additional service based on Data4Help that monitors the vital parameters of subscribed elderly people in order to start an emergency procedure in case of health problem.

The Data4Help system is composed by four main components:

- Two thin clients: Data4Help Mobile Application (for individual users) and Data4Help Web Application (for third companies);
- One Application Server;
- One data layer divided in two different database.

The system is structured in a four- layered fashion, which will be deeper described in this document, that adapts to several forms of the clients.

The architecture must be designed with the intent of being maintainable and extensible, also foreseeing future changes. This document aims to guide the implementation phase so that cohesion and decoupling are increased as much as possible.

## 1.3 Definitions, Acronyms and Abbreviations

### 1.3.2 Definitions

- Database: a structured set of data held in a computer, it could be distributed if it isn't in the same physical computer.
- Emergency Procedure: is a plan of actions to be conducted in a certain order or manner, in response to an emergency event.
- Query: a request (usually from a third party) in a portion of a single piece of data.
- Personal data: it is any information that relates to an identified or identifiable living individual.
- Third party: an external company that is interested in the software and its data.
- User: a person who is subscribed and uses the system.
- Vital signs or parameters (often shortened to just vitals) : are a group of the 4 to 6 most important signs that indicate the status of the body's vital (life-sustaining) functions.

### 1.3.2 Acronyms

- RASD: Requirements Analysis Specifications Document
- DD: Design document

- DB: Database
- DBMS: Database Management System
- API: Application Programming Interface
- GPS: Global Position System
- GUI: Graphical User Interface
- MVC: Model View Controller (design pattern)
- ACID: Atomicity, Consistency, Isolation and Durability (properties of DB transactions)
- RDBMS: Relational Database Management System
- SSL: Secure Sockets Layer
- ER: Entity Relational
- EMT: a specially trained medical technician certified to provide basic emergency services (as cardiopulmonary resuscitation) before and during transportation to a hospital.

### **1.3.3 Abbreviations**

- [Gn]: for the n-th goal
- [Rn]: for the n-th requirement.

## 1.4 Revision History

This is the first version of the document.

## 1.5 Reference Document

This document is strictly related to the RASD document.

## 1.6 Document Structure

The document is essentially structured in seven parts:

- Section 1: Introduction, it provides a general introduction and overview of DD and the system-to-be with attention to the details not considered in the RASD.
- Section 2: Architectural Design, this section shows the main system components together with sub-component and their interactions.
- Section 3: User Interface Design, it provides an overview on how the user interface will look like and behave giving further information.
- Section 4: Requirements Traceability, this section provides a table where the requirements described in the RASD are mapped to the design elements and components defined in this document.
- Section 5: Implementation, Integration and Test plan, it provides the guide line to implement the components and sub-components of the system; it shows their integration step by step and testing phase to a create a stronger software.
- Section 6: Effort Spent, it shows in a table the effort spent by each group member for each job.
- Section 7: References, it includes the reference documents.

# 2 Architectural Design

This chapter provides a detailed view on the architecture of the system. From a high level view of the macro components to a specific description of the physical layer and business logic.

It explains in detail the communication between layers and how components interacts one with the other.

Section 2.1 describes high-level components of the system; each of this components will be separately described and detailed in section 2.2; section 2.3 focuses on the deployment of the system on physic tiers; section 2.4 describes the dynamic behavior of the software; section 2.5 focuses on the interface between different components; finally section 2.6 provides a description of the design choices and patterns.

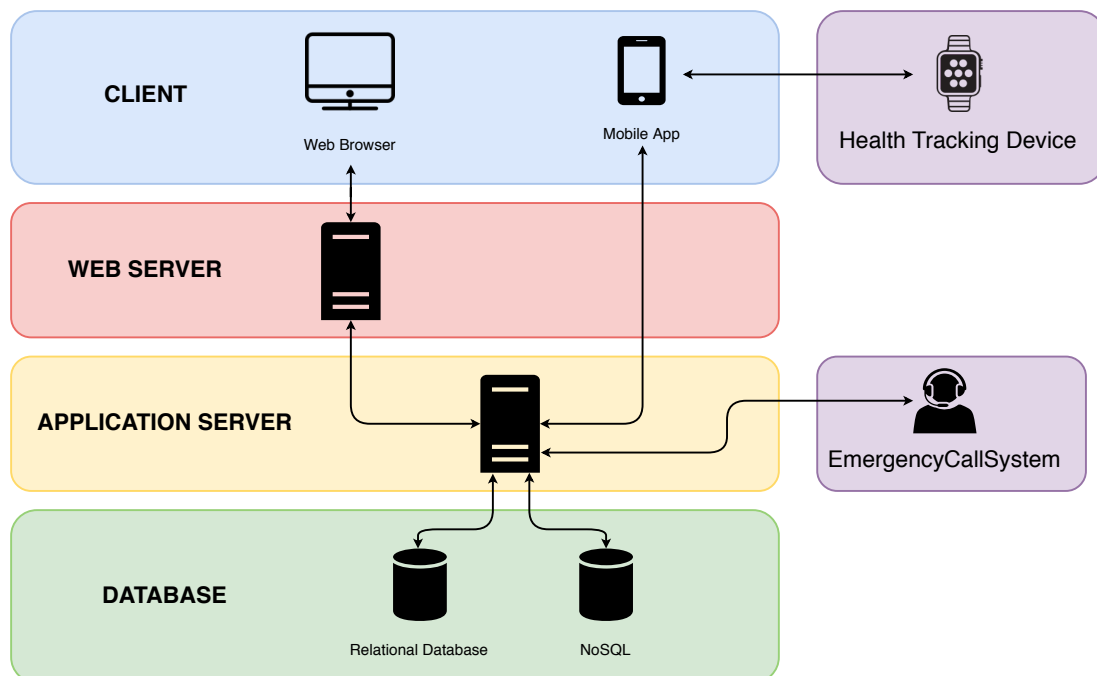
## 2.1 High-level Components and Their Interaction

At a high-level the architecture of the system is composed by four logical layers.

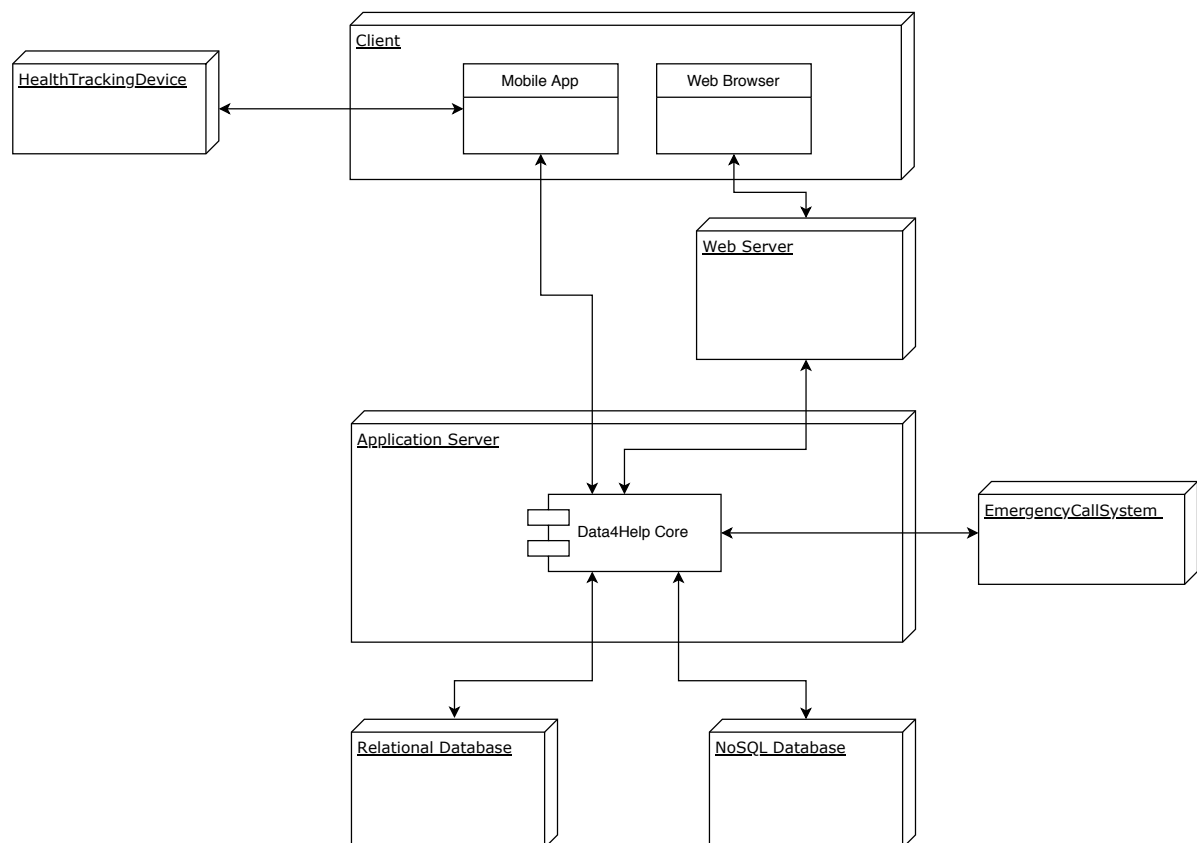
The Client Layer has been designed with the Thin Client approach.

As stated in the RASD, there are two different software interfaces available depending on the type of client. The distinction is visible in the Client Layer, that in turn is connected with the Web Server and with the Logical Layer. Here we can found the Application Server that contains all the logic that runs the system. At the base of the structure there is the Data Layer that stores and manages the big amount of data collected. Both the Client Layer and the Logic Layer interface with the external components, required for the functioning of the system.





**Figure 2.1:** Logical Layers.



**Figure 2.2:** High-Level component Overview.

## 2.2 Component View

### 2.2.1 Relational Database.

As shown in the Figure 2.1, the storage system in the Data Layer is composed by two databases.

The relational database is in charge of storing and managing all the data that is compatible with the relational model. This data is mainly about users' personal information, such as the one regarding identification and authentication.

The relational database runs MySQL as RDBMS, in order to manage the insertion, modification, deletion and logging of transactions.

The database communicates only with the Application Server, as MySQL client, through TCP/IP protocol and using MySQL Proxy, that implements MySQL Protocol .

ACID properties must be guaranteed over all transactions. Correct functioning among concurrent transactions, data protection and data loss avoidance must be guaranteed. For this reason the database must be physically protected and duplicated. Moreover the MySQL Protocol, using SSL, ensures to the client transparent encryption and compression. Those qualities are essential for the function of the system, as it is required to quickly manage and deliver big of mount of data.

### 2.2.2 NoSQL Database.

The storage system is also composed by a NoSQL database. It is meant to store and manage health parameters, related to localization, timing and user identifier. The choice of splitting the storage is due to the fact that this specific kind of data does not require to be updated or deleted, but accessed for reading only.

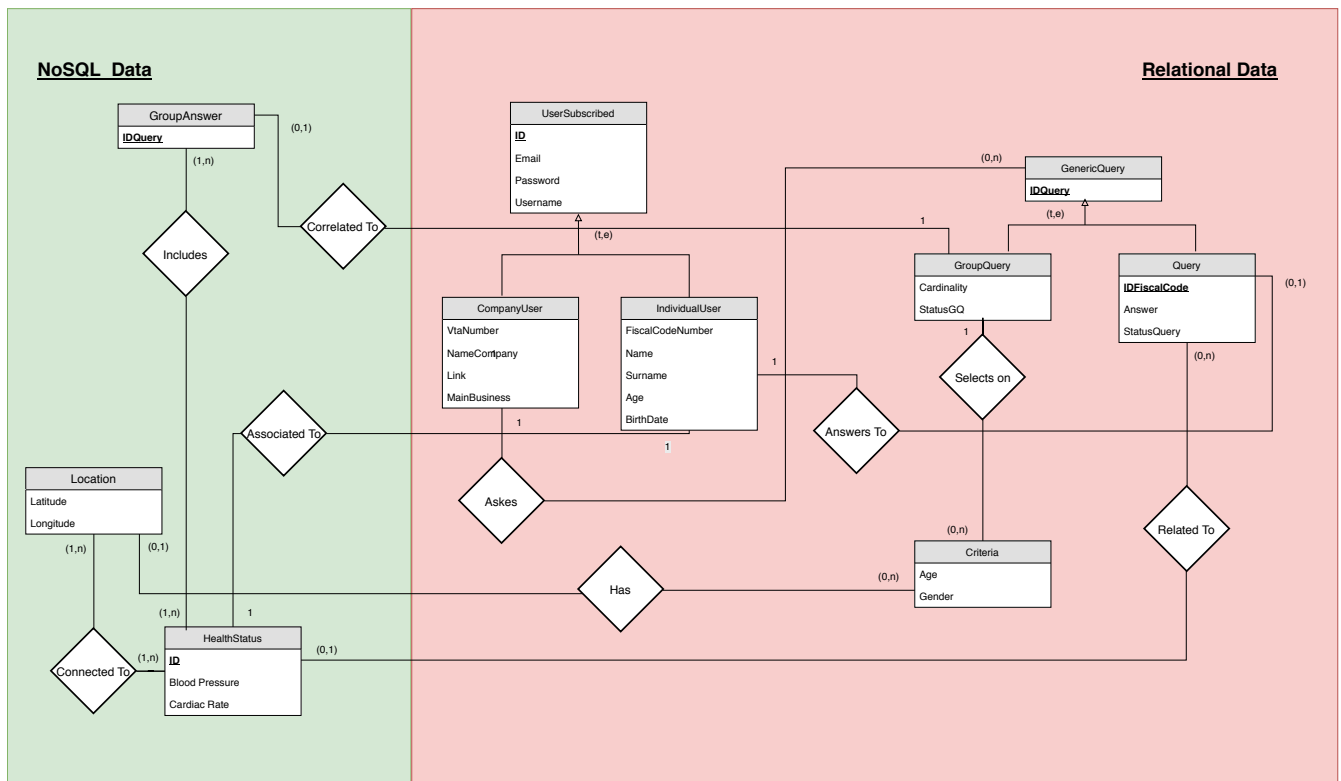
In particular NoSQL has been chosen over the traditional relational database because it offers a much higher speed performance, so that the main transactions, insertion and access, are quickly accomplished.

In addition NoSQL, since it's not based on the relational model, is more scalable than the relational database. It can manage large volumes of data that may rapidly change structure. This is an advantage in view of the

rapidity with who health-tracking sensors are evolving and pushing their technologies to detect new kinds of data.

The NoSQL database runs MongoDB as DBMS and the communication with the Application Layer is wired through TCP/IP protocol, with no handshake on an arbitrary port. On the DBMS side it uses the MondoDB Wire Protocol.

It's not required to ensure ACID properties, however for all the security concerns, the same measures of the relational database are applied.



**Figure 2.3:** Entity Relational diagram for both the relational and NoSQL database.

## 2.2.3 Application Server.

The Application Server, connecting the front-end with the back-end, is in charge of manage all functions of the system.

It's the only module that interfaces with databases and transforms data to be understandable at the user interface level.

The access to the Data Layer is implemented with JAVA Persistent API, so that persistency is ensured and also correct object relations and dynamic access to the proper database of the two.

The Application Server contains the unit that handles the communication with the external system. As well it communicates with different layered client and Web Server by using a specific API, well explained in section 5 of this chapter.

The Application servers is implemented with Java EE and runs on a Glass Fish Server. Moreover it executes the business logic of the system, that is implemented with Enterprise JavaBeans.

The module is the core of the system and it's architecture is organized in components.

### **User Manager.**

This module contains the logic about the identification of each user and the management of its account. It's involves the registration, login and account customization process. It manages consistently all the authentication status, password and sensible data encryption and also the recover account process.

### **Notification Sender.**

The component is responsible for sending to the user's device notifications regarding communication from the system. Alert notifications about parameters above the threshold are generated locally on the app, therefore they are not handled by this component.

In addition it is the gateway from which specific components communication emails are sent.

### **Communication Manager.**

The component is responsible for the REST APIs provided by the system and manages globally the communication between

components. It recognizes and addresses all the requests and answers towards the external system and other layers.

It is capable of handling parallel and repeated operations, necessary for several insertion requests to the storage system. Also recognizing the data to be stored, it wisely knows which protocol use for the proper database. The component ensure persistency communication with data layer.

Moreover is designed to protect the internal implementation and business logic of the Application server by encrypting the operations and the addressee component.

### **Collector Manager.**

It is the component that continuously receives packages containing health data detected from each user.

Each incoming package is stored in a queue. Here it will remain temporarily until the check procedure is ready. This will avoid the lost of packages, since the component is receiving simultaneously each user's new data.

The check procedure is in charge of checking the integrity and correctness of the data. Eventually data is translated in the format used by the data layer and directly forwarded to the storage system though Communication Manager.

In addition the module periodically notifies the Response Manager of what kind of data is been recently collected.

### **Individual Query Manager.**

The component rules the logic about individual requests.

It handles the storing of all the requests incoming from companies, for legal and privacy reasons. As stated in RASD, people in the requests are supposed the be registered to the system. As soon as a specific person is available, thought the Notification Manager, the request is sent to the front-end and displayed to the user. All individual requests are stored in a queue. It's unknown if and when the client will respond, so until that moment they will remain in the queue in a pending state. Long periods are well considered.

When a user responds to its individual request, the state of the request changes, it becomes accepted or denied. If accepted, all user's store data are requested to the data layer. Received the

information, the procedure deletes the individual request from the queue. The individual request state becomes closed and the request itself is logged in the databases.

### **Group Query Manager.**

The component receives from the Web Server each group query request and it's in charge of the logic that rules the acceptance each of them. All requirements of a Group Query are specified in RASD.

Once a query request is accepted it's submitted to the Response Manager, that will handle it.

When the material response is ready, the component organizes the data with the creation of a graphic representation. Eventually the package, with both the raw data and the representation, is sent to the front-end.

### **Response Manager.**

The module is in charge of creating the selections satisfying the requests from the companies.

It manages more requests at a time, so each selection is saved in a queue with a pending state.

When a selection is inserted in the Queries Queue, the procedure sends an initial request to the Data Layer to see if the data required is already available.

For the remaining selections in the queue, the component will smartly add data to each of them as it becomes available.

In this regard, the module is periodically updated by Collector Manager with what new data has been collected. When there is correspondence in what it's been updated and what is pending in the Queries Queue, the procedure sends a request to the data layer for the data of interest that might just have been inserted.

When a selection is fulfilled the module runs an acceptance test to verify that all constraints have been respected. If so the raw data is sent to Group Query Manager.

The module also manages Continuous Delivery functionality, by periodically forwarding the same query to the data layer.

Queries are marked with a special token, containing the information about the option activation and the expire date.

**Emergency Manager.**

The module has the role of interpret health parameters that are above the threshold and, if the case, assign the emergency code. It was the ability of deciding whether to wait for another parameter to be detected or immediately initiate the emergency procedure. Through the Communication Manager, it is the only component that can send to the external system a request, containing the data for the emergency call.

**2.2.4 Web Server.**

The Web Server is the intermediate module between the client and the business logic. It can be accessed only by a web browser using HTTPS protocol.

Each request is translated for been sent to the Application Server through the REST APIs.

No business logic is present in this module, but only the Page Presentation and a Controller.

The Web Servers is implemented with Java EE specific HTTP servlet and HTML5 and it runs on a Glass Fish Server.

**2.2.5 Mobile Application.**

The mobile application UI is designed following the iOS and Android guidelines. The iOS application must be written in Swift, while the Android one must be implemented in Java.

The application, through the controller module, communicates with the Application Server, via REST APIs.

The application has a module that interface with the GPS of the device and one for receiving data via Bluetooth connection.

Locally the application implements the Track Manager module, independent from the business logic of the system.

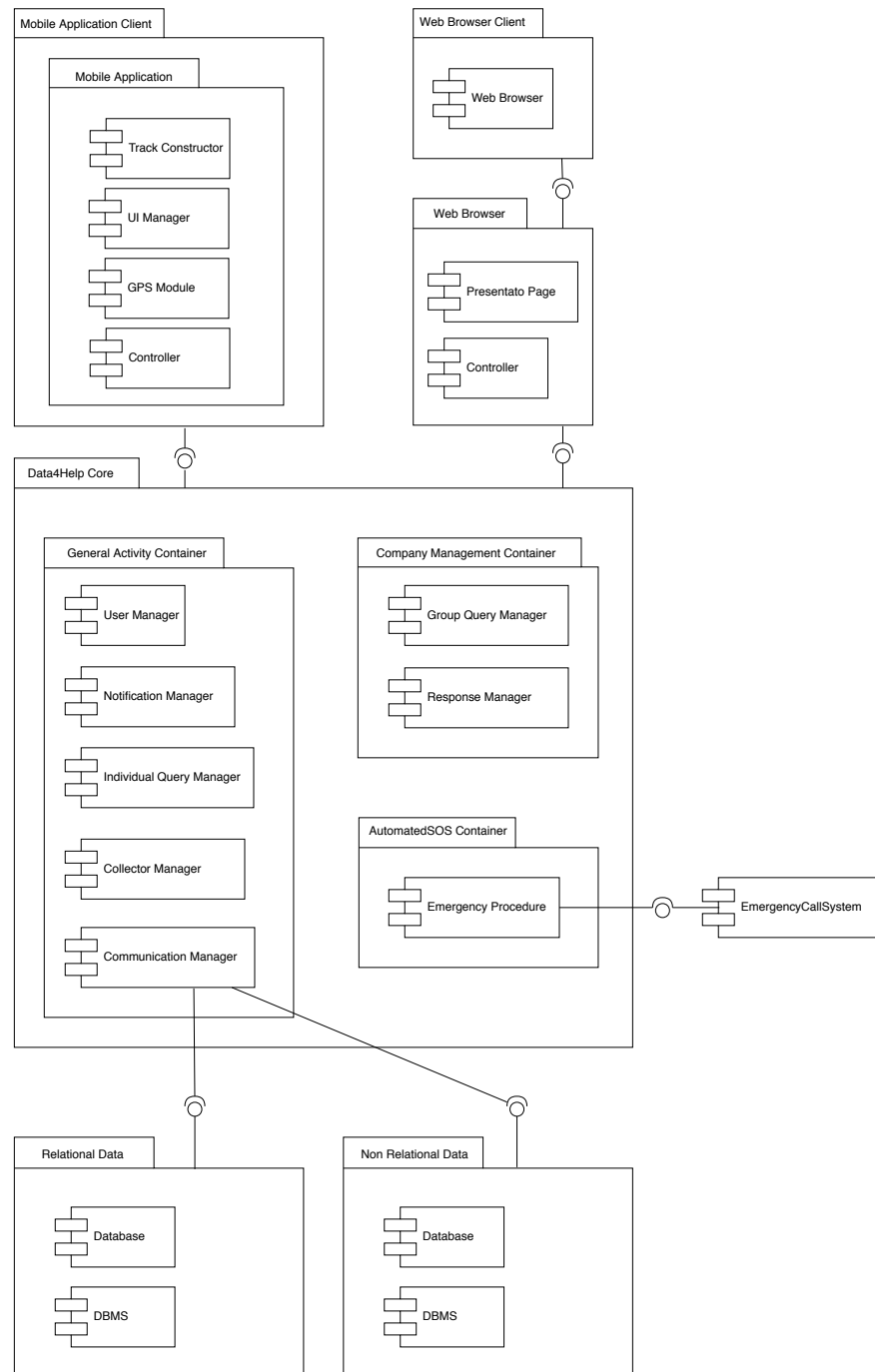
**Tracks Constructor.**

Here is where data collected is assembled into chronological charts. Each parameter is inserted in a chart directly after it is



detected from the sensor. The charts are saved locally and only the raw data is sent to the Server.

Moreover, when a parameter is below a specific threshold, the module is responsible for creating locally, without connection with server, the alert notification.



**Figure 2.4:** Global Component View.

## 2.3 Deployment View

This figure represents the physical deployment of the system divided in three layers: one referring to the Client, one to the Main Server and the last one to the Databases.

In the first layer there are:

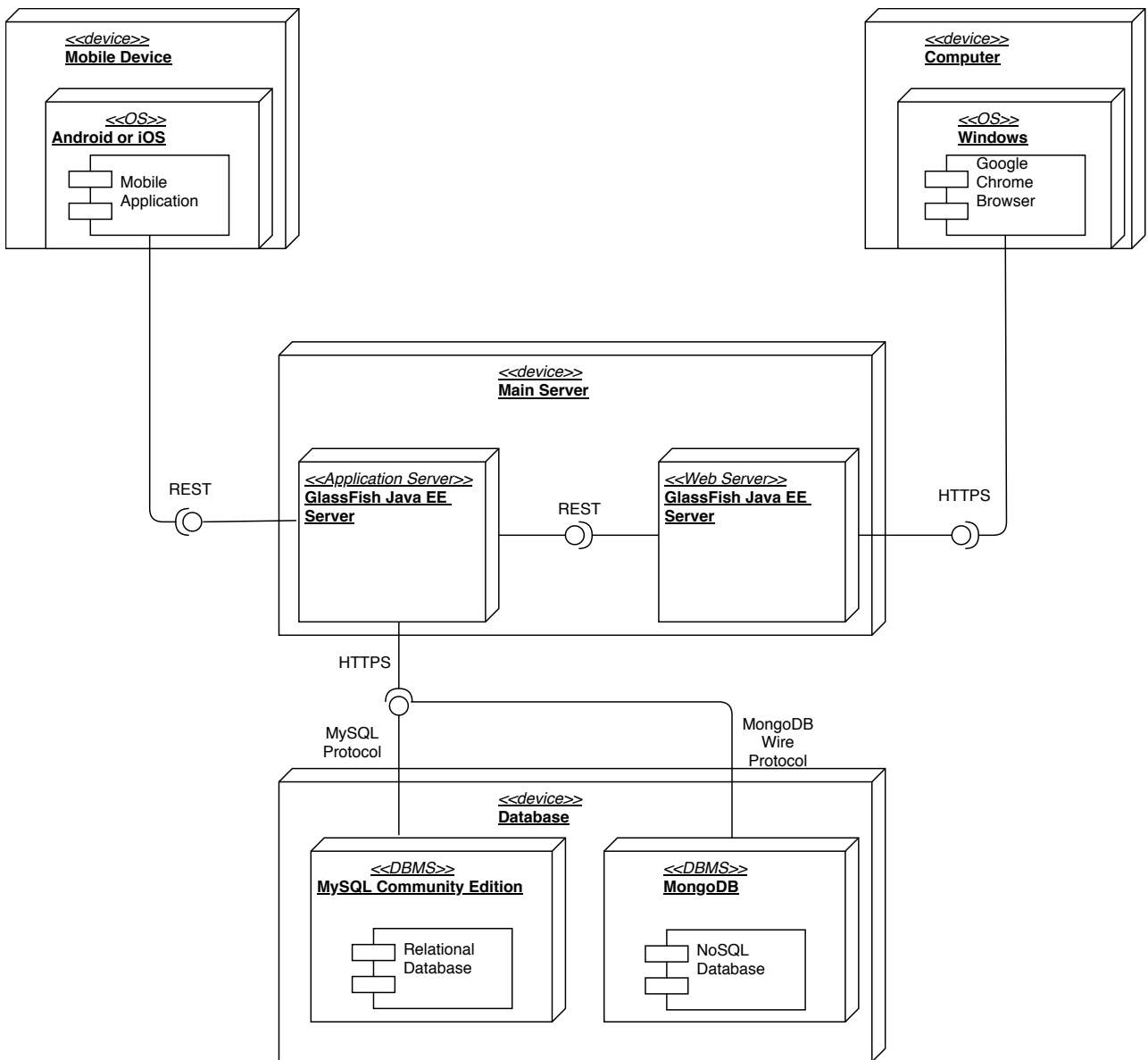
- the individual user, that uses the Mobile Application on the Mobile Device,
- the third party that uses the Browser on the computer.

In the second layer there is the main Server divided in two main blocks:

- the first one is the Application Server that receives data from the Mobile Device with the REST API;
- the second one is the Web Server that uses the HTTPS protocol to communicate with the computers and the REST API to the Application Server.

In the last layer there are two Databases that communicate with the Main Server with the HTTPS protocol:

- the NoSQL one memorizes all the unchangeable data and communicates with MySQL protocol;
- the Relational one memorizes all the other data using MongoDB Wire protocol.



**Figure 2.5:** Deployment Diagram.

## 2.4 Runtime View

This chapter provides the sequence diagrams of the most common scenarios, in order to clarify the behavior of the entire system for each of them.

Each scenario highlights the runtime interactions between clients, components of the system and the entities of the databases. In particular the components of the Application Server present their functionalities with their relationships.

The central node for each scenario is the Communication Manager component, it provides all the interactions between the Application Server and the external world.

- The first figure 2.6 represents the Registration Sequence Diagram, where a guest tries to join the Data4Help system. His request is analyzed by the Communication Manager who contacts the User Manager, then checks the inserted parameters and in case they are complete and coherent inserts them in the relational database and notify the guest.
- The second figure 2.7 represents the Login scenario. The user submits his credentials and the Communication Manager has to check them and send the answer to the client. In case of "forgot Password" the Communication Manager must notify the User Manager that starts the recovery procedure. First generate a new password associated with the user email, updates it on the database and then send it to the client with the Notification Manager. It's important to notice that each operation not related to the Application Server needs the intermediate role of the Communication Manager.
- The third figure 2.8 represents the Request Single Query Scenario where the company provides the fiscal code of the user and submit the query. If the data is correct the Communication Manager notifies the selected User and notify him. Then he can accept or refuse the request. In the positive case his Health Data are sent to the company.

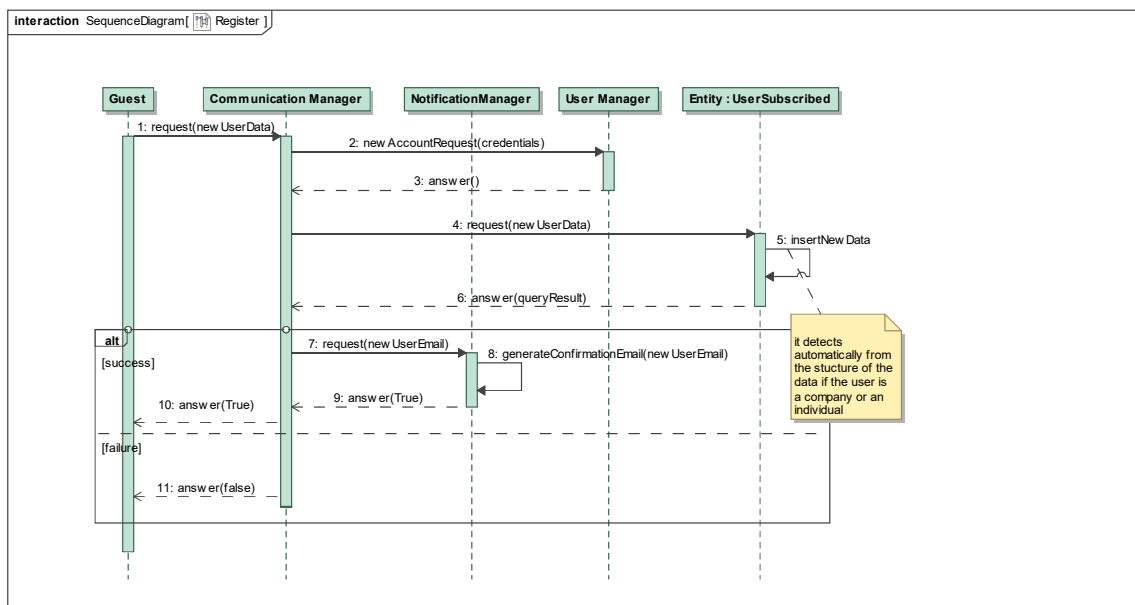
- In the fourth figure 2.9 there is the Request Group Query scenario. The company provides all the necessary criteria for the query and then submits it to the system. There are two possibilities:

1. The query is satisfied, and the data is all present in the database, so the Communication Manager selects them and sends to the company.
2. The query is satisfied but the data are insufficient, so the system takes data from the Collector Manager and waits until the amount of data is correct.

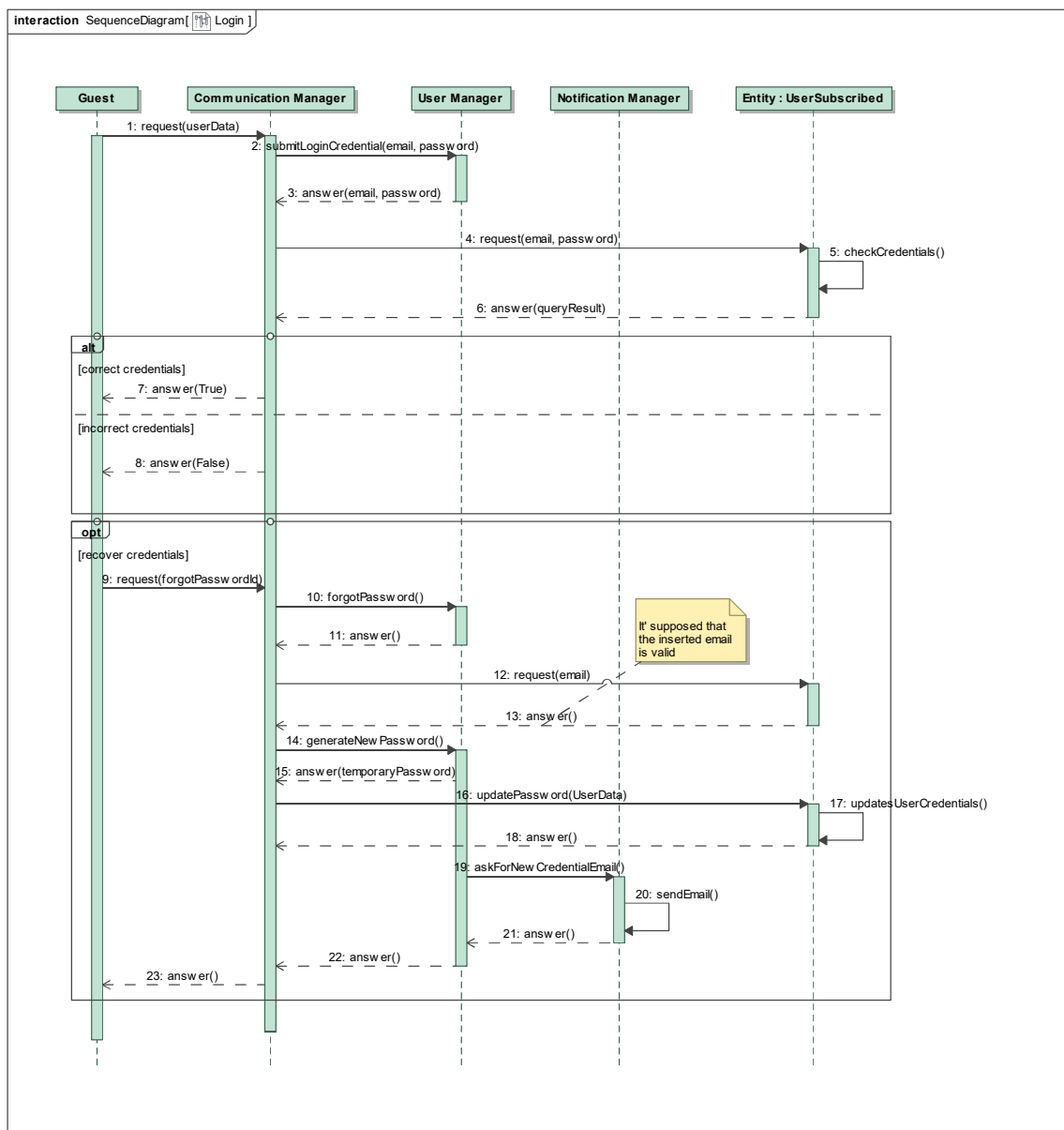
Finally all new data collected, that respect the criteria, are sent to the company with the continuous delivery function.

- In the last diagram 2.10 there are two possible alternatives:

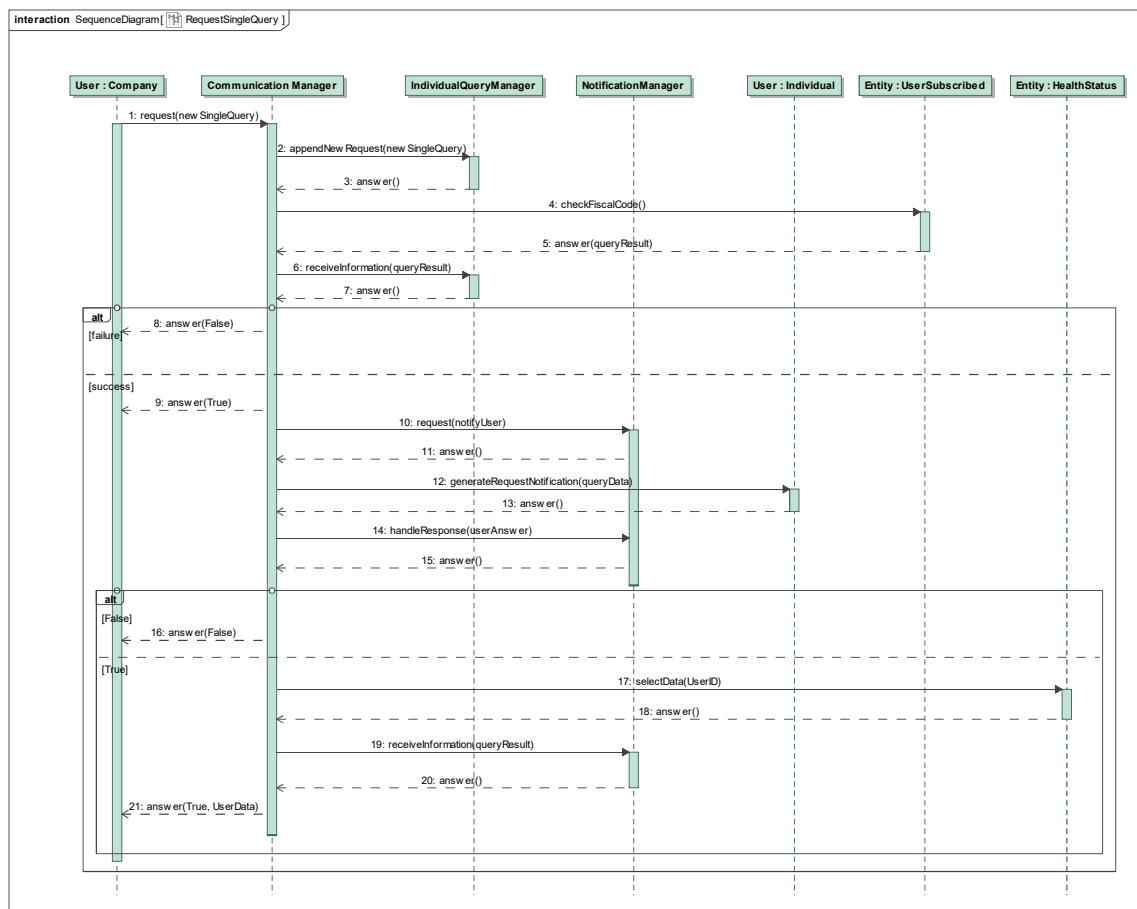
1. The user is under 60 and his device provides continuously new Health Data and the Communication Manager adds them to the NoSQL database.
2. The user is over 60 and his Health Data is under the threshold. In this case the Communication Manager adds the new data and notifies the Emergency Manager that starts the emergency procedure contacting the external service. Then the user is notified that an ambulance is arriving in his location.



**Figure 2.6:** Registration Sequence Diagram.

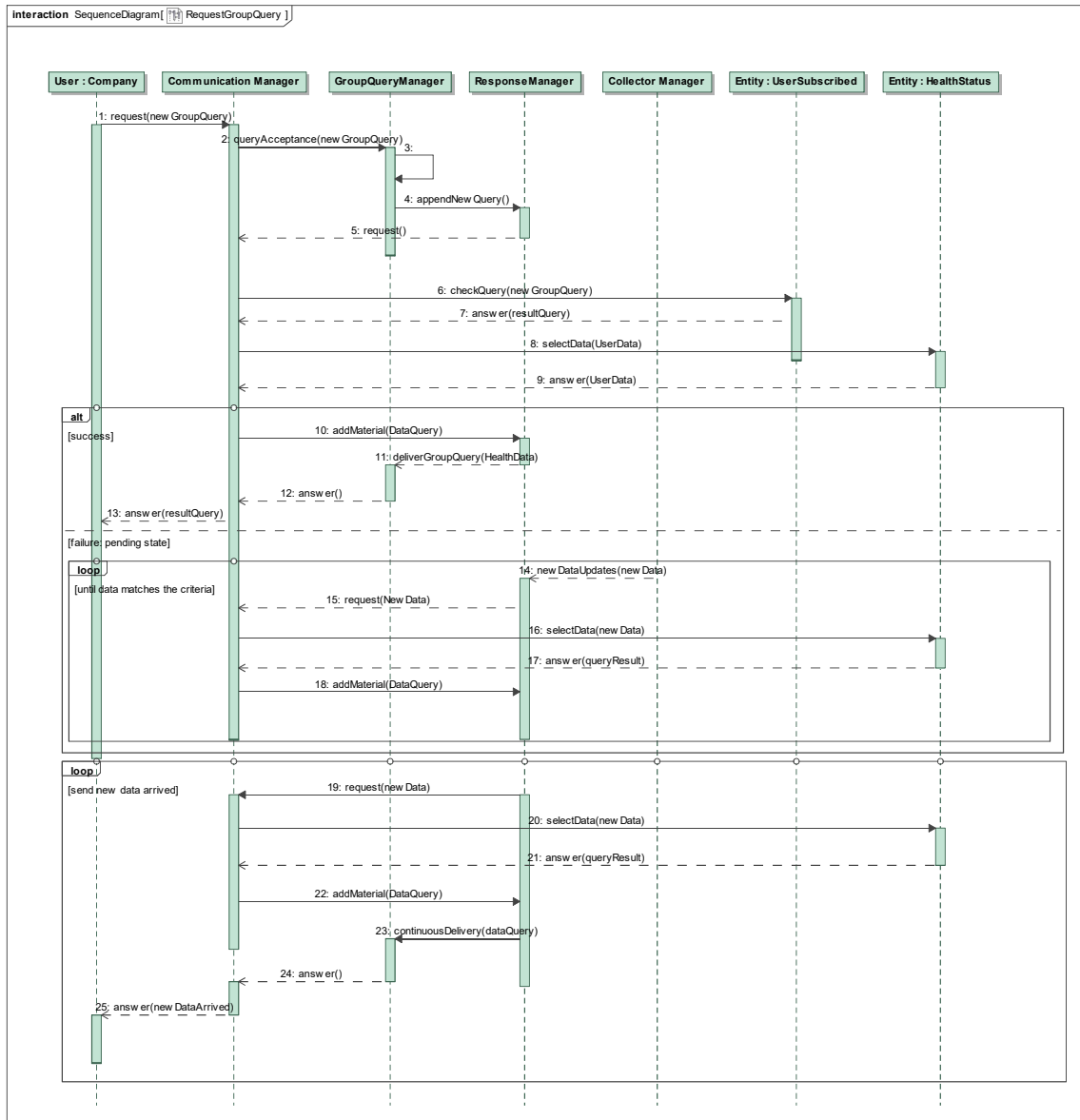


**Figure 2.7:** Login and Forgot Password Process Sequence Diagram.

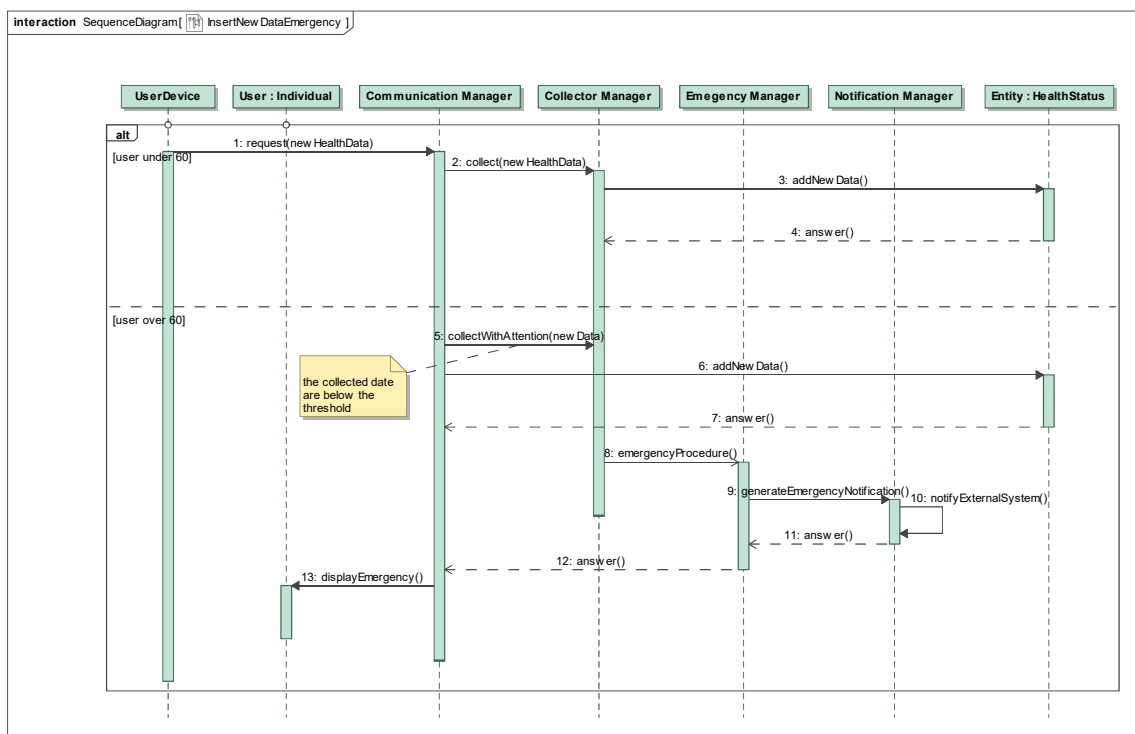


**Figure 2.8:** Single Query Sequence Diagram.





**Figure 2.9:** Group Query Sequence Diagram.



**Figure 2.10:** Data Collection Process Sequence Diagram.

## **2.5 Component Interfaces**

### **2.5.1 Database - Application Server**

The communication between the Application Server and the databases goes using standard TCP/IP protocol on two different ports. Both the relational and the NoSQL databases are abstracted in the Application Server, in the Communication Manager, using Java Persistent APIs. That's the correct SQL dialect for the interaction with both MySQL RDBMS and the MongoDB DBMS.

Since the two databases can not directly communicate one with the other, the merge of the information from both databases is done at the application level by merging two different queries results.

### **2.5.2 Web Server - Web Browser**

The communication between Web Server And Web Browser is the conventional one through HTTPS protocol.

### **2.5.3 Application Server - Web Server and Mobile Application**

The communication between Application Server and both Web Server and Client Mobile Application happens via REST APIs provided by the Communication Manager, component of the Application Server. The communication goes using HTTPS protocol and the responses are in textual, encoded in JSON format.

### **2.5.4 Application Server - External System**

The Application Servers is connected just with one external system, the EmergencyCallSystem. The usage of the system foresees a data flow from the Application Server to the external system, then a confirmation replay from the external system to the Application Server.

The Application Server is connected to the external system via the interface APIs provided by the second, to which the Application Server

must adapt in order to share the correct data needed for the Emergency call.

The communication goes using HTTPS protocol.

## 2.5.5 Internal Interfaces for Application Server Components

### User Manager.

#### *submitLoginCredentials:*

[called by: Client, through Communication Manager]

[input: email, password]

[output: login\_result\_code]

The procedure receives the credentials for the subscription.

#### *newAccountRequest:*

[called by: Client, through Communication Manager]

[input: credentials\_struct]

[output: registration\_result\_code]

The procedure receives the credentials for authenticate the user.

#### *forgotPassword:*

[called by: Client, through Communication Manager]

[input: email]

[output: result\_code]

The procedure receives the user personal email, it checks if it corresponds to the user one, then it creates a temporary password, delivered by the email provided. In case there is not correspondence an error code is returned.

#### *editProfile:*

[called by: Client, through Communication Manager]

[input: credentials\_struct, user\_id]

[output: result\_code }

The procedure receives all the credential of the user, it checks which are changed and if they are acceptable.

## **Communication Manager.**

*request:*

[called by: Everyone]

[input: operation\_id, data]

[output: result\_code]

The procedure is called from components of other layer to request an operation to the components of the Application Server. It implements the switching logic to properly address the request sent data to be managed. The operation\_id identifies univocally the operation to be done and it's related to the data received. Since the data package can be of different type, a specific algorithm for recognizing it has been choose.

## **Notification Manager.**

*generateRequestNotification:*

[called by: Individual Query Manager]

[input: company\_struct ]

[output: delivery\_code]

The procedure creates a notification to be displayed on the front-end. The notification asks the user to accept or deny to share his data with the interested company.

*generateEmergencyNotification:*

[called by: Emergency Manager]

[input: user\_id]

[output: delivery\_code ]

The procedure creates a notification to be displayed on the front-end to communicate that the emergency procedure is completed and ambulance is arriving.

*generateConfirmationEmail:*

[called by: User Manager]

[input: email]

[output: result\_code]

The procedure sends an email to confirm the success of the registration process.

*askForNewCredentialEmail:*

[called by: User Manager]

[input: email]

[output: result\_code ]

The procedure sends an email with the temporary password.

*generateDeliveryEmail:*

[called by: Group Query Manager]

[input: email]

[output: result\_code]

The procedure sends an email to communicate that the data required has been extracted and it is ready for consultation.

### **Individual Query Manager.**

*appendNewRequest:*

[called by: Web Server, through Communication Manager]

[input: fiscal\_code, company\_id]

[output: result\_code ]

The procedure receives a user fiscal code and if it is not corresponding to any user, an error code is returned. If there is correspondence the procedure associated it to a new individual request.

*handleResponse:*

[called by: Client, through Communication Manager]

[input: response\_code, request\_code]

[output: delivery\_code]

The procedure is called to communicate to the Server what has been responded on the front-end. The response\_code can be accepted or denied. A confirmation code is returned by the procedure, meaning that the user information will or will not be extracted and shared.

*receiveInformation:*

[called by: Data Layer, through Communication Manager]

[input: health\_file, request\_code]

[output: delivery\_code ]

The procedure receives the health\_file containing all the tuples extracted from the storage system and related to the user . The data layer is provided with a confirmation message.

Consequently it starts the procedures for closing the request and deliver the information.

### **Collector Manager.**

*collect:*

[called by: Client, through Communication Manager]

[input: health, user\_id]

[output: - ]

The procedure is automatically and periodically called by the front-end. It delivers the health data collected on each user application to this component. There is no need to confirm the success of the operation.

*collectWithAttention:*

[called by: Client, through Communication Manager]

[input: health, user\_id]

[output: delivery\_code ]

The procedure is automatically called by the front-end when a parameter is above the threshold. Since the delivery of this information to the Collector Manager is essential, the procedure returns a confirmation message or an error message if the timer has expired. In this case the procedure is called again.

### **Group Query Manager.**

*queryAcceptance:*

[called by: Web Server, through Communication Manager]

[input: query\_struct, company\_id]

[output: result\_code]

The procedure receives a query request and it decides if it is acceptable or not. If acceptable, the request is sent to the Response Manager that will handle it.

*deliverGroupQuery:*

[called by: Response Manager]

[input: selection\_file, selection\_id, company\_id]

[output: delivery\_code ]

The procedure receives the selection\_file containing the raw data collected to fulfill a selection request. Then it starts the data organization and presentation process. Delivery confirmation code is returned, to make aware of the success of the operation.

*continuousDelivery:*

[called by: Web Server, through Communication Manager]

[input: health\_list, selection\_id, company\_id]

[output: result\_code]

The procedure is called by the Response Manager each time it extracts data to be forwarded in addition to packages already delivered to the front-end. The procedure simply sends data through the communication manager, specifying the identification of the query to which add the information. The confirmation code is returned.

**Response Manager.**

*appendNewQuery:*

[called by: Group Query Manager]

[input: query\_struct, company\_id]

[output: - ]

The component receives a query request, creates the corresponding selection object and inserts it in the Queries Queue, marking with a special token the ones that requires continuous data delivery.



*newDataUpdates:*

[called by: Collector Manager]

[input: information]

[output: - ]

The procedure receives technical information about last period, highlighting what kind of data has been collected the most.

Then the procedure smartly requests to the storage system the data it needs to fulfill the pending selections.

*addMaterial:*

[called by: Data Layer, through Communication Manager]

[input: health\_list, selection\_id]

[output: delivery\_code ]

The procedure has as parameter the list of tuples previously requested to the data layer. Therefore the procedure receives the data and adds it to the related selection in the queue.

The confirmation code is sent to data layer.

## **Emergency Manager.**

*emergencyProcedure:*

[called by: Collector Manager]

[input: health, user\_id]

[output: result\_code ]

The procedure examines the data incoming from the Collector manager. It returns a confirmation value to ensure the Collector Manager that the information is arrived and the emergency procedure has started. In the health\_struct parameter are contained the position information and, with the user\_id and emergency code, they will be sent to the EmergencyCallSystem.

## 2.6 Selected Architectural Styles and Patterns

The architectural styles and patterns used are:

### **Multi-layer architecture in a Client-Server Model**

The client-server multi-tiered architecture allows to physically separate presentation, application processing and data management operations. The system is split on four logical layers and three physical layers.

The presentation layer is on the client as mobile app and web app, the application layer is on web server and the data layer on the database. The mobile application is a client with respect to the Application Server that receives the data, otherwise the company communicates with the Web Server which provides the Web page with all the requested information received from the Application Server. This last component takes the role of the client when asks queries to the Databases that is responsible of fetching query results.

### **Model-View-Controller Pattern**

All the system is based on the MVC software design pattern suggested for every object-oriented project. It separates the model, that manages the data, from the view, that allows presentation layer in the Mobile App and on the computer, and from the controller that checks the right interaction between the previous two components.

### **Location**

The external device provides the position of the user towards the GPS. It is a global navigation satellite system that provides geolocation and time information to the health track device that sends all this data to the mobile app.

The user app forwards this data to the system that memorize them in the database.

## **Bluetooth Connection**

The external health track device must be connected quickly and easily to the app, but not all the models presents the possibility to have a sim. In order to gain the greatest amount of client the Bluetooth connection is the best option in terms of time and cost.

## **External Call Service**

The system is connected to an external service named "External Call Service". When the system recognizes an emergency automatically creates an input to send to this service by REST.

The system does and manages calls and it's a COTS already present on the market. It's completely developed and friendly with the EMT's dispatch, that takes into account all emergency calls.

## **Reliability and Availability**

In order to achieve these properties is necessary a combination of redundancy and parallelization in the system implement phase.

The described architecture is created in order to easily support scalability and parallelism (See following section for a deeper analysis).

## **Password and Sensible Data**

The system collects users' password and other sensible data as the private health status and their specific parameters linked to their position; for this reason, the database must be protected both physically and logically in order to avoid any kind of data loss or force.

So, password should be salted (in order to avoid cyber-attack) and encrypted using the most advanced algorithms (e.g. cryptographic hash functions).

# 3 User Interfaces Design

In this chapter are presented more precise mockups for the Application on the mobile device (individual user) and the Web Application on the computer (company).

- Mobile App

When a user downloads the app has the possibilities to login or sing-up. In case he has forgotten his password he can start the procedure clicking on "Forgot Password?".

It's presented also the Registration form that the User has to compile it, then there is the "PairYourDeviceUser" figure that represents how to connect the Health Track device to the App.

After these mockups there is the home page that presents all the possible User actions.

He can edit the profile, see the pending individual requests from the companies, he can see his Health Track or pair again the device in case it have been disconnected.

- Web Application

Similarly, to the user one, in the desktop version the Web site provides login and register operations to Data4Help system.

The company home page presents three main possibilities:

1. The single query operation, the company easily insert the user fiscal code and press enter to submit the query.
2. The Group Query page, where the third party inserts all the criteria and submits the query.
3. Collected Data page that provides all the data from all the queries accepted. The company can also research in his data and see graphs that summarize the results in order to do mining on specific parameters.

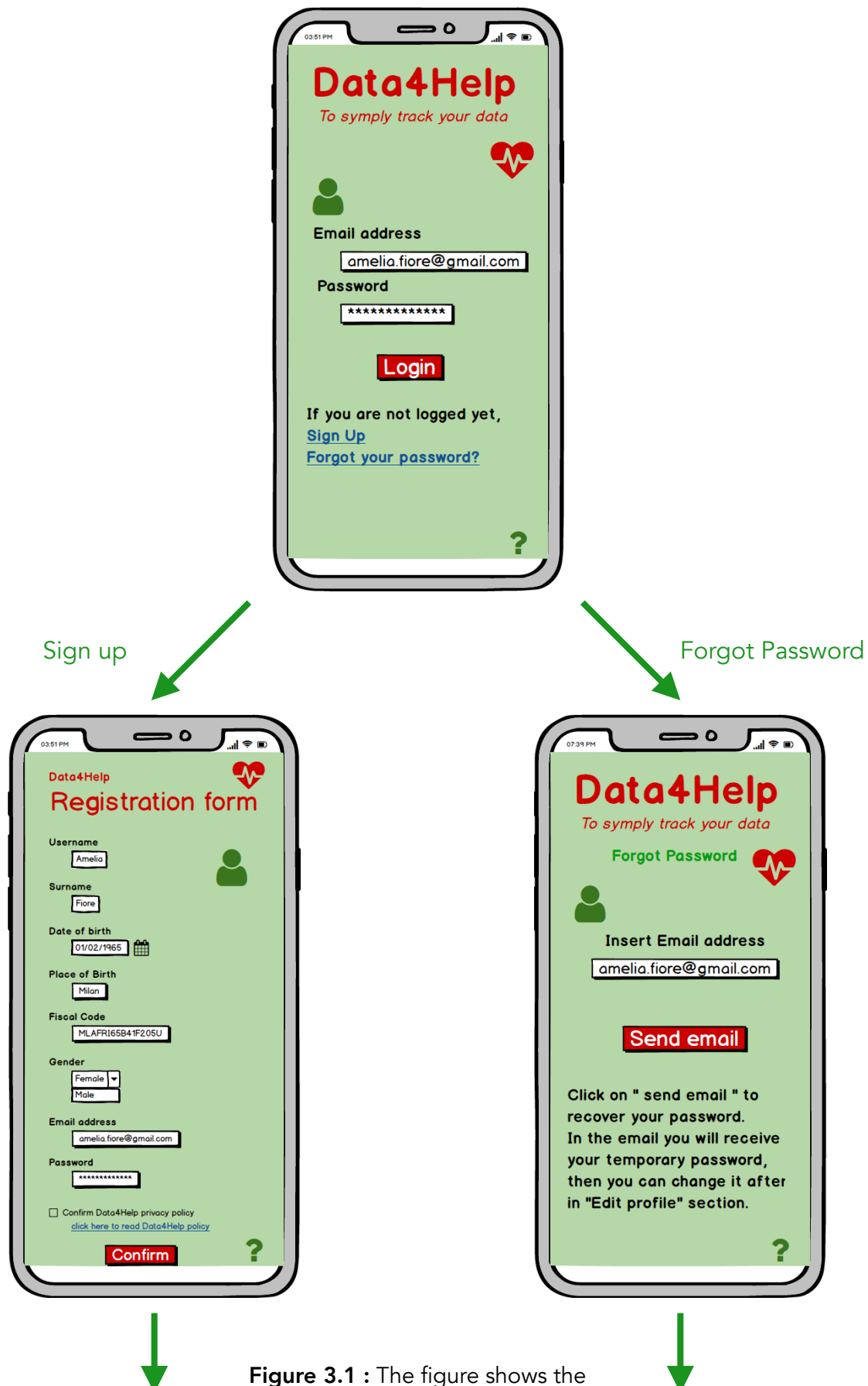
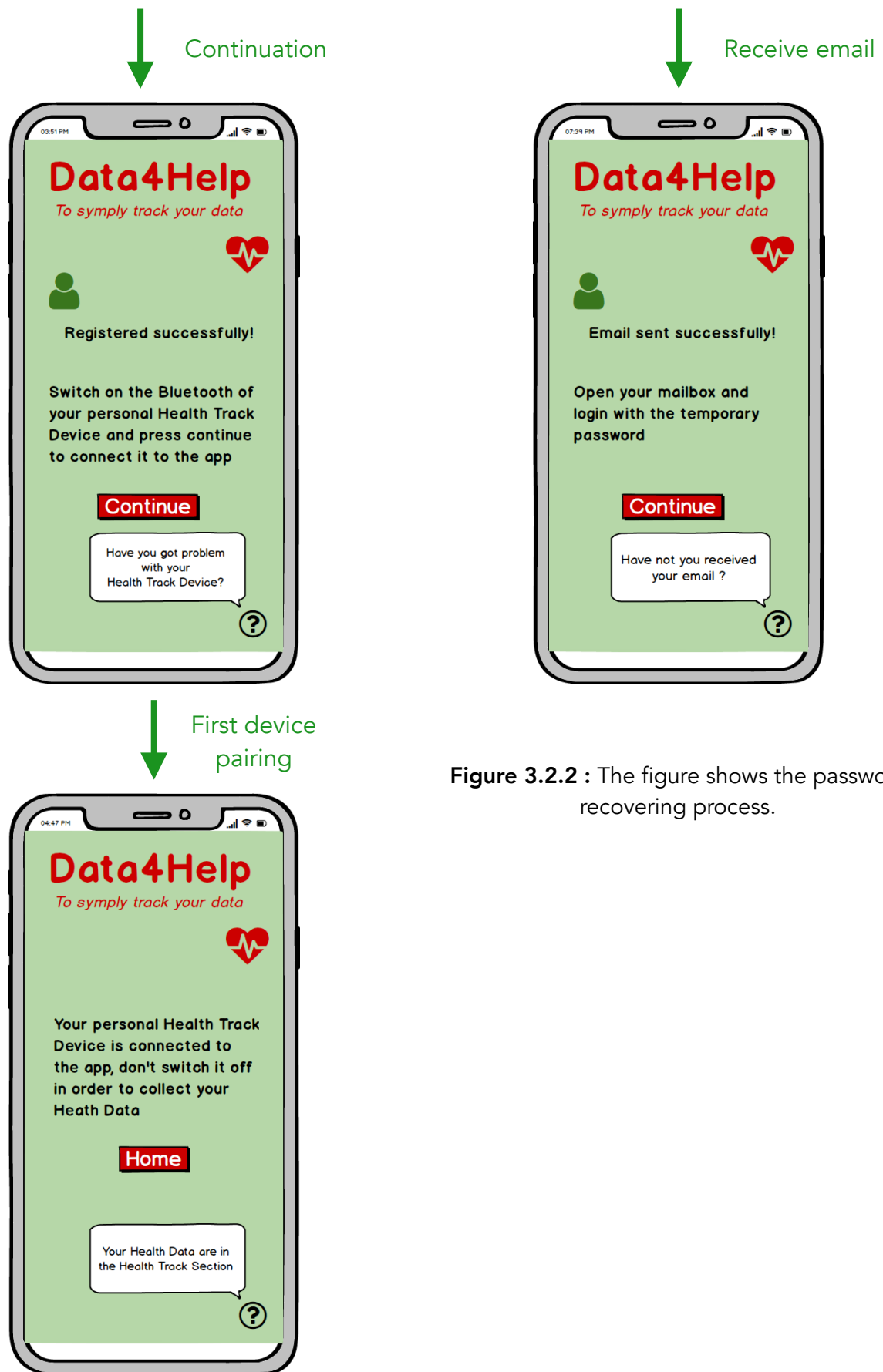
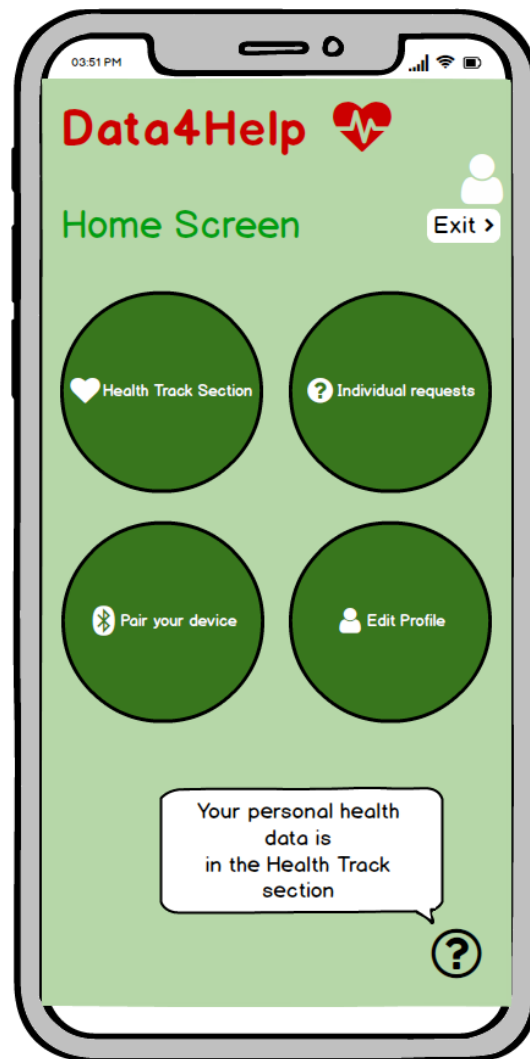


Figure 3.1 : The figure shows the Start Page.



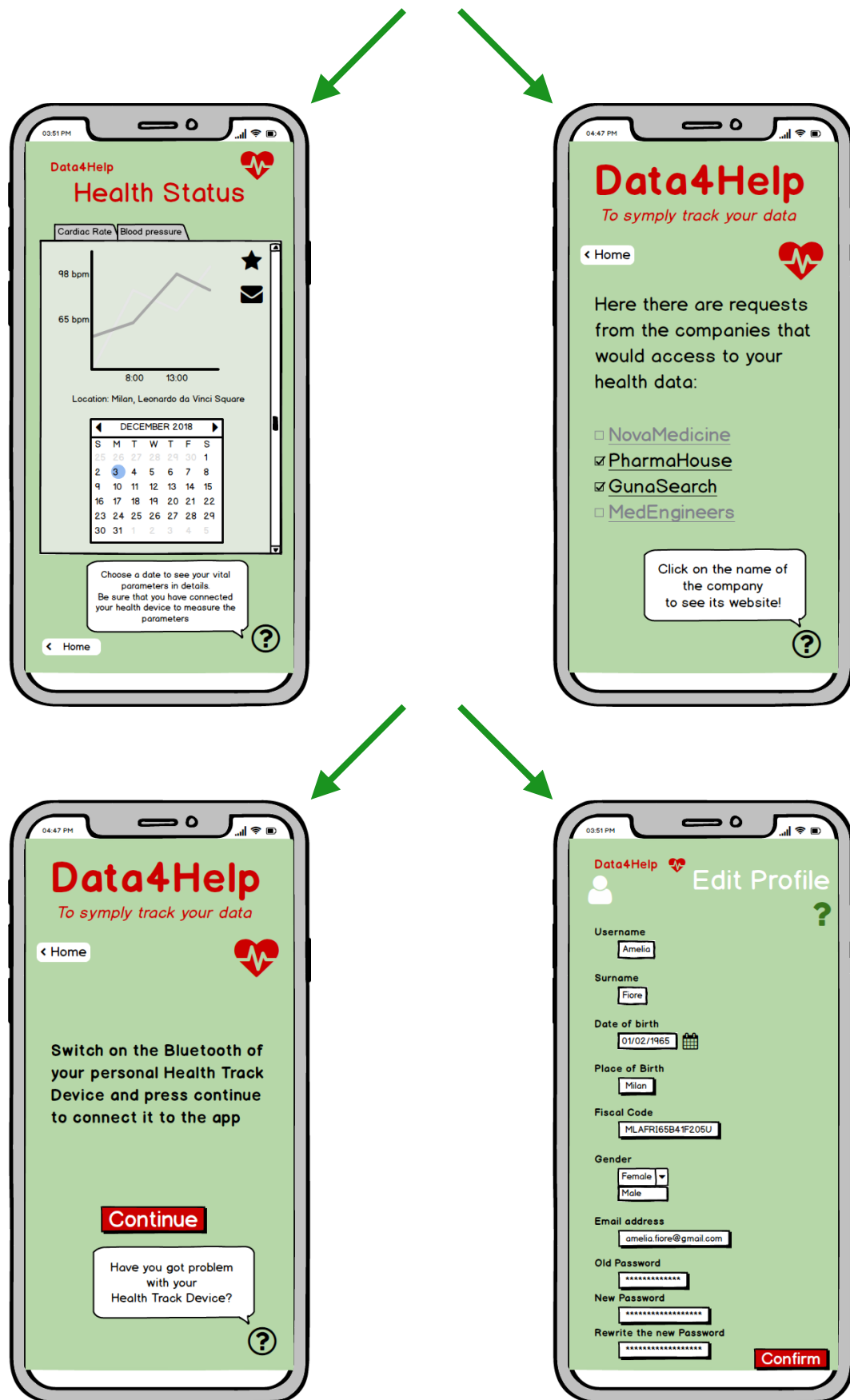
**Figure 3.2.2 :** The figure shows the password recovering process.

**Figure 3.2.1 :** The figure shows the first device pairing.



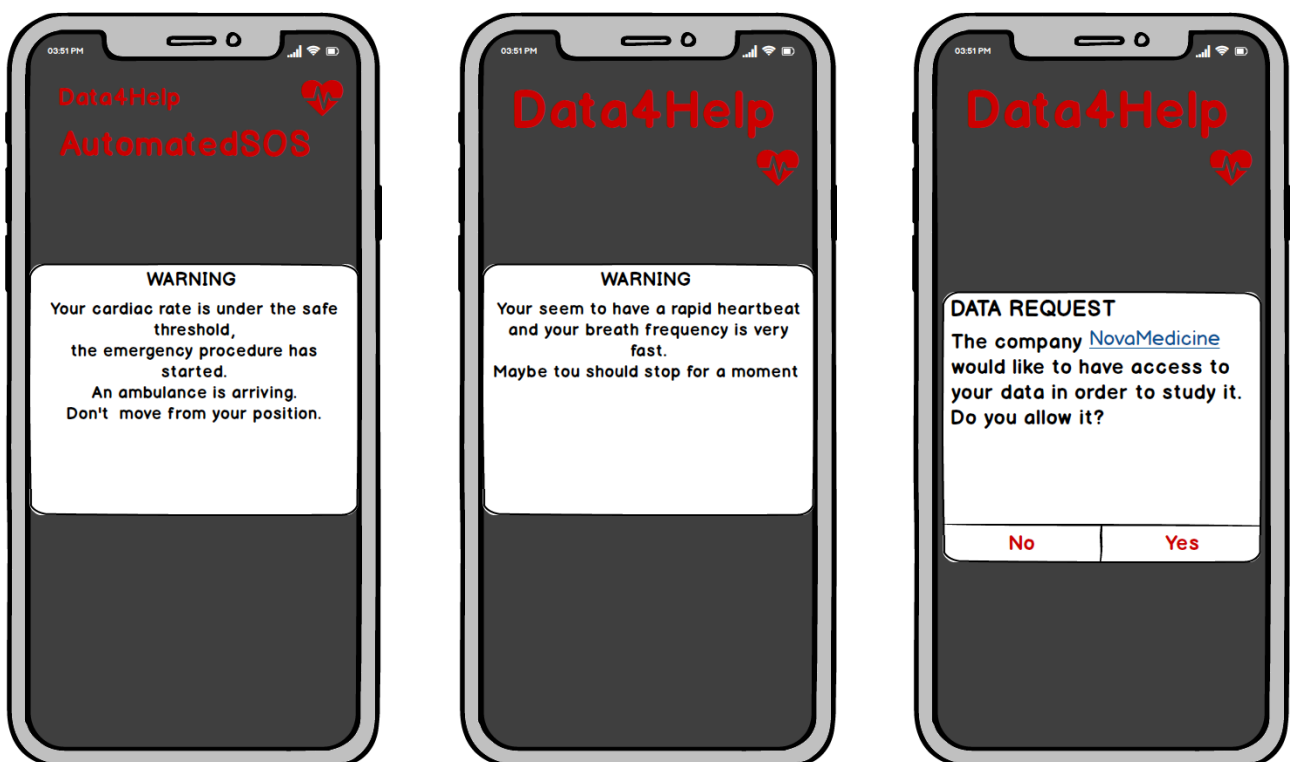
Opening Function

**Figure 3.3 :** The figure shows the Home Page.  
**Note:** The arrow splits for each function in the following page.



**Figure 3.4 :** The figure shows each function accessible from the Home Page. From left to right: Health Track Section, Individual Request, Pair Your Device, Edit Profile.





**Figure 3.5 :** The figure shows each possible warning.

← → × 🏠

Welcome!

[Login](#) [Who we are](#) [Contact us](#) [Join us](#)

# Data4Help

## LOGIN

Email address

Password

**Submit**

If you are not logged yet, [Sign Up](#)  
[Forgot your password?](#)

If you are an individual user,  
[Download the app](#)

?

↓ Sign up

← → × 🏠

RegistrationForm

# Data4Help

Name of the company

VTA number

Principal Activity

Email address

Password

☐ Confirm Data4Help privacy policy  
[click here to read Data4Help policy](#)

**Confirm**

Data4Help tracks health activity and status of a large amount of people of all age and places.


?

Figure 3.6 : The figure shows the Start Page for Desktop.

Browser window: Data4Help  
Address bar: http://www.trackme.com/Data4Help/WelcomePage

Welcome!

Navigation: Login | Who we are | Contact us | Join us

**Data4Help** 

### FORGOT PASSWORD

Insert Email address

**Send Email**

Click on " send email " to recover your password.  
In the email you will receive your temporary password, then you can change it after in "Edit profile" section.

Have not received your email?






Receive email

Browser window: Data4Help  
Address bar: http://www.trackme.com/Data4Help/WelcomePage

Welcome!

Navigation: Login | Who we are | Contact us | Join us

**Data4Help** 

### FORGOT PASSWORD

**Email sent successfully!**

Open your mailbox and login with the temporary password

**Login**

If you have not received your email try again to compile the form


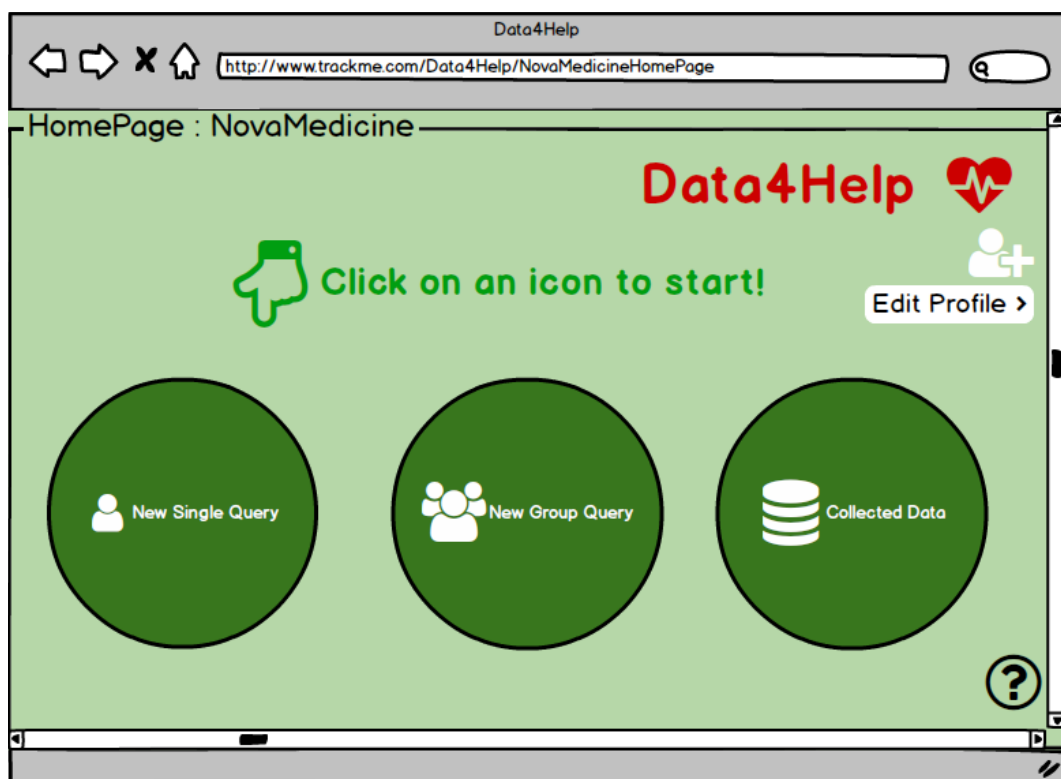


Figure 3.7 : The figure shows the password recovering process.



Opening Function

**Figure 3.8 :** The figure shows the Home Page Desktop Version. **Note:** The arrow splits for each function in the following pages.

→

Insert a single query : NovaMedicine

[< Home](#)

**Data4Help**

**INSERT USER PERSONAL DATA**

User Fiscal code

**Submit**

The results can be visible in the Collected Data section

→

Insert a group query : NovaMedicine

[< Home](#)

**Data4Help**

**INSERT CRITERIA**

Gender

☐ Female and Male

Age From  To

Location

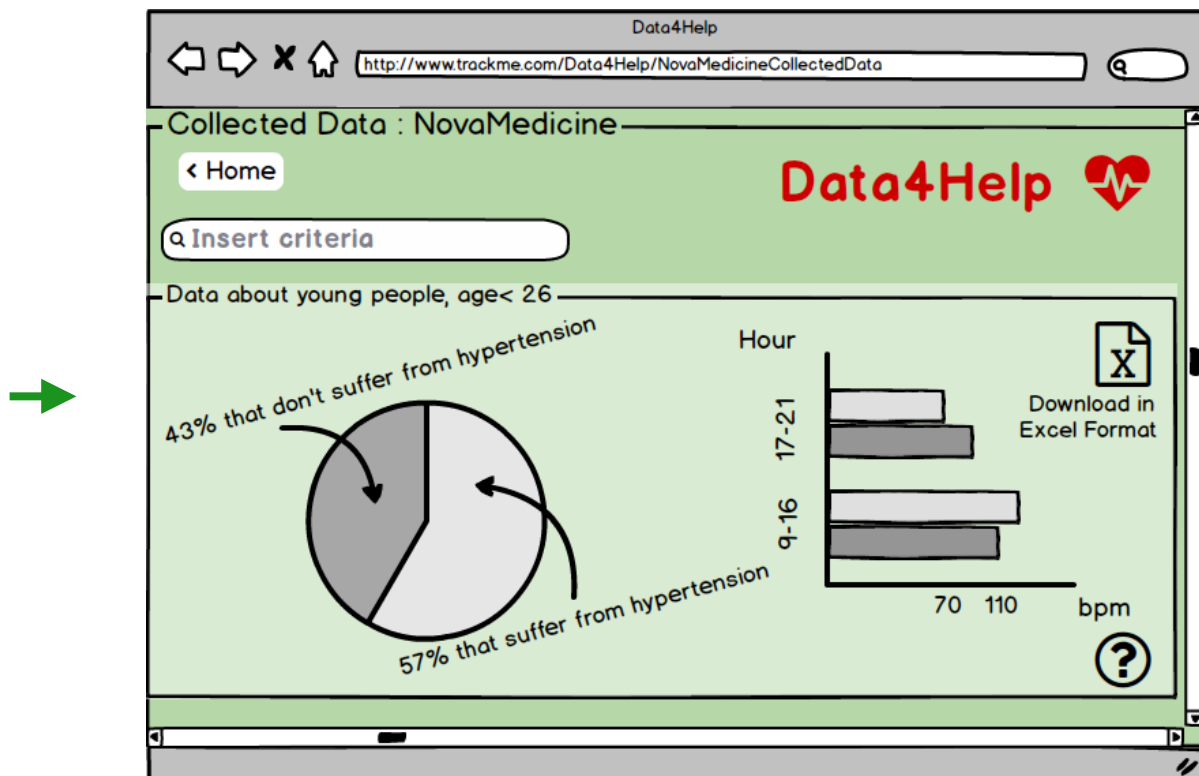
Cardinality

people

**Submit**

The results can be visible in the Collected Data section

**Figure 3.9 :** The figure shows the functions New Single Query and New Group Query, accessible from the Home Page.



EditProfile : NovaMedicine

Name of the company

NovaMedicine

VTA number

0764352056C

Principal Activity

Search on medicine

Email address

NovaMedicine.CEO@novaMedicine.com

Password

\*\*\*\*\*

New Password

\*\*\*\*\*

Rewrite new Password

\*\*\*\*\*

Confirm

To change a field of your profile, you have only to click on it

?

Figure 3.10 : The figure shows the functions Collected Data and Edit Profile, accessible from the Home Page.

# 4 Requirements Traceability

Goal	Component	Functional Requirements
G2	User Manager	[R9] - Identification requirement. [R]10 Authentication requirement.
G5, G7	Notification Manager	[R12] - Warns requirement.
G1	Collector Manager	[R7] - Data collection requirement.
G3, G4	Mobile Application with Track Constructor	[R8] - Health tracking device connection requirement. [R12] - Warns requirement. [R11] - User interface requirement.
G5, G6	Individual Query Manager	[R2] - Company interface requirement. [R5] - Individual request requirement.
G5, G6	Group Query Manager	[R1] - Data Selection requirement. [R2] - Company interface requirement. [R6] - Continuous data delivery requirement.
G6	Response Manager	[R3] - Data supply requirement. [R4] - Data anonymization requirement. [R6] - Continuous data delivery requirement.
G7, G8	Emergency Manager	[R13] - Different emergency cases analysis and emergency code generation requirement. [R14] - Automated and within 5 seconds management of the external system requirement.

**Table 4.1** : Mapping between DD components and RASD functional requirements. **Note:** Communication Manager is not present because its function is the management of the communication between components and other layers. Therefore is not directly linked to a specific goal or requirement.

<b>DD Sections</b>	<b>Non Functional Requirements</b>
2.1 High Level Components	3.6.2 Availability
2.3 Deployment View	3.6.1 Reliability.
2.5 Component Interfaces	3.1.4 Communication Interfaces.
2.6 Selected Architectural Styles and Patterns	3.6.3 Security.
3 User Interface Design	3.1.1 User Interfaces.
5 Implementation Choices	3.6.5 Portability.
5 Implementation Choices	3.4.1 Parallel Operations.

**Table 4.2 :** Mapping between DD sections and RASD non functional requirements.



# 5 Implementation, Integration and Test Plan

## 5.1 Implementation Plan and Integration Strategy

In this chapter is presented a possible strategy that the development team should follow in order to implement and integrate Data4Help system to avoid possible errors and modifications that can occur during the development and that can cause losses of time.

Many system's components implement specific tasks, they are designed to have only the necessary dependencies so each of them can be implemented by a team in parallel with the other components.

After the implementation of two components that are related, the integration must be done as soon as possible in order to find eventually errors and misunderstandings.

The only component that is the centre of all relationships is the Communication Manager. This one has to be implemented at the end of the process in such way that its functions can be expanded and modified in future. For a slim but efficient implementation of the component the Visitor Pattern is used.

Finally, the development strategy should follow the modern approaches based on the release of multiple versions of the software. From a basic prototype all the versions of the software should be shown to the stakeholders in order to receive feedbacks and make corrections.

The plan suggested is:

1. Implement User Manager
2. Implement Individual Query Manager
3. Integrate User Manager and Individual Query Manager

**At this point the system should manage all the user options and operations (except the notifications).**

4. Implement Notification Manager

5. Integrate User Manager, Notification Manager

**At this point the user is complete and it should be tested in all its possible functions. This first prototype of the user profile should be evaluated by the stakeholders.**

6. Implement Collector Manager

7. Implement Response Manager

8. Integrate Collector Manager with the Response Manager

9. Implement Group Query Manager

10. Integrate Group Query Manager with the two previous components

11. Implement Communication Manager

12. Integrate Communication Manager with all the components that communicate with other layers.

**At this point all the Data4Help system and small version is complete, all the test should be done, and feedbacks of this version should improve modifications and corrections.**

13. Implement EmergencyProcedure

14. Integrate EmergencyProcedure with Data4Help system.

## 5.2 Test Plan

The test plan is the base of all robust software, it starts at the begin of each component function and continues to all the components until the integration and the development of the system.

For each components this section provides a list of tests to perform during the implementation tests and the main integration tests. This list could be extended during the coding process and corrected in order to achieve a complete coverage of the components and the system.

### 5.2.1 Unit Tests

Component	Unit test
User Manager	<ul style="list-style-type: none"><li>• Test newAccountRequest (Registration).</li><li>• Test submitLoginCredential (Login).</li><li>• Test forgotPassword.</li><li>• Test editProfile.</li></ul>
Individual Query Manager	<ul style="list-style-type: none"><li>• Test appendNewQuery. Try with a fiscal code not associated to a subscribed user are discarded.</li><li>• Test handleResponse.</li><li>• Test receiveInformation.</li></ul>
Notification Manager	<ul style="list-style-type: none"><li>• Test generateRequestNotification.</li><li>• Test generateEmergencyNotification.</li><li>• Test the procedure for automatically write the emails.</li></ul>

Collector Manager	<ul style="list-style-type: none"> <li>• Test that corrupted or fractional data is discarded.</li> <li>• Test that updates to Response Manager are pertinent and well time-scheduled.</li> <li>• Test collectWithAttention with a parameter above the threshold.</li> </ul>
Response Manager	<ul style="list-style-type: none"> <li>• Test appendNewQuery.</li> <li>• Test newDataUpdates: the pertinence of queries created for data layer and the correspondence with data updates.</li> <li>• Test addMaterial.</li> </ul>
Group Query Manager	<ul style="list-style-type: none"> <li>• Test queryAcceptance. Try with criteria under a thousand.</li> <li>• Test deliverGroupQuery.</li> <li>• Test continuosDelivery.</li> </ul>
Communication Manager	<ul style="list-style-type: none"> <li>• Test that incoming request are properly linked to the components with the function accomplish them.</li> <li>• Test persistency for the data transferred and ensure there is not data loss.</li> </ul>
Track Constructor	Test considering all bound parameters representation limits.
Emergency Manager	<ul style="list-style-type: none"> <li>• Test EmergencyProcedure: the automation of the component and proper emergency code association according to different situations.</li> </ul>

**Table 5.1 :** Table matching components to Test Units.

## 5.2.2 Integration Tests

Component	Integration test
User Manager	<ul style="list-style-type: none"><li>• Test newAccountRequest by creating a new account via Client Interface</li><li>• Test submitLoginCredential by logging via Client Interface</li><li>• Test forgotPassword via Client Interface</li><li>• Test editProfile via Client Interface</li></ul>
Individual Query Manager	<ul style="list-style-type: none"><li>• Test that, creating a new request via Web Browser, the information, including fiscal code, arrives to the component.</li><li>• Test that for each request in the component the client is notified via Notification Manager.</li><li>• Test that the user-data request in the data layer arrives correct.</li><li>• Test the final data package is correctly and completely delivered on front-end.</li></ul>
Notification Manager	<ul style="list-style-type: none"><li>• Given a notification-input, test the delivery of the notification or email on the front-end.</li></ul>

Collector Manager	<ul style="list-style-type: none"> <li>• Test the presence and correctness in the component of a group of data detected on the front-end and that it has been sent with the proper procedure: collect or collectWithAttention</li> <li>• Test the delivery of the group data in the data layer by making a query.</li> <li>• Test that updates are correctly delivered to Response Manager.</li> <li>• Test that the communication with Emergency Manager is correct and ensured.</li> </ul>
Response Manager	<ul style="list-style-type: none"> <li>• Test that the queries requested by the component are correctly forwarded to the data layer.</li> <li>• Test that a fulfilled selection incorrectly and completely delivered to the Group Query Manager.</li> </ul>
Group Query Manager	<ul style="list-style-type: none"> <li>• Test that a request created on the front-end arrives to the component.</li> <li>• Test a accepted request is forwarded to the Response Manager.</li> <li>• Test the final data package and single data additions are correctly and completely delivered on front-end.</li> </ul>
Communication Manager	<ul style="list-style-type: none"> <li>• Test that the request procedure forwards the command, and corresponding data, to the right component by trying for each operation_id, via the proper sender.</li> </ul>

Emergency Manager	<ul style="list-style-type: none"> <li>• Test that the component automatically appeals to the external system by generating via the client an emergency code such that an ambulance is needed.</li> </ul>
-------------------	---

**Table 5.2 :** Table matching components to Integration Tests.

# 6 Effort

Date	Eugenio Cortesi	Chiara Criscuolo
22/11/18	-	1
23/11/18	-	2
26/11/18	6	1
30/11/18	2	2
1/12/18	4	5
2/12/18	6	-
3/12/18	4	8
5/12/18	5	2
6/12/18	3	5
7/12/18	6	-
8/12/18	1	2
9/12/18	1	1

**Table 6.1** : Effort Spent.



# Change-log

- 2.2.5 Mobile Application: description and figure have been updated.