# REMOTE HEALTHCARE SYSTEM EXPLOITING MOBILE AND WEARABLE DEVICES

BY

Voltsis Evaggelos

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DIPLOMA

OF

ELECTRICAL & COMPUTER ENGINEERING

**SCHOOL OF ELECTRICAL & COMPUTER ENGINEERING**

**TECHNICAL UNIVERSITY OF CRETE**

**2018**

# Abstract

In this thesis we discuss the design and implementation of a platform that would be able to simplify and improve the life of people with special needs (caretakers). This platform will also help their families as well as professionals that have many patients under their supervision. More specifically, we present a web application supporting the creation and management of some specific events. With these events, the person that takes care of the caretaker, (caregiver) will be able to monitor him and get informed in case of an emergency. The web application is compatible with most state-of-the-art mobile devices and computers and is able to monitor more than one caretaker. We also have created an android application for smart watches. This android application will run on the smart watch that the caretaker will wear, so that the caregiver can monitor him through the web application. Finally, it is worth mentioning that the applications have been evaluated for their usability by professionals and nonprofessionals users.

# Περίληψη

Σε αυτή την διπλωματική εργασία αναλύουμε τον σχεδιασμό και την υλοποίηση μιας πλατφόρμας η οποία θα διευκολύνει την ζωή ατόμων με ειδικές ανάγκες. (φροντιζόμενοι) Η συγκεκριμένη πλατφόρμα θα βοηθήσει επίσης και τις οικογένειες των ατόμων αυτών καθώς και την δουλειά ανθρώπων οι οποίοι έχουν υπό την επίβλεψή τους άτομα με ειδικές ανάγκες. Πιο συγκεκριμένα, παρουσιάζουμε μία διαδικτυακή εφαρμογή η οποία υποστηρίζει την δημιουργία συγκεκριμένων γεγονότων. Με τον ορισμό αυτών των γεγονότων, το άτομο το οποίο επιβλέπει τον φροντιζόμενο (φροντιστής) θα έχει την δυνατότητα να τον «παρακολουθεί» και να ενημερώνεται σε περίπτωση ανάγκης. Η διαδικτυακή εφαρμογή είναι συμβατή με τις περισσότερες τελευταίας τεχνολογίας συσκευές και ο φροντιστής έχει την δυνατότητα να «παρακολουθεί» έναν η περισσότερους φροντιζόμενους. Επιπλέον δημιουργήσαμε μια εφαρμογή για έξυπνα ρολόγια. Η εφαρμογή αυτή θα τρέχει πάνω στο ρολόι το οποίο θα φοράει ο φροντιζόμενος ώστε ο φροντιστής να μπορεί να τον επιβλέπει μέσω της διαδικτυακής εφαρμογής. Τέλος, σημαντικό θα ήταν να αναφερθεί ότι οι εφαρμογές έχουν αξιολογηθεί για την ευχρηστία τους από επαγγελματίες και μη επαγγελματίες χρήστες.

# Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Antonios Deligiannakis, for his help and his support throughout my thesis.

My thanks also go to Nektarios Gioldasis for his continuous support, his valuable advices, our useful discussions, as well as for coming up with the idea of this application. I would also like to thank Prof. Katerina Mania and Prof. Michail G. Lagoudakis for serving on my thesis committee. My sincerest gratitude also goes to all my friends for the priceless moments we lived together during our studies.

Finally, this thesis would not have been possible without the support and encouragement of my family: my father Tasos, my mother Katerina and my brother Marios. Thank you from the bottom of my heart for everything you have done for me.

# Contents

# List of Tables

## List of Figures

# Chapter 1

## Introduction

In today's society people do not have a lot of free time. They spend too much time on their work or in other activities. For that reason they are not always able to know if all the members of their family are safe and well. As a result they are worried and they never feel relaxed in their daily life. Furthermore there are people with special needs that forget to do important tasks in their day or they do not feel safe. Many of them are concerned that they will not get help in case of an emergency and they get depressed.

In this thesis, we present the design and implementation of a platform that will be able to simplify and improve the life of families that have people with special needs (from now on caretakers) .More specifically; we present a web application that we have designed via which the family members will be able to monitor one or more caretakers. Moreover, we present an android application that will run on the device which the caretaker will wear in order to be connected with the web application. The applications are compatible with most state – of – the art mobile devices, while both have been designed with flexibility and extensibility in mind. This application will bring peace of mind to all family members and will also help people with diseases like dementia or Alzheimer.

The aim of this thesis is to: (a) define the model needed to present all the features of the application; (b) design and implement an infrastructure supporting the model; (c) design and implement a web application supporting the connection with the device that the caretaker will wear; (d) design and implement an android application supporting the connection with the web application.

This thesis is structured as follows:

- Chapter 2 presents the systems and research that are relevant to the topics addressed in this thesis.
- Chapter 3 provides a brief overview of the technologies used for the implementation of the systems.
- Chapter 4 describes the functional specification of the system that has been developed.
- Chapter 5 specifies the model for describing our application.
- Chapter 6 presents the architecture that has been designed and implemented.
- Chapter 7 describes the implementation details of some of the components in more detail, providing some more insight on how specific parts of the system have been used.
- Chapter 8 presents the user interfaces that have been developed for the interaction with the user.
- Chapter 9 summarizes and reviews the presented work and describes some perspectives for future extensions.

# Chapter 2

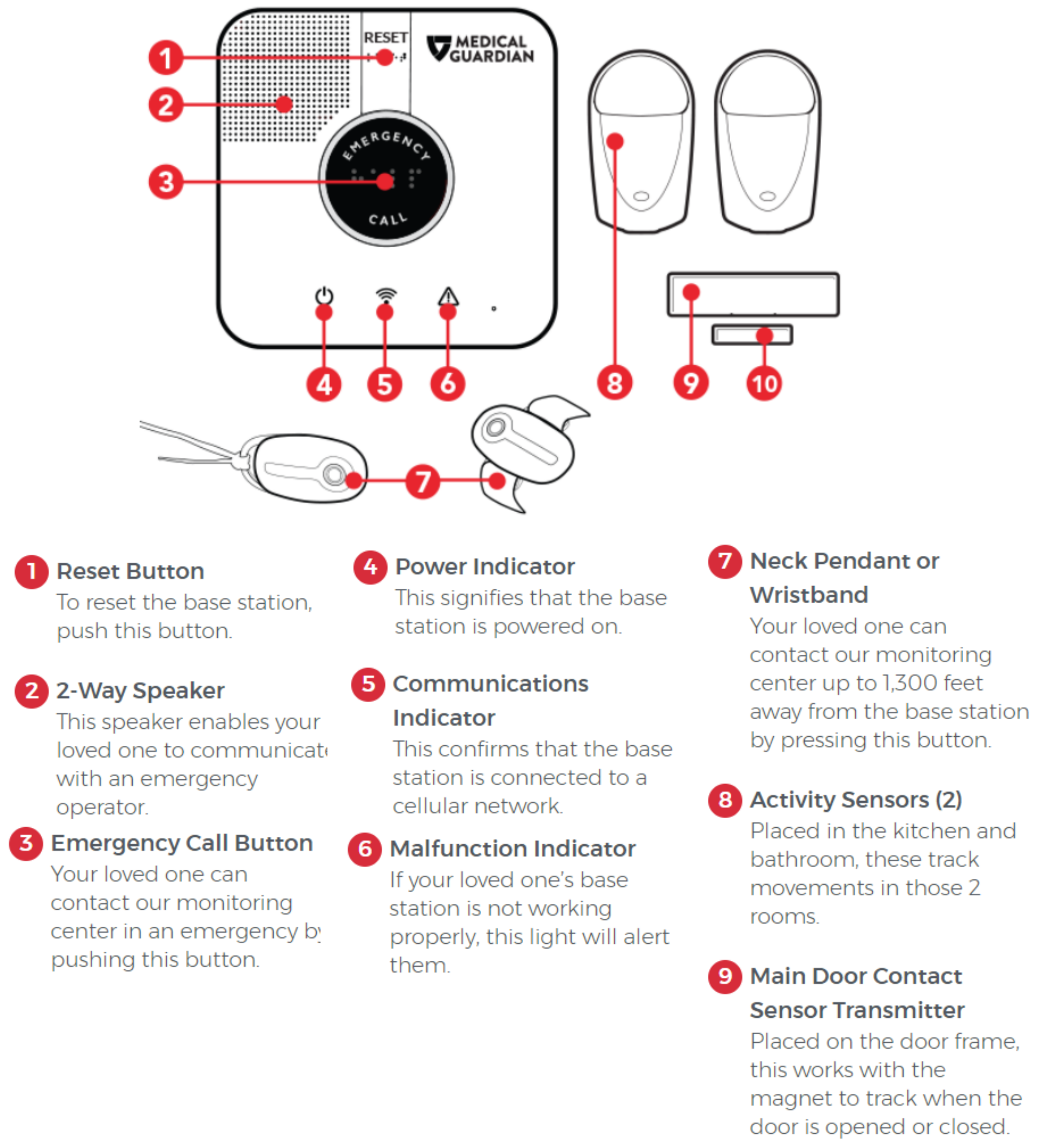
## Related Work

In this chapter, we present tools and applications that are considered relevant to our work. For each of the systems described below, we present their capabilities and discuss their functionality. Additionally, we compare them to the services that we developed, focusing on their strengths and weaknesses. To the extent of our knowledge, there is no platform or tool that combines all these features that our web application provides like the option to monitor the caretaker's heart beats, as well as his blood pressure with just one device. In addition to these features, our platform also supports scheduled reminders that can be repetitive or non repetitive.

## 2.1 Medical Guardian

Medical Guardian is a personal emergency response system. Medical Guardian provides you with a lightweight and water-resistant medical alert button that can be worn around the neck, wrist, or on a belt clip. When pressed, the device immediately sends a wireless signal to the base station (Figure 2.1) and then the base station sends a signal to the Medical Guardians motoring center, alerting them of an emergency. Within moments of pressing your medical alert button, you will be connected to a highly-certified operator. After that the operators call your family when you use your medical alert system. Medical Guardian also provides you with three in-home safety sensors and a Family Guardian application that sends instant notifications right to your email or smart phone. With this web application you will receive a number of Medical Guardian alerts regarding activity and inactivity in the home, including:

- **Panic Alert:** the caretaker has pressed their medical alert help button to contact our monitoring center.
- **Wake and Well:** the caretaker is active in the morning.
- **Non-Activity:** the safety sensors do not detect any activity in the home.
- **Door Left Open:** the front door is left open for an excessive amount of time.
- **Power Loss:** the base station has been disconnected from its power supply.
- **Power Restore:** the base station has been reconnected to its power supply.



**1** **Reset Button**
To reset the base station, push this button.

**2** **2-Way Speaker**
This speaker enables your loved one to communicat with an emergency operator.

**3** **Emergency Call Button**
Your loved one can contact our monitoring center in an emergency b pushing this button.

**4** **Power Indicator**
This signifies that the base station is powered on.

**5** **Communications Indicator**
This confirms that the base station is connected to a cellular network.

**6** **Malfunction Indicator**
If your loved one's base station is not working properly, this light will alert them.

**7** **Neck Pendant or Wristband**
Your loved one can contact our monitoring center up to 1,300 feet away from the base station by pressing this button.

**8** **Activity Sensors (2)**
Placed in the kitchen and bathroom, these track movements in those 2 rooms.

**9** **Main Door Contact Sensor Transmitter**
Placed on the door frame, this works with the magnet to track when the door is opened or closed.

**Figure 2.1:** Medical guardian base station and sensors.

Compared to our work, Medical Guardian application supports mainly indoor activities. On the contrary, we provide a wearable that supports indoor just as outdoor caretakers' activities. With our application you are able to monitor caretakers' heart beats, blood pressure and also get informed when a caretaker enters a danger area, leaves a safety area or he is inactive for some minutes. Also, we give the option to send reminders to a caretaker and get informed when he sees them. Furthermore, through our web application we provide the capability to organize all caretakers with their information and manage their events separately.

## 2.2 Bay Alarm Medical

Bay Alarm Medical provides you with Mobile GPS Device (Figure 2.2) with Help Button which is able to detect a fall and can call for help on its own, but with the added advantage of not needing a medical alert system nearby to work. When the button is pressed, the device immediately sends a wireless signal to Bay Alarm motoring center, alerting them of an emergency. Within moments of pressing your medical alert button, you will be connected to an operator. After that the operators call your family and inform them about the current situation. Bay Alarm Medical also gives you the option to track a caretaker's location with a web application and get notifications via SMS or email if a caretaker leaves the area.

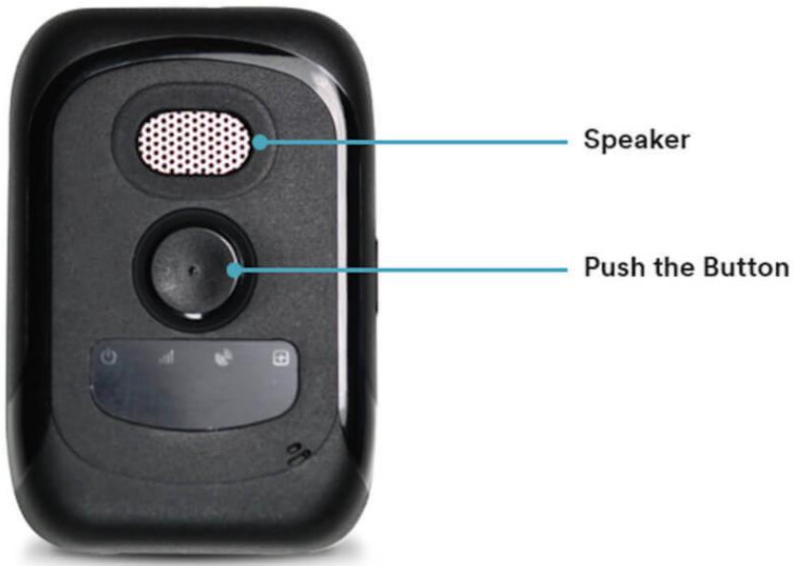


**Figure 2.2:** Bay Alarm Medical Mobile GPS Device.

Compared to Bay Alarm Medical, our application gives you the option to send reminders to a caretaker. Also, a caregiver is notified when a caretaker's heart beats or blood pressure are not normal or if a caretaker is standing still for a significant period of time. Finally, in contrast to the Bay Alarm Medical, our application also gives you the option to define not only safety areas but also areas on the map that are dangerous and a caregiver will be informed if a caretaker enters them.

## 2.3 Medical Alert

Medical Alert is an emergency response system. It provides you with a base station and with an alert bracelet or pendant (Figure 2.3), so in case of emergency a caretaker can push the button on the bracelet to get help directly through the system's base station. Medical Alert devices also have fall detection technology and call an emergency center in case they detect a fall. Furthermore, it includes a mobile application with which a caregiver can contact the caretaker and check the caretaker's location on the map.

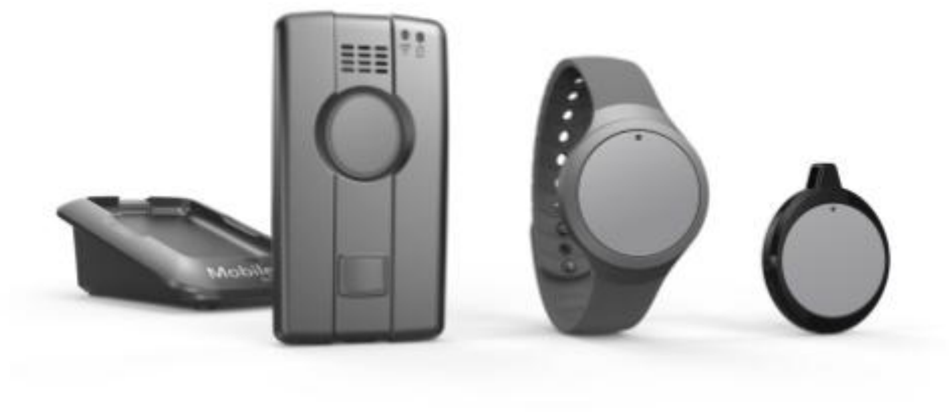


**Figure 2.3:** Medical Alert base station and wearables.

On the contrary, our device is fully independent and the caretaker does not need to carry with him a base station in addition to the wearable. Furthermore, our application is not a mobile application but a web application. This means that the caregiver can use the application not only on his phone but also on his personal computer and laptop.

# Chapter 3

## Background

This chapter presents a quick overview of the standards and technologies used in this thesis. Section 3.1 presents AngularJS, the JavaScript framework that we use to build our client side applications. Section 3.2 presents JavaScript, the scripting language used mainly for the implementation of the client side logic and the interaction with the users, as well as the JavaScript libraries used. Section 3.3 describes Java, the programming language that was used in our server to insert and extract data from our database and also to program our Android application. Section 3.4 presents MySQL, the database system used to save all the data.

## 3.1 AngularJS

**AngularJS** is an open source web application framework for creating RICH Internet Applications (RIAs). It gives you the option to write client side code in a MVC (Model-View-Controller) way, while its data binding and dependency features reduces the code that a developer needs to write. Furthermore, it allows the use of HTML as a template language and lets the developers extend HTML's syntax. Applications using AngularJS are cross-browser compliant and AngularJS automatically handles the javascript code that is suitable for each browser [2].

### RICH Internet Applications (RIAs)

A RICH Internet Application is a Web application which has same features and appearances as a desktop application. A RICH Internet Application requires a browser and a browser plug-in or a virtual machine in order to deliver the user application. The data is handled by the server and the user interface is handled by the client machine.

## AngularJS Core Features

The core features of the AngularJS framework are presented in Figure 3.1 and most of them are described below in detail.

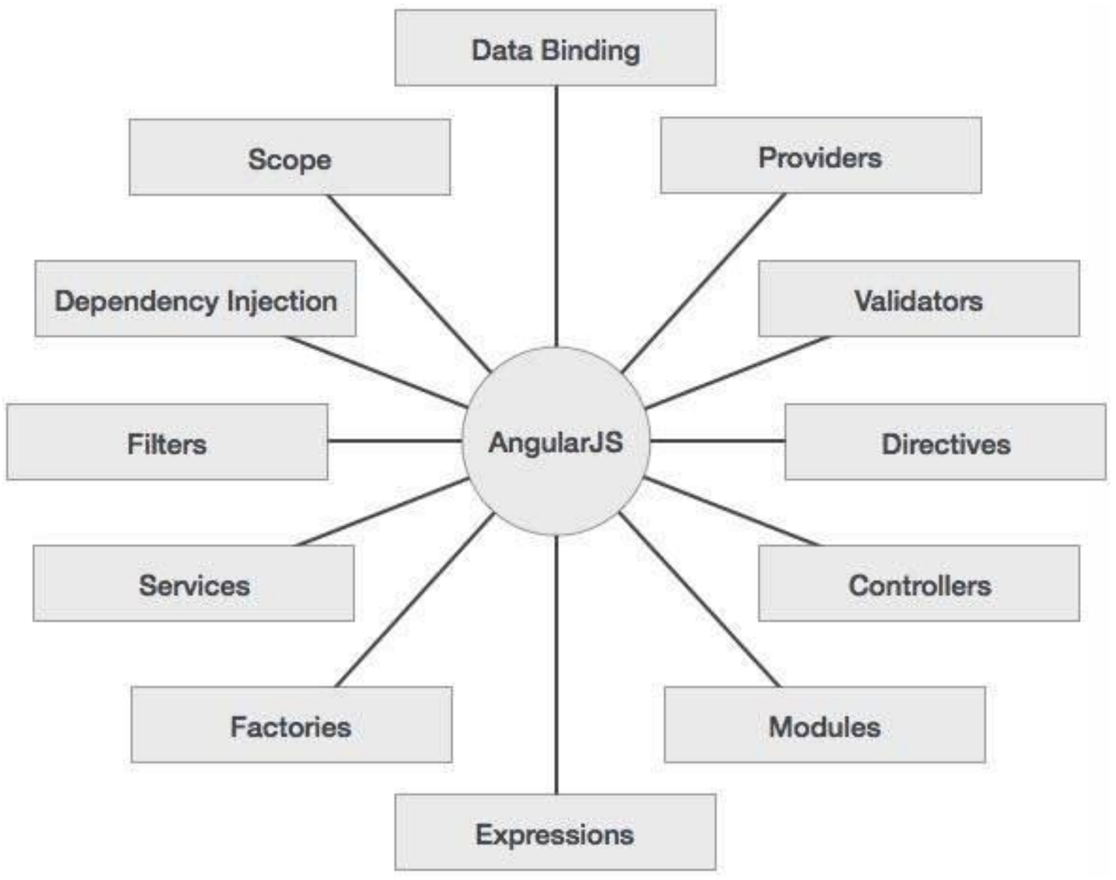


**Figure 3.1:** Most important parts of AngularJS.

## Data-binding

Data-binding in Angular applications is the automatic synchronization of data between the model and view components. The view is every time a projection of the model. Every time the model changes, the view shows the change to the user, and vice versa.

**Scope**

Scope is an object that refers to the application model. It is an execution context for expressions. Scopes can create events and watch expressions.

**Controller**

A Controller is a JavaScript constructor function that is used to enlarge the Angular Scope. When a Controller is attached to the DOM by the ng-controller directive, Angular will create a new Controller object, using the specified Controller's constructor function. A new child scope will be available in a form of a parameter to the Controller's constructor function as $scope.

**Services**

Angular services are substitutable objects that are linked together using Dependency Injection (DI). You can use services to share and organize code at your application. The Angular services are:

• Lazily instantiated - Angular instantiates a service only when a component of the application depends on it.

• Singletons - Each component of the application with is depending on a service gets a reference to the instance which is generated by the service factory.

**Filters**

A filter formats the value of an expression for display to the user. Filters can be used in view templates, services or controllers. AngularJS comes with a variety of built-in filters and it is also very easy to define your own as well if you like.

**Directives**

At a high level, directives are markers on a DOM element that tell AngularJS's HTML compiler to give a specified behavior to that DOM element. It can also transform the DOM element or even its children.

Angular includes a set of these built-in directives, like ngBind, ngClass, and ngModel. As you can create controllers and services, you can also create some of your own directives for Angular to be used.

**Templates**

In Angular, templates are written with HTML that contains Angular-specific attributes and elements. Angular combines the template with information from the model and controller to render the dynamic view that is presented to the user by the browser.

### Model View Controller

Model View Controller (MVC) is a design pattern used to divide an application into three different parts called Model, View and Controller, each one with different and distinct responsibilities.

### Deep Linking

Deep linking allows the developer to encode the state of the application in the URL so that it can be bookmarked. After that the application is able to be restored from the URL to the same state.

### Dependency Injection

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies. The Angular injector subsystem is responsible of creating components, providing them to other components and resolving their dependencies.

## 3.2 Javascript

**JavaScript** is a dynamic computer programming language released by Netscape and Sun Microsystems in 1995. Today it is used on almost all web pages by adding functionality and it is suppoted by all modern browsers (Internet Explorer, Firefox, Chrome, Opera, Safari). Its code can be embedded in an HTML file or called from one or more external files. Also its use is free which means that no licensed is needed to be used as well as all the technologies used to develop our application. It is used to make webpages interactive and provide online programs, including video games. It has an API for working with text, arrays, dates, regular expressions, and basic manipulation of the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded. Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, and in non-web programs such as word processors and PDF software, and in runtime environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets. We used JavaScript within our application to develop the functionality needed for the client-side of our system. The following sections highlight the most important features of JavaScript:

### Dynamic typing

As in most scripting languages, types are associated with values and not with variables. For example, a variable could be bound to a string and then later rebound to a number.

### Run-time evaluation

JavaScript includes an evaluation function that can execute statements provided as strings at run-time.

### First-class functions

Functions in JavaScript are complete objects themselves. They have methods and properties, such as call() and bind(). JavaScript also supports nested functions. A nested function is a function that is defined within another function. It is created each time the outer function is invoked. In addition, each one of the created functions forms a closure: the scope of the outer function, becomes part of the internal state of each inner function object, even after execution of the outer function ends.

### Prototypes

JavaScript uses prototypes where many other languages that are object oriented use specific classes for inheritance. It is possible to simulate many class-based features with prototypes in JavaScript [11].

### Functions as methods

In JavaScript there is no distinction between a function definition and a method definition unlike many object-oriented languages. If the distinction occurs during function calling (when a function is called as a method of an object), the function's local this keyword is bound to that object for that invocation.

### Run-time environment

JavaScript typically relies on a run-time environment to provide objects and methods by which scripts are able to interact with the environment. It also relies on the run-time environment to include or import scripts. This is not a language feature, but it is common in most JavaScript implementations.

### Variadic functions

An indistinct number of parameters can be passed to a function. The function can access them through formal parameters. It can also access them through the local arguments object. Array and object literals like many scripting languages, arrays and objects (associative arrays in other languages) can each be created with a succinct shortcut syntax. In fact, these literals form the basis of the JSON data format.

### Regular expressions

JavaScript also supports regular expressions, which provide a brief and powerful syntax for text manipulation that is more sophisticated than the built-in string functions.

## 3.3 Java

**Java** is a widespread computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers run Java code on all platforms that support Java without the need for recompilation. Java applications are typically compiled to byte code that can run on any Java virtual machine (JVM) regardless the architecture of the computer. As of 2016, Java is one of the most popular programming languages in use and particularly for client-server web applications.

### Concurrent computing

Concurrent computing is a form of computing in which several computations are executed during overlapping time periods, concurrently, instead of sequentially (one completing before the next starts). This means that a computation can advance without waiting for all other computations to complete.

### Class-based programming

Class-bassed programming, or as it is called more commonly class-orientation, is a style of object-oriented programming. In class-bassed programming inheritance is achieved by defining classes of objects, as opposed to the objects themselves.

**Object – oriented programming**

Object-oriented programming is a programming language in which developers define not only the type of data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both functions and data. Moreover, developers are able to create relationships between objects. For example, objects can inherit characteristics from other objects.

**Use of Java outside Java platform**

The Java programming language requires a software platform in order for compiled programs to be executed. Oracle supplies the Java platform for use with Java. One very popular software platform is the Android SDK which is used primarily for developing Android applications.

**Android**

Android is a mobile operating system developed by Google. It is used by several smartphones, smartwatches and tablets. The Android operating system is bases on the Linux kernel and it is open source which means that developers can modify and customize the operating system for each phone. Developers can create programs for Android using the free Android software developer kit. Moreover, Android programs are written in Java and run through a Java virtual machine JVM that is optimized for mobile devises.

## 3.4 MySQL

**MySQL** is a relational database management system (RDBMS) that runs as a server providing access to many users to a number of databases. In MySQL the beginning of the name My comes from the daughter of the finnish developer. SQL was at first developed to work on data in databases that follow the relational model. It is a programming language for querying, managing and modifying data. MySQL is one of the most common open source database tools. It is considered an easy and reliable program compared to other database software and offers many different programs that are database related.

## What is a database?

Database is a structured set of data that stores organized information. Most of the databases contain multiple tables. Each one of these tables may include several different fields. For example a database of a company may contain tables for employees, products, financial records. Each of these tables would have different files that are relevant to the information stored in the table.

A database consists of both data and metadata. Metadata is data that describes the structure of the data in a database. A database can come in all sizes, from a few records to hundreds or millions of records.

## Index Support

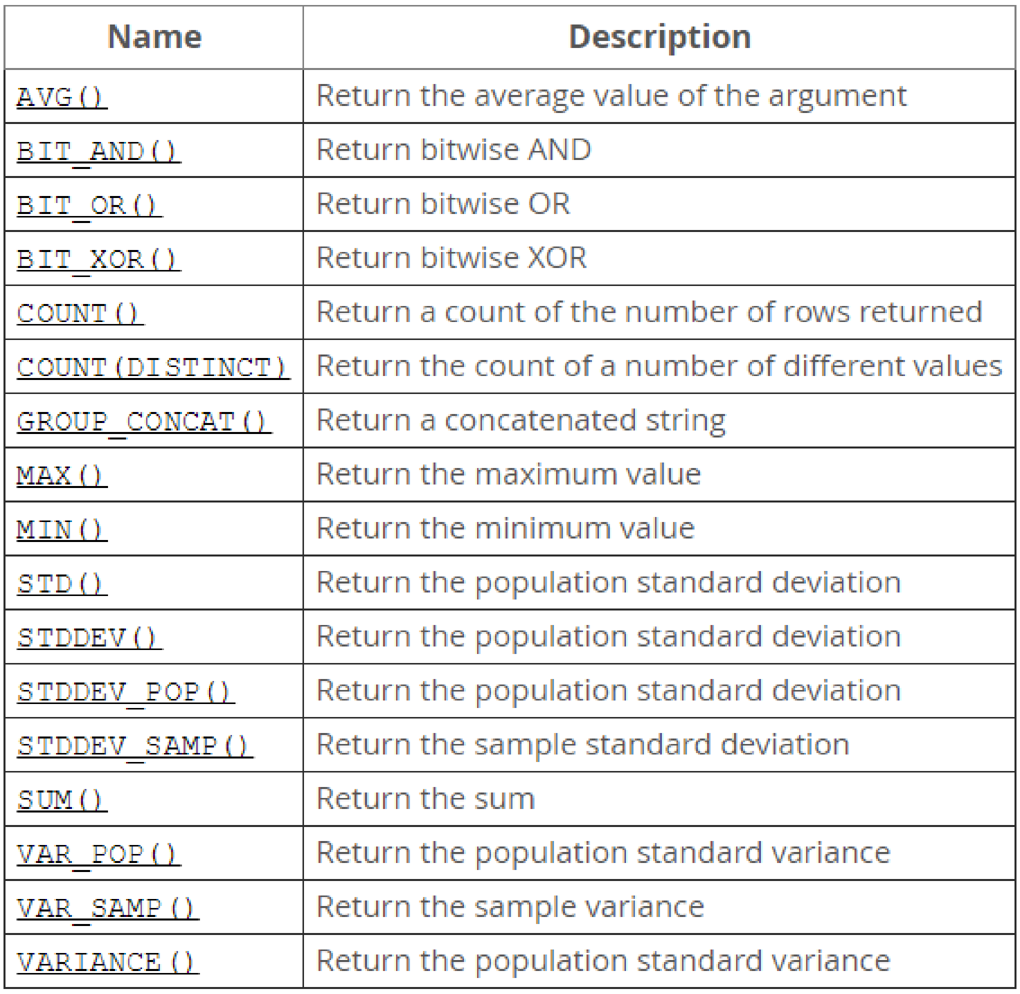
Indexes are used to find rows with specific column values quickly. Without the use of an index, MySQL has to begin with the first row and then read through the entire table to find specific rows. The larger the table, the more this process costs. If the table has an index for the columns in the question then MySQL can quickly find the position needed in the data file without having to look at all the data in the database. This is obviously much faster and more efficient than reading every row sequentially.

## Querying

In MySQL a query targets a specific collection of data in the database. Queries specify criteria or conditions that identify the data that is returned to the users. If you want, you can modify queries to apply limits, skips, and sort orders.

## Aggregation

Aggregations are some operations that process data records and return computed results. MySQL provides the user with a variety of aggregation operations (Figure 3.2) that process and perform calculations on the data sets. Group aggregate functions that operate on sets of values in MySQL are described below:



| Name | Description |
|---|---|
| AVG() | Return the average value of the argument |
| BIT_AND() | Return bitwise AND |
| BIT_OR() | Return bitwise OR |
| BIT_XOR() | Return bitwise XOR |
| COUNT() | Return a count of the number of rows returned |
| COUNT(DISTINCT) | Return the count of a number of different values |
| GROUP_CONCAT() | Return a concatenated string |
| MAX() | Return the maximum value |
| MIN() | Return the minimum value |
| STD() | Return the population standard deviation |
| STDDEV() | Return the population standard deviation |
| STDDEV_POP() | Return the population standard deviation |
| STDDEV_SAMP() | Return the sample standard deviation |
| SUM() | Return the sum |
| VAR_POP() | Return the population standard variance |
| VAR_SAMP() | Return the sample variance |
| VARIANCE() | Return the population standard variance |

**Figure 3.2:** MySQL aggregation functions.

# Chapter 4

## Functional Specification

This chapter describes the functional specification of the platform and tools that have been developed for the creation, management and editing of the events in our application. Section 4.1 specifies the system's stakeholders and their role in our system, while Section 4.2 discusses the technical requirements that had to be provided. Finally, Section 4.3 provides the functionality that has to be provided by our system in the form of use cases.

## 4.1 Stakeholders

A stakeholder is someone or something that has an interest in the behavior of the use case. Our platform is targeted for everyone who wants to monitor people with special needs and know if they are safe and well. Also, our platform is targeted for everyone who wants to feel safe in his daily life and get help when it is needed. It is planned to be used by both non-experts and professionals, either just for motoring a family member, or for providing a tool in the hands of a professional making him capable of monitoring his patients. For those scenarios, the system's stakeholders are:

- The Caretaker who will wear the smart watch with the installed android application.
- The Caregiver who can be a family member that will use the web application to monitor his loved ones.
- Professionals (Caregivers) who will use our system to monitor their patients (Caretakers) and provide them with the necessary help.

## 4.2 Technical Requirements

This section discusses the technical requirements that were identified and set for the development of the model and the systems. Section 4.2.1 describes the requirements of the Web application, while Section 4.2.2 describes the requirements of the android application.

### 4.2.1 Web application

The application that the caregiver will use needs to be a desktop application due to the extended functionality that should offer to the caretaker. More specifically it should be web based in order to be easily accessible and to have minor setup requirements. As a web application it should be compatible with the state-of-the-art standards and its user interface needs to be responsive and operate smoothly in any screen and browser. Using this tool the users should be able to create and manage their caretakers and events. In more detail, the authoring tool should:

- Be compatible with state-of-the-art web standards.
- Have responsive user interface.
- Be able to be used not only by professionals but also by non experts.
- Support personalization for making the user feel comfortable with the user interface.
- Allow the creation and management of events for monitoring the caretakers.
- Have easy access and switching between the caretakers.

### 4.2.2 Android application

The caretaker application needs to be a mobile application due to the fact that the caretaker should have it always on him. A problem with this requirement is the fact that different devices have different software. Some of these are using completely different programming languages. To this end, we decided to base our application on android which will support the majority of smart watches. The features of the android application should include:

- Compatibility with most state-of-the-art mobile devices.
- Beautiful and usable interface.
- Support of reminders.
- Ability to save data when device is offline and send it when it is back online.
- Ability to run in the background.
- Compatibility with our web application, in the sense that it should process data that has been added by the web application.
- Ability to use the hardware tools (GPS, Accelerometer etc.) of the mobile device.

## 4.3 Use Cases

A use case describes the system's behavior under various conditions as the system responds to a request from the stakeholders, called primary actor. The stakeholder interacts with the system to accomplish a goal. The system responds always protecting the interests of the stakeholders.

This section describes the system's behavior under its interaction with the stakeholders. For further understanding we present some of the use cases in form of use case diagrams. Figures 4.1, 4.2 and 4.4 present the use cases of the Caregiver, also Figure 4.5 presents the use cases of the Caretaker.

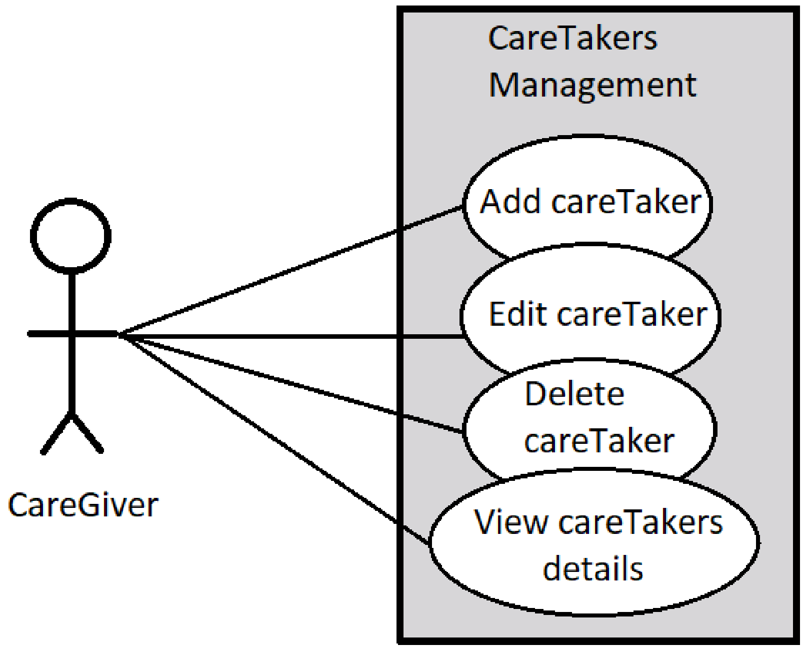


**Figure 4.1:** Web application – User Management.

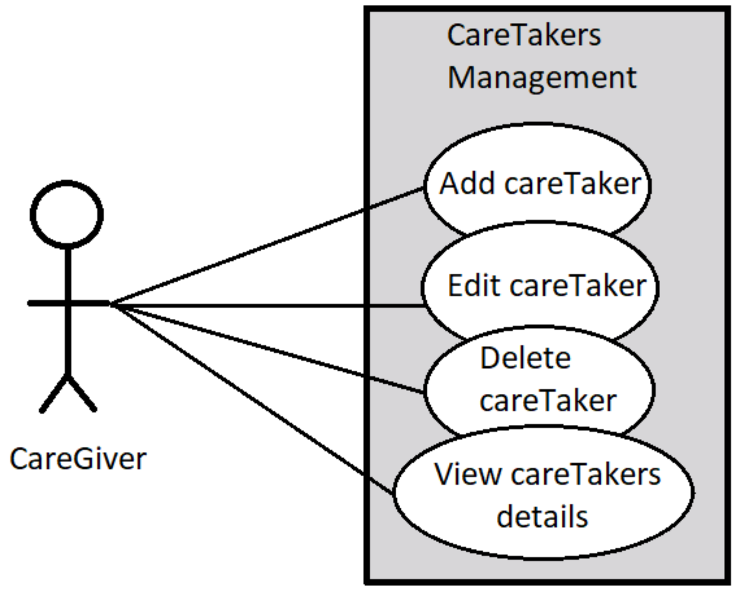


**Figure 4.2:** Web application – CareTakers' Management

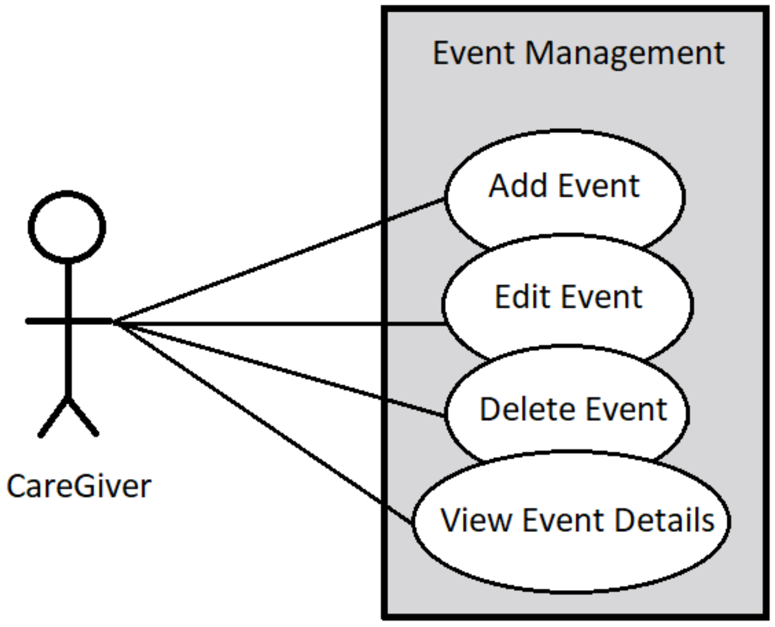


**Figure 4.3:** Web application – Event Management.

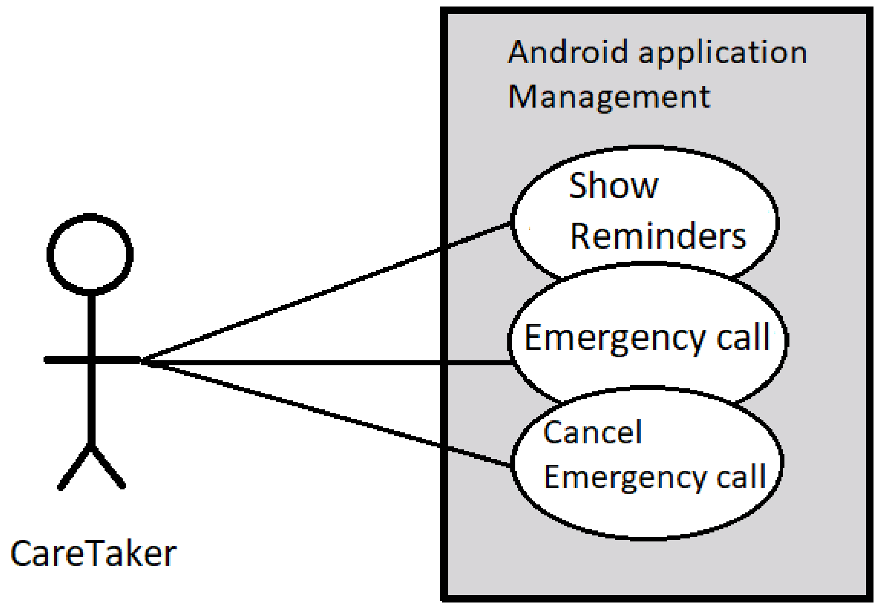


**Figure 4.4:** Android application – Application Management.

**Table 4.1:** Use case 1: “Create Account”.

| Use Case 1: “Create Account” | | |
|---|---|---|
| **Goal in Context** | The user wants to use the services that the system provides to registered users. | |
| **Preconditions** | The user is not logged in the system. | |
| **Success End Condition** | The user has created an account successfully and is able to use the services that the system provides to registered users. | |
| **Failed End Condition** | The user could not create a new account. | |
| **Primary, Secondary Actors** | Caregiver, System. | |
| **Trigger** | 1. The user wants to create an account. | |
| **Description** | **Step** | **Action** |
| | 1 | The system displays the Sign Up form. |
| | 2 | The user fills in the required information about his new account. |
| | 3 | The system validates the user’s input. |
| | 4 | The system will create a new account for the user. |
| | 5 | The user is able to use the services of the system and system displays the first screen to the user. |

**Table 4.2:** Use case 2: “Log In”.

| Use Case 2: ”Log in” | |
|---|---|
| **Goal in Context** | The user wants to use the services that the system provides to registered users. |
| **Preconditions** | The user is registered in the system. |
| **Success End Condition** | The user has been identified and is able to use the services that the system provides to registered users. |
| **Failed End Condition** | The user cannot use the services that the system provides to registered users. |
| **Primary, Secondary Actors** | Caregiver, System |
| **Trigger** | The user selects to enter in his account. |
| **Description** | **Step** / **Action**<br>1 — The user inputs his name.<br>2 — The user inputs his password.<br>3 — The user presses the log in button.<br>4 — The system checks if the fields are filled.<br>5 — The system will search if the username and password are correct.<br>6 — The system will log in the user to his account.<br>7 — The user is able to use the services of the system. |
| **Extensions** | **Step** / **Branching Action**<br>4a — The fields are not filled.<br>4a1 The system indicates that the fields are not filled, asks for corrections and returns to step 1<br>5a — The user is not registered.<br>5a1. The system informs the user and indicates him to register. |

**Table 4.3:** Use case 3: “Log out”.

| Use Case 3: “Log out” | | |
|---|---|---|
| **Goal in Context** | The user wants to be logged out from the system. | |
| **Preconditions** | The user is already logged in the system. | |
| **Success End Condition** | The user is logged out successfully from the system. | |
| **Failed End Condition** | The system could not log the user out of the system. | |
| **Primary, Secondary Actors** | Caregiver, System | |
| **Trigger** | The user selects to log out from the system. | |
| **Description** | **Step** | **Action** |
| | 1 | The system logs out the user. |
| | 2 | The system displays the log in screen. |

**Table 4.4:** Use case 4: “Add a CareTaker”.

| Use Case 4: “Add a CareTaker” | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to monitor a new smart watch and add a new CareTaker to the system. | |
| **Preconditions** | The user has the id code of the device–smart watch. | |
| **Success End Condition** | The new device has been added and the user is able to monitor the caretaker. | |
| **Failed End Condition** | The user is not able to monitor the new caretaker. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System | |
| **Trigger** | The user selects to add a new caretaker. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to add a new caretaker. |
| | 2 | The system asks him for an id code and for the caretaker’s information. |
| | 3 | User inputs the id code of the device and the caretaker’s information to the system. |
| | 4 | The system checks if the id code is correct. |
| | 5 | The system connects with the smart watch. |
| | 6 | The user is able to monitor the new device. |
| **Extensions** | **Step** | **Branching Action** |
| | 4a | The field are not filled correctly |
| | | 4a1. The system indicates that the fields are not correct, asks for corrections and returns to step 2 |

**Table 4.5:** Use case 5: “Edit a CareTaker”.

| **Use Case 5: “Edit a CareTaker”** | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to edit a caretaker. | |
| **Preconditions** | There is a saved caretaker to be edited. | |
| **Success End Condition** | The caretaker has been edited successfully. | |
| **Failed End Condition** | The caretaker has not been edited successfully. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System | |
| **Trigger** | The user selects to edit a caretaker. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to edit a caretaker. |
| | 2 | The user changes the caretaker’s information. |
| | 3 | The user presses the save button. |
| | 4 | The system saves the new information. |
| **Extensions** | **Step** | **Branching Action** |
| | 3a | Some information fields are not filled. |
| | | 3a1. The system indicates that all the fields are not filled correctly and returns to step 2. |

**Table 4.6:** Use case 6: “Delete a CareTaker”.

| **Use Case 6: “Delete a CareTaker”** | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to delete a saved caretaker. | |
| **Preconditions** | There is a saved caretaker to be deleted. | |
| **Success End Condition** | Caretaker has been deleted. | |
| **Failed End Condition** | Caretaker has not been deleted. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker , System. | |
| **Trigger** | The user selects to delete a caretaker. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to delete a caretaker. |
| | 2 | The user presses the delete button. |
| | 3 | The system asks user for confirmation. |
| | 4 | The user confirms. |
| | 5 | The system deletes the caretaker. |

**Table 4.7:** Use case 7: “Add Region”.

| **Use Case 7: “Add Region”** | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to add a new region to the account. | |
| **Preconditions** | The area has been selected and the information fields are filled. | |
| **Success End Condition** | The area has been saved and the user will be informed if the caretaker leaves or enters the area. | |
| **Failed End Condition** | The user will not be informed if the caretaker leaves or enters the area. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System | |
| **Trigger** | The user selects to add a new region. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to add a new region. |
| | 2 | The user enters the coordinates of the region on the displayed map, region name and other information. |
| | 3 | The caregiver presses the save button. |
| | 4 | The system saves the new region. |
| | 5 | The system sends the new region’s data to the device - smart watch. |
| **Extensions** | **Step** | **Branching Action** |
| | 3a | The area is not selected or some info fields are not filled. |
| | | 3a1. The system indicates that all the fields are not filled correctly or that the region has not been selected asks the user to select it and returns to step 2. |

**Table 4.8:** Use case 8: “Edit Region”.

| Use Case 8: “Edit Region” | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to edit a region. | |
| **Preconditions** | There is a saved region to be edited. | |
| **Success End Condition** | The region has been edited successfully. | |
| **Failed End Condition** | The region has not been edited successfully. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System | |
| **Trigger** | The user selects to edit a region. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to edit a region. |
| | 2 | The user changes the coordinates of the region or other information. |
| | 3 | The user presses the save button. |
| | 4 | The system saves the new information. |
| | 5 | The system sends the region’s new data to the device - smart watch. |
| **Extensions** | **Step** | **Branching Action** |
| | 3a | The area is not selected or some info fields are not filled |
| | | 3a1. The system indicates that all the fields are not filled correctly or that the region has not been selected , asks the user to select it and returns to step 2. |

**Table 4.9:** Use case 9: “Delete Region”.

| Use Case 9 : “Delete Region” | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to delete a saved region. | |
| **Preconditions** | There are saved regions to be deleted. | |
| **Success End Condition** | The region has been deleted. | |
| **Failed End Condition** | The region has not been deleted. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System. | |
| **Trigger** | The user selects to delete a region. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to delete a region. |
| | 2 | The system asks user for confirmation. |
| | 3 | The user confirms. |
| | 4 | The system deletes the region. |
| | 5 | The system informs the device about the deleted region. |

**Table 4.10:** Use case 10: “Add Reminder”.

| Use Case 10: “Add Reminder” | |
|---|---|
| **Goal in Context** | The user indicates that he wants to add a new reminder to the account. |
| **Preconditions** | The time, title , description and all the required fields of the reminder have been filled. |
| **Success End Condition** | The reminder has been saved and the caretaker will be informed at the given date and time. |
| **Failed End Condition** | The Caretaker will not be informed. |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System |
| **Trigger** | The user selects to add a new reminder. |
| **Description** | **Step** / **Action**<br>1 – The user indicates that he wants to add a new reminder.<br>2 – The user enters the date, title, description and all the required fields of the reminder.<br>3 – The system checks if the fields are filled.<br>4 – The system saves the new reminder.<br>5 – The system sends the new reminder to the device. |
| **Extensions** | **Step** / **Branching Action**<br>3a – All the fields of the reminder are not filled.<br>3a1. The system indicates that all the fields of the reminder are not filled, asks the user to fill them and it and returns to step 2. |

**Table 4.11:** Use case 11: “Edit Reminder”.

| Use Case 11: “Edit Reminder” | | | |
|---|---|---|---|
| Goal in Context | The user indicates that he wants to edit a reminder. | | |
| Preconditions | There is a saved reminder to be edited. | | |
| Success End Condition | The reminder has been edited successfully. | | |
| Failed End Condition | The reminder has not been edited successfully. | | |
| Primary, Secondary Actors | Caregiver, Caretaker, System | | |
| Trigger | The user selects to edit a reminder. | | |
| Description | **Step** | **Action** | |
| | 1 | The caregiver indicates that he wants to edit a reminder. | |
| | 2 | The caregiver changes reminder’s information. | |
| | 3 | The caregiver presses the save button. | |
| | 4 | The system saves the new information. | |
| | 5 | The system sends reminder’s new data to the smart watch. | |
| Extensions | **Step** | **Branching Action** | |
| | 3a | Some information fields are not filled. | |
| | | 3a1. The system indicates that all the fields are not filled correctly and returns to step 2. | |

**Table 4.12:** Use case 12: “Delete Reminder”.

| Use Case 12: “Delete Reminder” | | |
|---|---|---|
| Goal in Context | The user indicates that he wants to delete a saved reminder. | |
| Preconditions | There is a reminder to be deleted. | |
| Success End Condition | The reminder has been deleted. | |
| Failed End Condition | The reminder has not been deleted. | |
| Primary, Secondary Actors | Caregiver, Caretaker, System. | |
| Trigger | The user selects to delete a reminder. | |
| Description | **Step** | **Action** |
| | 1 | The user indicates that he wants to delete a reminder. |
| | 2 | The system asks user for confirmation. |
| | 3 | The user confirms. |
| | 4 | The system deletes the reminder. |
| | 5 | The system informs the device about the deleted reminder. |

**Table 4.13:** Use case 13: “Add Blood Pressure event”.

| Use Case 13: “Add Blood Pressure event” | |
|---|---|
| **Goal in Context** | The user indicates that he wants to add a new Blood pressure event to the account. |
| **Preconditions** | The time, title and all the required fields of the Blood pressure event have been filled. |
| **Success End Condition** | The Blood pressure event has been saved. |
| **Failed End Condition** | The Blood pressure event has not been saved. |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System |
| **Trigger** | The user selects to add a new Blood pressure event. |
| **Description** | **Step** / **Action** |
| | 1 — The user indicates that he wants to add a new Blood pressure event. |
| | 2 — The user selects the Blood pressure event from the events tab. |
| | 3 — The user enters the time, title and all the required fields of the Blood pressure event. |
| | 4 — The system checks if the fields are filled. |
| | 5 — The system saves the new Blood pressure event. |
| | 6 — The system sends the new Blood pressure event to the device. |
| **Extensions** | **Step** / **Branching Action** |
| | 3a — All the fields of the Blood pressure event are not filled. |
| | 3a1. The system indicates that all the fields of the Blood pressure event are not filled, asks the user to fill them and it and returns to step 2. |

**Table 4.14:** Use case 14: “Edit Blood Pressure event”.

| Use Case 14: “Edit Blood Pressure event” | | |
|---|---|---|
| **Goal in Context** | User indicates that he wants to edit a Blood pressure event | |
| **Preconditions** | There is a saved Blood pressure event to be edited. | |
| **Success End Condition** | The Blood pressure event has been edited successfully. | |
| **Failed End Condition** | The Blood pressure event has not been edited successfully. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System. | |
| **Trigger** | The user selects to edit a Blood pressure event. | |
| **Description** | **Step** | **Action** |
| | 1 | The caregiver indicates that he wants to edit a event |
| | 2 | The user selects the Blood pressure event that he wants to edit. |
| | 3 | The caregiver changes the event’s information. |
| | 4 | The caregiver presses the save button. |
| | 5 | The system saves the new information. |
| | 6 | The system sends the event’s new data to the smart watch. |
| **Extensions** | **Step** | **Branching Action** |
| | 3a | Some of the information fields are not filled. |
| | | 3a1. The system indicates that all the fields are not filled correctly and returns to step 2. |

**Table 4.15:** Use case 15: “Delete Blood Pressure event”.

| Use Case 15: “Delete Blood Pressure event” | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to delete a saved Blood pressure event. | |
| **Preconditions** | There is a Blood pressure event to be deleted. | |
| **Success End Condition** | The Blood pressure event has been deleted. | |
| **Failed End Condition** | The Blood pressure event has not been deleted. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System. | |
| **Trigger** | The user selects to delete a Blood pressure event. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to delete a Blood pressure event. |
| | 2 | The user presses the delete button. |
| | 3 | The system asks user for confirmation. |
| | 4 | The user confirms. |
| | 5 | The system deletes the Blood pressure event. |
| | 6 | The system informs device about the deleted event. |

**Table 4.16:** Use case 16: “Add Heart Beat event”.

| Use Case 16: “Add Heart Beat rule” | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to add a new Heart Beat rule to the account. | |
| **Preconditions** | The time, title and all the required fields of the Heart Beat rule have been filled. | |
| **Success End Condition** | The Heart Beat rule has been saved. | |
| **Failed End Condition** | The Heart Beat rule has not been saved. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System | |
| **Trigger** | The user selects to add a new rule. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to add a new Heart Beat event. |
| | 2 | The user selects the Heart Beat event from the events tab. |
| | 3 | The user enters the time, title and all the required fields of the Heart Beat event. |
| | 4 | The system checks if the fields are filled. |
| | 5 | The system saves the new Heart Beat event. |
| | 6 | The system sends the new Heart Beat event to the device. |
| **Extensions** | **Step** | **Branching Action** |
| | 3a | All the fields of the Heart Beat event are not filled. |
| | | 3a1. The system indicates that all the fields of the Heart Beat event are not filled, asks the user to fill them and it and returns to step 2. |

**Table 4.17:** Use case 17: “Edit Heart Beat event”.

| Use Case 17: “Edit Heart Beat event” | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to edit a Heart Beat event. | |
| **Preconditions** | There is a saved Heart Beat event to be edited. | |
| **Success End Condition** | The Heart Beat event has been edited successfully. | |
| **Failed End Condition** | The Heart Beat event has not been edited successfully. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System. | |
| **Trigger** | The user selects to edit a Heart Beat event. | |
| **Description** | **Step** | **Action** |
| | 1 | The caregiver indicates that he wants to edit a Heart Beat event |
| | 2 | The user selects the Heart Beat of event that he wants to edit. |
| | 3 | The caregiver changes the event’s information. |
| | 4 | The caregiver presses the save button. |
| | 5 | The system saves the new information. |
| | 6 | The system sends the Heart Beat event’s new data to the smart watch. |
| **Extensions** | **Step** | **Branching Action** |
| | 3a | Some of the information fields are not filled. |
| | | 3a1. The system indicates that all the fields are not filled correctly and returns to step 2. |

**Table 4.18:** Use case 18: “Delete Heart Beat event”.

| Use Case 18: “Delete Heart Beat event” | | |
|---|---|---|
| **Goal in Context** | User indicates that he wants to delete a saved Heart Beat event | |
| **Preconditions** | There is a Heart Beat event to be deleted. | |
| **Success End Condition** | The Heart Beat event has been deleted. | |
| **Failed End Condition** | The Heart Beat event has not been deleted. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System. | |
| **Trigger** | The user selects to delete a Heart Beat event. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to delete a Heart Beat event. |
| | 2 | The user presses the delete button. |
| | 3 | The system asks user for confirmation. |
| | 4 | The user confirms. |
| | 5 | The system deletes the Heart Beat event. |
| | 6 | The system informs device about the deleted event. |

**Table 4.19:** Use case 19: “Add Immobility event”.

| Use Case 19: “Add Immobility event” | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to add a new Immobility event to the account. | |
| **Preconditions** | The time, title and all the required fields of the Immobility event have been filled. | |
| **Success End Condition** | The Immobility event has been saved. | |
| **Failed End Condition** | The Immobility event has not been saved. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System | |
| **Trigger** | The user selects to add a new Immobility event. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to add a new Immobility event. |
| | 2 | The user selects the Immobility event from the events tab. |
| | 3 | The user enters the time, title and all the required fields of the event. |
| | 4 | The system checks if the fields are filled. |
| | 5 | The system saves the new Immobility event. |
| | 6 | The system sends the new Immobility event to the device. |
| **Extensions** | **Step** | **Branching Action** |
| | 3a | All the fields of the Immobility event are not filled. |
| | | 3a1. The system indicates that all the fields of the Immobility event are not filled, asks the user to fill them and it and returns to step 2. |

**Table 4.20:** Use case 20: "Edit Immobility event".

| Use Case 20: "Edit Immobility event" | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to edit an Immobility event. | |
| **Preconditions** | There is a saved Immobility event to be edited. | |
| **Success End Condition** | The Immobility event has been edited successfully. | |
| **Failed End Condition** | The Immobility event has not been edited successfully. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System. | |
| **Trigger** | The user selects to edit an Immobility event. | |
| **Description** | **Step** | **Action** |
| | 1 | The caregiver indicates that he wants to edit an Immobility event |
| | 2 | The user selects the Immobility event that he wants to edit. |
| | 3 | The caregiver changes the event's information. |
| | 4 | The caregiver presses the save button. |
| | 5 | The system saves the new information. |
| | 6 | The system sends the Immobility event's new data to the smart watch. |
| **Extensions** | **Step** | **Branching Action** |
| | 3a | Some of the information fields are not filled. |
| | | 3a1. The system indicates that all the fields are not filled correctly and returns to step 2. |

**Table 4.21:** Use case 21: "Delete Immobility event".

| Use Case 21: "Delete Immobility event" | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to delete a saved Immobility event. | |
| **Preconditions** | There is an Immobility event to be deleted. | |
| **Success End Condition** | The Immobility event has been deleted. | |
| **Failed End Condition** | The Immobility event has not been deleted. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System. | |
| **Trigger** | The user selects to delete an Immobility event. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to delete an Immobility event. |
| | 2 | The user presses the delete button. |
| | 3 | The system asks user for confirmation. |
| | 4 | The user confirms. |
| | 5 | The system deletes the Immobility event. |
| | 6 | The system informs the device about the deleted event. |

**Table 4.22:** Use case 22: “CareTaker Location”.

| Use Case 22: “Caretaker Location” | | |
|---|---|---|
| **Goal in Context** | The user indicates that he wants to see a caretaker’s current location. | |
| **Preconditions** | The caretaker is online. | |
| **Success End Condition** | The user is able to see the caretaker’s location in the map. | |
| **Failed End Condition** | The user is not able to see the caretaker’s current location in the map. | |
| **Primary, Secondary Actors** | Caregiver, Caretaker, System. | |
| **Trigger** | The user selects to see a caretaker’s location. | |
| **Description** | **Step** | **Action** |
| | 1 | The user indicates that he wants to see a caretaker’s location. |
| | 2 | The system will search for the caretaker’s coordinates. |
| | 3 | The system will display a map and the caretaker’s location in it. |
| **Extensions** | **Step** | **Branching Action** |
| | 2a | The Caretaker is not online. |
| | | 2a1. The system will display to the user the last coordinates of the caretaker that have been saved. |

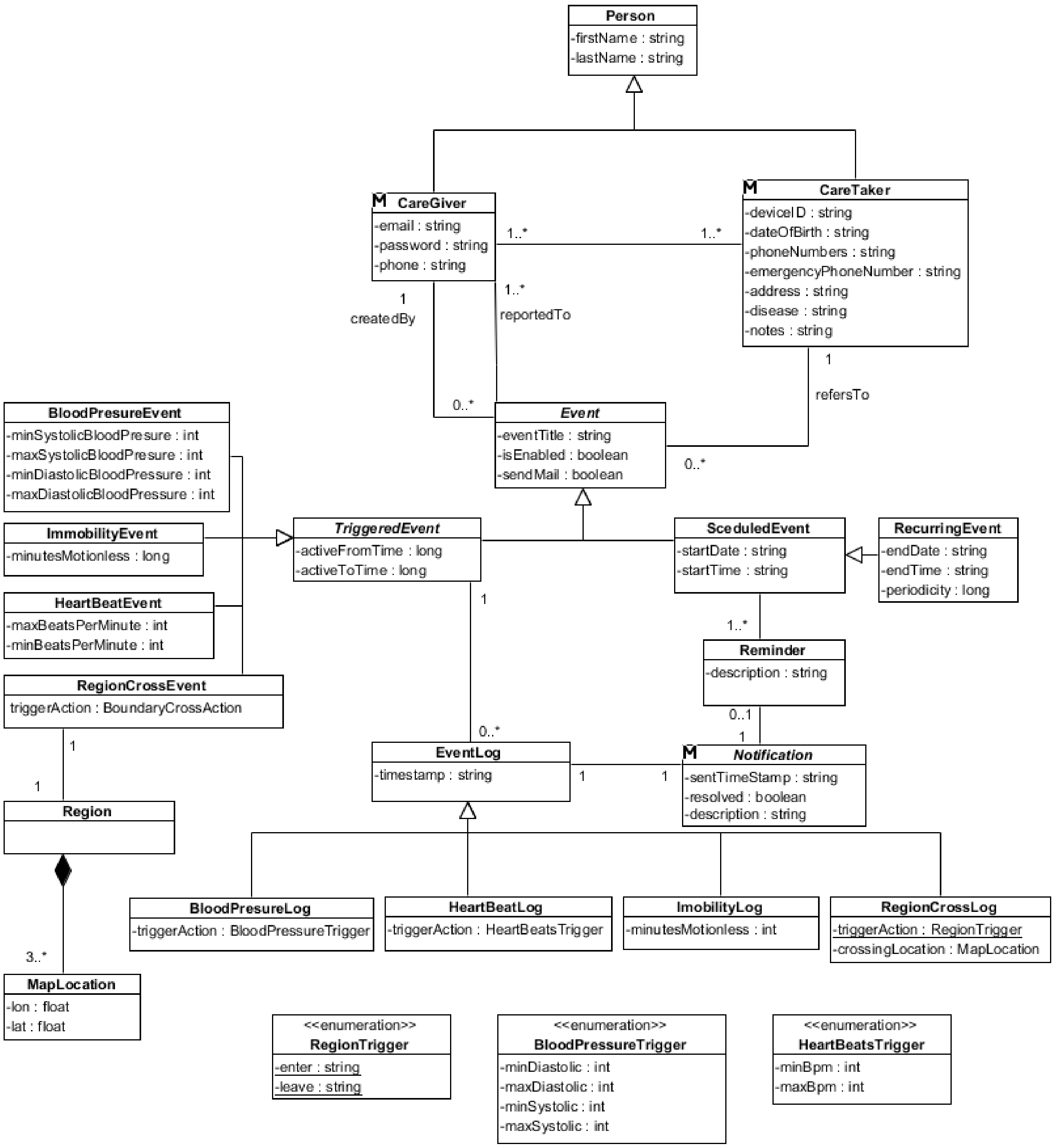
# Chapter 5

## Model

This chapter presents the model that has been developed to support our system. Section 5.1 describes the model's basic idea and structure.

## 5.1 Specification

The model developed and supported by our system has been designed with extreme detail, so as to be able to describe it as correctly as possible. To do this, we identified that an event should be divided to scheduled events that are defined from the caregiver and triggered events that are triggered from the caretaker. Also a triggered event can be blood pressure, immobility, heartbeat or region cross event. Figure 5.1 presents the identified model entities in the form of a class diagram.


Person
-firstName : string
-lastName : string
CareGiver
-email : string
-password : string
-phone : string
CareTaker
-deviceID : string
-dateOfBirth : string
-phoneNumbers : string
-emergencyPhoneNumber : string
-address : string
-disease : string
-notes : string
1..*
1..*
1
createdBy
1..*
reportedTo
1
refersTo
0..*
0..*
Event
-eventTitle : string
-isEnabled : boolean
-sendMail : boolean
BloodPresureEvent
-minSystolicBloodPresure : int
-maxSystolicBloodPresure : int
-minDiastolicBloodPressure : int
-maxDiastolicBloodPressure : int
ImmobilityEvent
-minutesMotionless : long
HeartBeatEvent
-maxBeatsPerMinute : int
-minBeatsPerMinute : int
RegionCrossEvent
triggerAction : BoundaryCrossAction
TriggeredEvent
-activeFromTime : long
-activeToTime : long
SceduledEvent
-startDate : string
-startTime : string
RecurringEvent
-endDate : string
-endTime : string
-periodicity : long
1
1..*
Reminder
-description : string
0..1
1
1
1
Region
3..*
MapLocation
-lon : float
-lat : float
0..*
EventLog
-timestamp : string
1
1
Notification
-sentTimeStamp : string
-resolved : boolean
-description : string
BloodPresureLog
-triggerAction : BloodPressureTrigger
HeartBeatLog
-triggerAction : HeartBeatsTrigger
ImobilityLog
-minutesMotionless : int
RegionCrossLog
-triggerAction : RegionTrigger
-crossingLocation : MapLocation
<<enumeration>>
RegionTrigger
-enter : string
-leave : string
<<enumeration>>
BloodPressureTrigger
-minDiastolic : int
-maxDiastolic : int
-minSystolic : int
-maxSystolic : int
<<enumeration>>
HeartBeatsTrigger
-minBpm : int
-maxBpm : int

### CareGiver

The CareGiver class represents the users of the web application. Each CareGiver has his own account and is able to monitor many CareTakers. The attributes that each CareGiver has are those presented below:

- id: careGiver's unique identifier.
- firstName: careGiver's first name.
- lastName: careGiver's last name.
- email: careGiver's email address.
- password: careGiver's password used for logging in the system.

### CareTaker

The CareTaker class represents the users of the android application which is running on the smart watch. The attributes that each CareTaker has are:

- id: careTaker's unique identifier.
- firstName: careTaker's first name.
- lastName: careTaker's last name.
- deviceID: unique identifier of smart watch device.
- dateOfBirth: careTaker's date of birth.
- phoneNumbers: careTaker's or family phone numbers.
- emergencyNumber: phone number that the smart watch will call in case of emergency.
- address: careTaker's address.
- disease: careTaker's disease in case he has one.
- notes: notes for caretaker.

### Event

The Event class represents all the events that have been saved in our system. There are two categories of events, events that the caregiver sets on the smart watch to get fired at a specific time and events that are defined by the caregiver and are triggered from a caretaker's actions. The attributes composing an event are the following:

- id: the event's unique identifier.
- isEnabled: specifies if the event is enabled or not.
- sendMail: specifies if the caregiver will be sent an email in case the event is triggered.

**Scheduled Event**

The Scheduled Event class represents the events that the caregiver sets to be shown and ring on the smart watch at a specific date and time. A Scheduled Event may be described using the following attributes:

- startDate: date that the event will ring on the smart watch for the first time.
- startTime: time that the event  will ring on the smart watch for the first time.

**Recurring Event**

The Recurring Event class represents scheduled events that are repetitive and set to ring more than one time. The attributes composing a recurring event are the following:

- endDate: date that the event will stop showing.
- endTime: time that the event will stop showing.
- periodicity: specifies every how many hours the event will be displayed.

**Reminder**

The Reminder class represents the events that the caregiver defines to ring on the smart watch at a specific date and time. Reminders can be repetitive or non repetitive. A Reminder may be described using the following attributes:

- description: description of the reminder.

**Triggered Event**

The Triggered Event class represents the events that the caregiver defines and are triggered by a caretaker's actions. Moreover it can be of four basic types: Blood Pressure, Immobility, Heart Beat and Region Cross. A Triggered Event may be described using the following attributes:

- activeFromTime: time that event start to be enabled.
- activeToTime: time that event stops to be enabled.

### Blood Pressure Event

The Blood Pressure Event class is a type of Triggered Event that gets triggered when a caretaker's blood pressure, systolic or diastolic, is not normal. A Blood Pressure Event may be described using the following attributes:

- minSystolicBloodPressure: min systolic pressure limit.
- maxSystolicBloodPressure: : max systolic pressure limit.
- minDiastolicBloodPressure: : min diastolic pressure limit.
- maxDiastolicBloodPressure: : max diastolic pressure limit.

### Immobility Event

The Immobility Event class is a type of Triggered Event that gets triggered when a caretaker is standing still for a long period of time. An Immobility Event may be described using the following attributes:

- minutesMotionless: minutes that the caregiver will get notified if a caretaker is motionless.

### Heart Beat Event

The Heart Beat Event class is a type of Triggered Event that gets triggered when a caretaker's heart beats are not normal. A Heart Beat Event may be described using the following attributes:

- maxBeatsPerMinute: max heart beats per minute limit.
- minBeatsPerMinute: min heart beats per minute limit.

### Region Cross Event

The Region Cross Event class is a type of Triggered Event that gets triggered when a caretaker gets in or out of a defined area on the map. A Region Cross Event may be described using the following attributes:

- triggerAction: specifies if the region is an enter or leave area.

### Region

The Region class represents all the regions that the caregiver has defined.

**Map Location**

The Map Location class represents a geographical point on the map. More than three points define a geographical area. A Map Location may be described using the following attributes:

- lon: longitude value of the point.
- lat: latitude value of the point.

**Notification**

The Notification class represents the notifications that are send to the caregiver after the occurrence of an event. A notification can be described by the following attributes:

- sendTimeStamp: the time that the notification was sent.
- resolved: shows if the notification has been seen by the caregiver.
- description: the description of the notification

**Event Log**

The Event Log class represents the occurrence of a triggered event and is described using the attribute:

- timestamp: the time that the event was triggered.

**Region Cross Log**

The Region Cross Log class is a type of Event Log and presents the occurrence of a region event. The Region Cross Log may be described by the following attributes:

- enter: shows that a caretaker entered a danger area.
- leave: shows that a caretaker left from a safety area.

**Immobility Log**

The Immobility Log class is a type of Event Log and presents the occurrence of an Immobility event. The Immobility Log may be described by the following attributes:

- minutesMotionless: minutes that a caretaker was standing still.

## Heart Beat Log

The Heart Beat Log class is a type of Event Log and presents the occurrence of a Heart Beat event. The Heart Beat Log may be described by the following attributes:

- minBpm: min allowed beats per minute.
- maxBpm: max allowed beats per minute.

## Blood Pressure Log

The Blood Pressure Log class is a type of Event Log and presents the occurrence of a Blood Pressure event. The Blood Pressure Log may be described by the following attributes:

- minDiastolic: min allowed diastolic pressure.
- maxDiastolic: max allowed diastolic pressure.
- minSystolic: min allowed systolic pressure.
- maxSystolic: max allowed systolic pressure.

# Chapter 6

## Architecture

This chapter describes the system architecture, presents its basic components and provides a detailed analysis of the internal functionality. Furthermore, it advocates the architectural decisions that were made for the most important application components.

Built as a web application, the system adopts the Rich Internet Application (RIA) principles, which promote the development of web applications as desktop applications performing business logic operations on the server side, as well as on the client side. The client side logic operates within the web browser running on a user's local computer, while the server side logic operates on the web server hosting the application. Figure 6.1 displays our system's architecture.

For the development of the application we used many design patterns. The use of well-established and documented design patterns, speeds up the development process, since they provide reusable solutions to the most common software design problems. All the design patterns that were used in designing the system's architecture are presented in the following sections. The Model View Controller (MVC) design pattern and the Observer pattern were used on the client side, and a multi-tier architecture was implemented on the server side.

The analysis of the architecture is composed out of two parts; Section 6.1 presents the server side architecture and Section 6.2 presents the client side architecture.

## 6.1 Server Side

The Server Side part of our framework has a multi-layered architectural pattern and is consisted out of three basic layers: Service Layer, Business Logic Layer and Data Layer. This architecture increases the system's maintainability, reusability of the components, robustness, scalability, and security. As shown in Figure 6.1, the server side is comprised of a number of distinct modules which are described below.

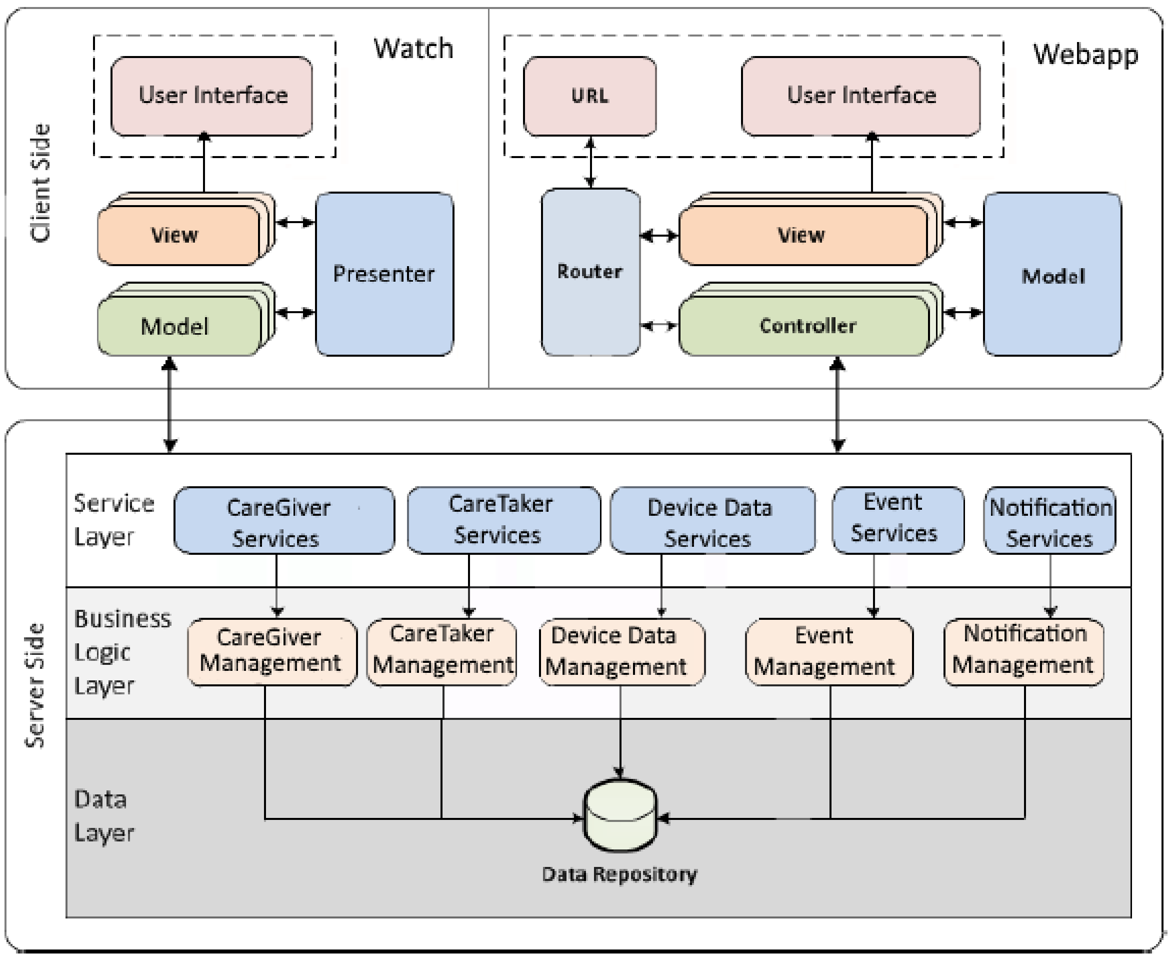


**Figure 6.1:** System reference architecture

### 6.1.1 Service Layer

The Service Layer is responsible for the communication between the client-side logic and the server-side logic, by exposing a set of services (operations) to the client-side components [16]. The basic services of our system are:

• CareGiver Services: facilitating the creation, retrieval, update and deletion of a caregiver.
• CareTaker Services: facilitating the creation, retrieval, update and deletion of a caretaker.
• Device Data Services: providing the means for the web application client side to use the data of the smart watch.
• Event Services: facilitating the creation, retrieval, update and deletion of an event.
• Notification Services: facilitating the saving and retrieval of Notifications.

### 6.1.2 Business Logic Layer

The Business Logic Layer, which is also called Domain Layer, contains the business logic of the application and separates it from the Data Layer and the Service Layer. In more detail the basic management modules are:

• The CareGiver Management Module: is responsible for the caregiver management.
• The CareTaker Management Module: is responsible for the caretaker management.
• The Event Management Module: is responsible for the event management.
• The Notification Management Module: is responsible for the notification management.
• The Device Date Management Module: is responsible for the persistence and accessing of smart watch data that that have been collected from the smart watch.

### 6.1.3 Data Layer

The Data Layer accommodates the external system which is used to index and persist data. This system is the Data Repository which is storing all the data of the system. Section 7.1 presents all the implementation aspects of the components in detail.

## 6.2 Client Side

The Client Side of the application is responsible for the interaction with the user. All the actions performed by an individual using the system, are handled by the client side logic, which is responsible for the presentation of the information as well as the communication with the server. In order to achieve a high level of decoupling between the components forming the client logic we adopted the Model View Controller (MVC) design pattern in our web application, as well as Model View Presenter (MVP) design pattern in our android application.

The usage of the MVC pattern introduces the separation of the responsibilities for the visual display and the event handling behavior into different entities, named respectively, View and Controller. Some of the advantages on this approach are: (a) maximization of the code that can be tested with automation (Web pages containing HTML elements are hard to test), (b) code sharing between pages that require the same behavior, and (c) separation of the business logic from the user interface logic from User Interface logic to make the code easier to understand and maintain.

On the other hand MVP design pattern is a set of guidelines that if followed, decouples the code for reusability and testability. It divides the application components based on its role, called separation of concern.

### 6.2.1 Model (Web application)

The Model refers to the business objects which are being used by our system. When the system needs to present information about a business object, the client side requests the respective information from the server side using the services that the later exposes. Similarly, when an update on the Model needs to be persisted, the client side sends the updated Model to the server side, triggering the indexing and storage of the business objects by the appropriate modules and external systems.

### 6.2.2 View

The views are responsible for the presentation of information to the user. Each one of the views controls a number of widgets on the application's graphical user interface. It consists of several handlers that are responsible for listening user actions, as well as HTML templates that define the presentation of the widgets.

### 6.2.3 Controller (Web application)

The Controllers are the modules that respond to the user input and interact with the Views in order to perform any change on the user interface. Furthermore, they maintain the Model and change it appropriately. Every View has a dedicated Controller which is responsible for managing, handling and propagating any changes that are to be performed or have already been performed to the user interface. Moreover there are several cases where a "composite" Controller manages a number of other Controllers in order to create complex widgets.

### 6.2.4 Router

The Router is used for deep-linking URLs to controllers and views. It manages the URL of the client browsers, providing a different path to each distinct interface, without raising a browser event that will force a reload on the whole page. When the URL changes, the Router analyzes the new path and handles the transition to the new View. This is performed using mappings between the different URLs supported in the system, the Controllers and the Views.

### 6.2.5 Model (Android application)

The model is responsible for handling the data part of the application.

### 6.2.6 View (Android application)

The view is responsible for laying out the views with specific data on the screen.

### 6.2.7 Presenter (Android application)

The presenter is a bridge that connects a Model and a View. It also acts as an instructor to the View.

# Chapter 7

## Implementation

This chapter provides the implementation details of some of the most important components in more detail. We have split the chapter into two Sections. Section 7.1 describes the Server Side and Section 7.2 describes the Client Side.

## 7.1 Server Side

The server side of our system is based on the Java programming language. Moreover, it is worth to mention that we have used Websocket communication protocol in order to send notifications directly to the user.

### 7.1.1 RESTful Web Services

In order to expose our data to our client applications and other external systems we developed an API that enables data access through the use of RESTful services. The advantages of REST are: (1) less overhead compared to SOAP, (2) less duplication since HTTP already represents DELETE, PUT, GET and POST operations, (3) more standardized, providing HTTP operations that operate consistently, (4) more human readable and testable, and (5) there is no requirement to use complex data interchange formats like XML. We built a package of RESTful services exposing all the functionality of our system and we created CRUD (create, retrieve, update, delete) operations for every possible feature. These services are used by both our client side applications, web application and the smart watch application, to fetch, create, update and delete data on our system.

### 7.1.2 Security

Security is a very important aspect in every system, especially the web-based applications.

For securing our application, we require the users to be logged in, in order to use the services, check for the existence of the user data in the database, and deny access otherwise.

### 7.1.3 Websocket

WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection. WebSocket is a different TCP (transmission control protocol) protocol from HTTP.

The WebSocket protocol enables the communication between a browser and a web server with lower overheads, facilitating real-time data transfer from and to the server. Websocket protocol is providing a way for the server to send content to the browser without being requested by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way, a two-way ongoing conversation can take place between a browser and the server.

## 7.2 Client Side

The client side refers to both the web application and the android application. These two applications are based on the latest web-application standards, rely on the JavaScript and java programming languages and the web application also makes extensive use of the AngularJS framework.

Moreover, they have been created in order to match different user requirements, and thus their user interfaces are implemented differently in order to match the goals of their users. It is worth to note that apart from JavaScript, the user interface layout has been built using HTML5 and CSS3.

Regarding our android application it is worth to mention that when the smart watch is offline, the application is storing the data locally and sends all the data when the device is again back online. The data is stored using the Shared Preferences interface implementations with which we can store persistent sets of data in the file system. The data is available across application restarts or even device stop/start. Moreover the android application is able to run in the background which means if the user is using other applications, our application will still send and receive all the needed data from the database.

For the code organization of the client side we have used various open source libraries and frameworks. Section 7.2.1 presents the use of MVC pattern which is used in our system. Section 7.2.2 provides some details about the way that the user interfaces have been implemented for our applications. Section 7.2.3 describes the parts that map the html templates with controllers and urls, while Sections 7.2.4 and 7.2.5 describe the AngularJS controller and service implementation on the client side. Finally, Section 7.2.6 describes the use of maps in our application.

### 7.2.1 MVC Pattern

The client side of our applications has been based on the Model View Controller design pattern. Numerous JavaScript frameworks implementing the MVC pattern have emerged during the last years. Most of them force strict definition format rules to the various components due to the special handling that complex JavaScript libraries need. After having evaluated the most popular open-source frameworks, we decided to use AngularJS.

AngularJS [1] has been developed as an MVC framework for JavaScript applications. It's main purpose it to provide the basic structure to the application. It is very lightweight and very extensible, allowing developers to customize it with minimum effort according to their needs. Apart from the MVC features, it provides other useful functionality, including: models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.

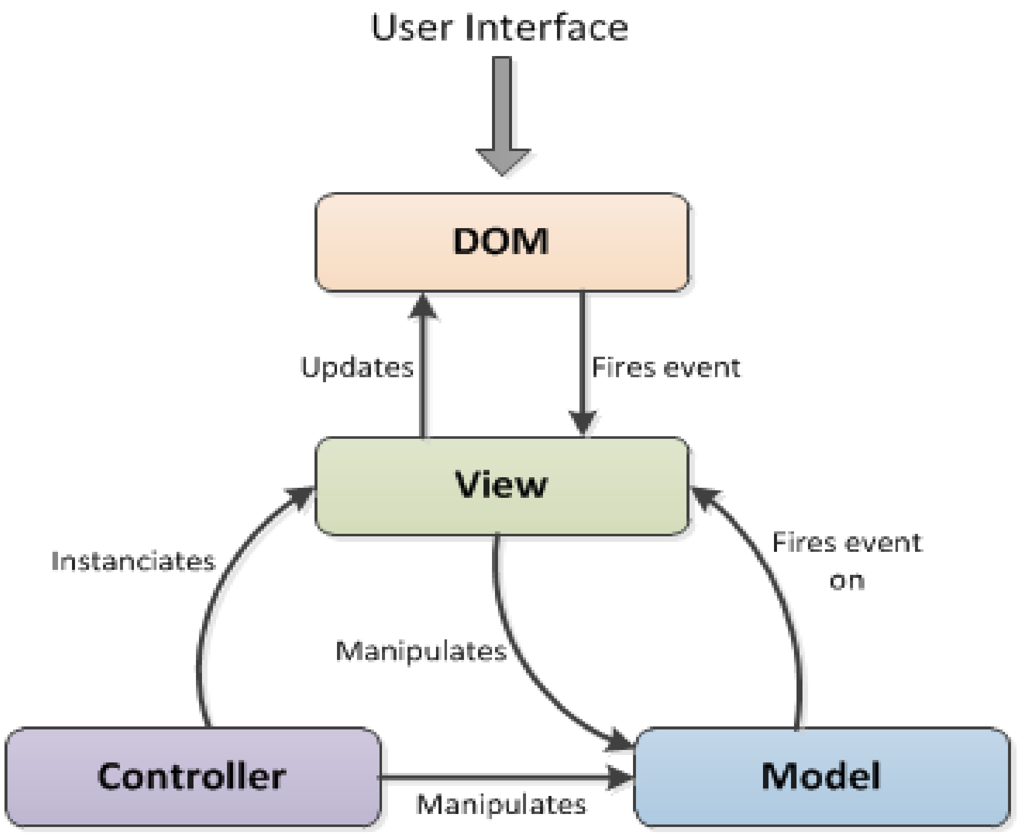


**Figure 7.1:** Model View Controller Pattern in the application

Figure 7.1 presents the MVC components and the interaction between them in our applications. The Controller instantiates the View and manipulates the Model. After that, the View listens for updates of the Model, and when the updates happen it is re-rendered with the changes. This functionality is called two way binding and is presented in Figure 7.2. The View is also responsible for the updating of the Document Object Model (DOM) [15] when needed, as well as the handling of interaction events which are triggered by the user. In some cases the View also manipulates the Model.

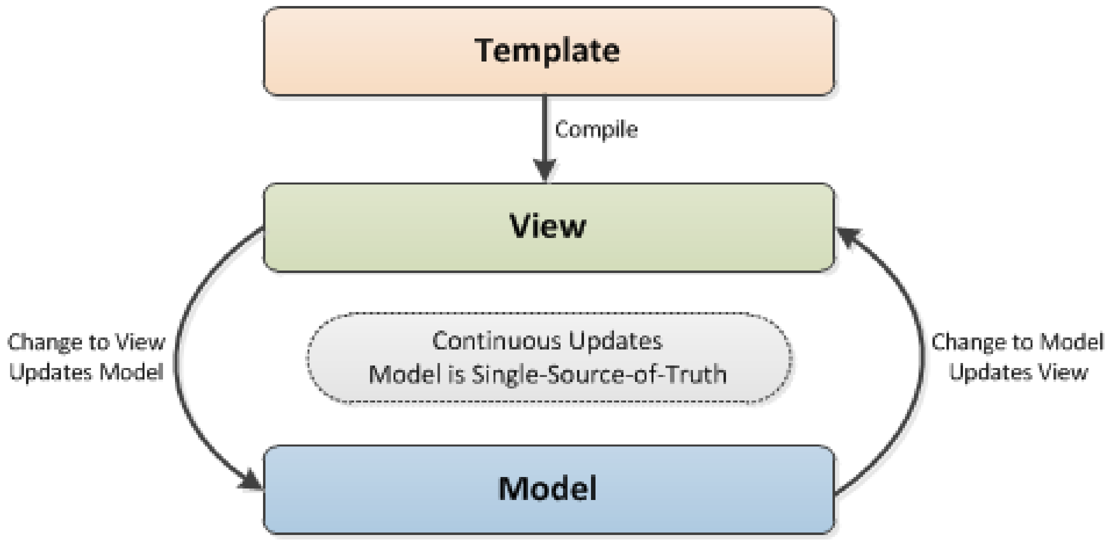

Template
Compile
View
Change to View Updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model Updates View
Model

**Figure 7.2:** Two way binding.

### 7.2.2 User Interface

This section presents some user interface implementation details for our applications.

#### Web application desktop

When building an application optimized for desktop browsers, the developer must take into account the size of the screen which affects the usability of the applications. Figure 7.3 presents the breakdown of main elements (top bar, left menu and main content) of our desktop user interfaces.

Additionally, we tried to build our desktop application as much responsive as possible. To this end, we made use of the @media rule which is used to define style rules for different media types and devices.

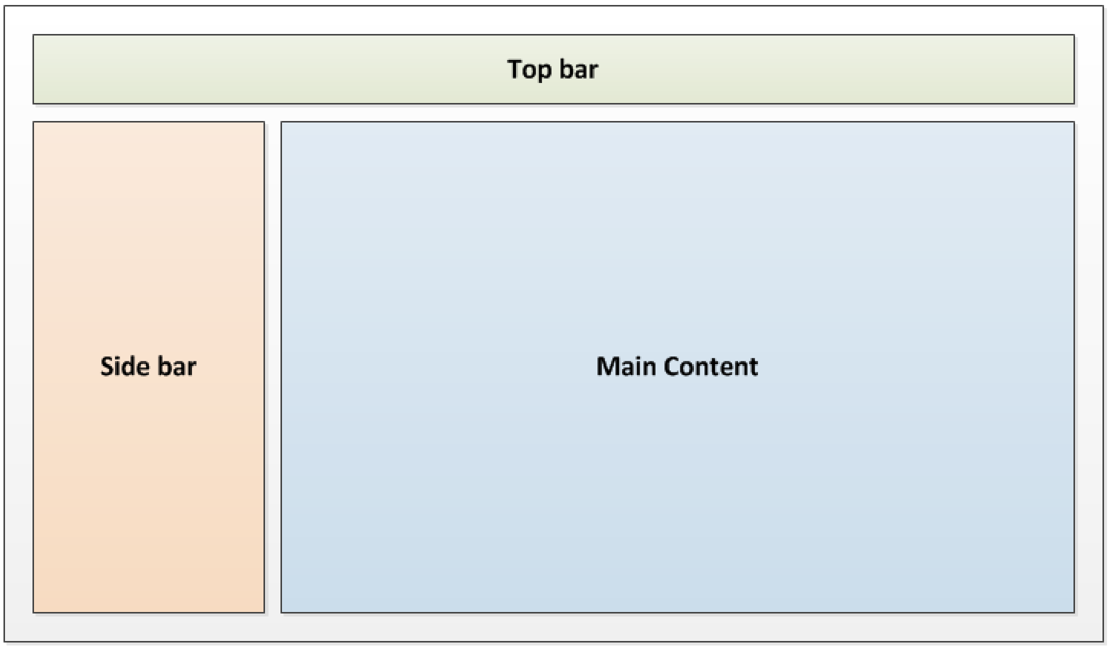


**Figure 7.3:** The main areas of desktop user interfaces.

**Web Application on Mobile Phones**

When building an application optimized for mobile devices, the developer must take into account the small size of the screen which affects the usability of the applications. The menus that are provided with the mobile applications are at most times hidden and become visible only when the user needs them. This happens with the use of a button that reveals the menu.

However, when building web applications that can be used from mobile devices, the developer needs to implement the functionality for the sliding menu by himself. There are a number of free implementations available, but they all have their disadvantages when it comes to performance, since they try to apply themselves to the general cases, providing more functionality than needed most of the times. To this end we formed an open-source side bar according to our needs.

Figures 7.4 and 7.5 present the main elements of our mobile user interfaces. These are the two menus (right and top) and the main content area in bellow our top menu. When the left menu is hidden, it is pushed away from the visible area (Viewport) of the document. When the left menu is becoming visible, the whole canvas is moved to the right, which causes the left menu to appear.

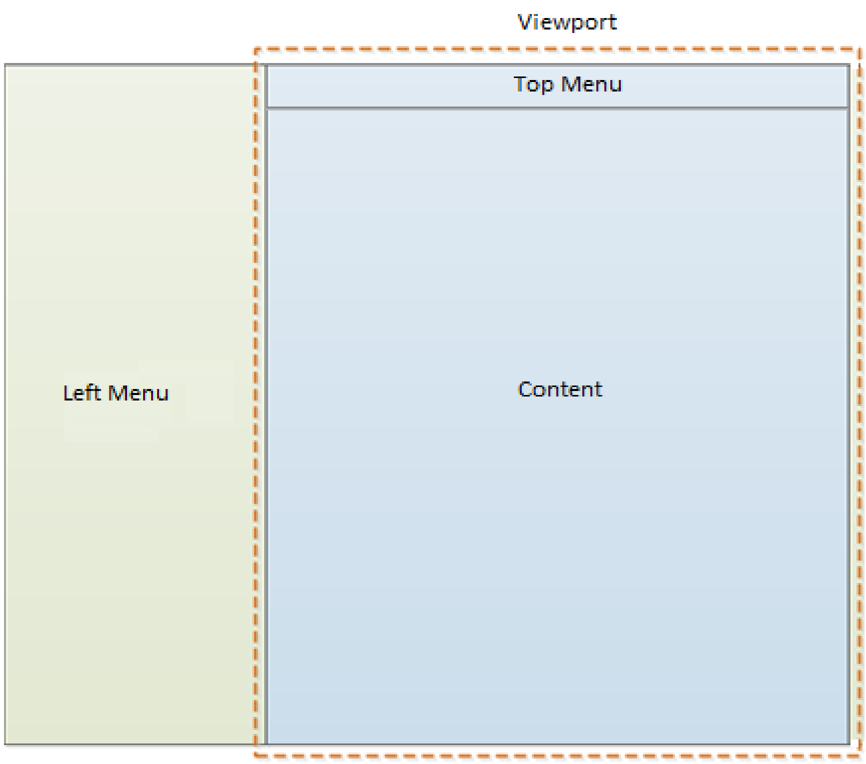


**Figure 7.4:** Viewport – Hidden Menu.

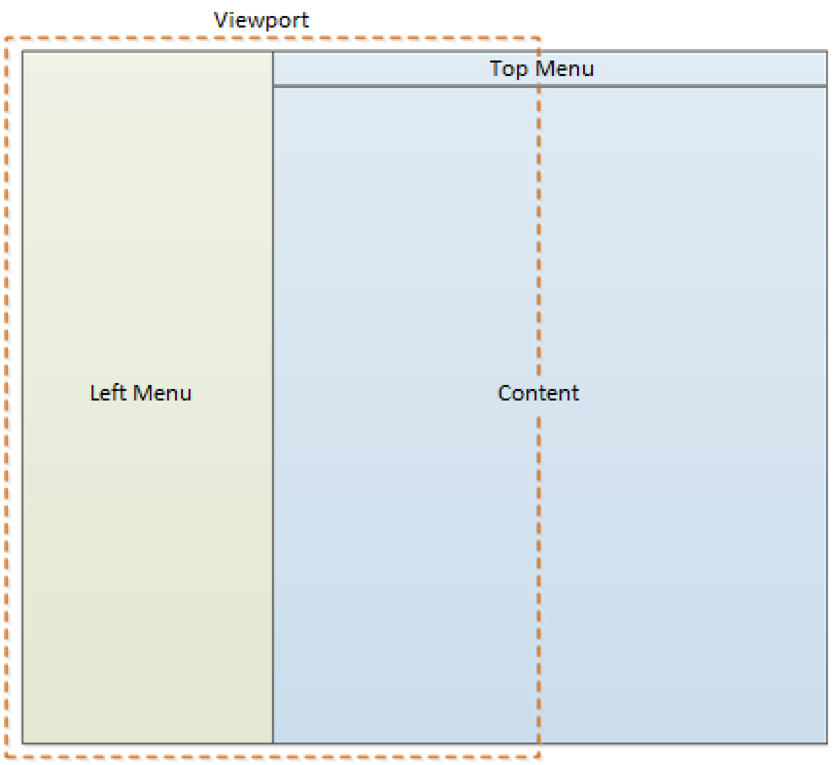


**Figure 7.5:** Viewport – Visible Menu.

## Android Application on Smart Watch

Similarly when building an application optimized for mobile watch devices like smart watches, the developer must take into consideration the small screen size. All the buttons of the application must be big enough for the user to interact with them and the app also needs to function with minimum user interaction.

Moreover, the user interface should have a simple and minimalist layout so it will not make the user confused. With a simple glace, the user should be able to view and understand any information that is deemed as being critical in the context of the app. Taking everything under consideration, our android application stayed as simple as possible, showing the caretaker all the needed information in one layout.

Figure 7.6 present the main elements of our smart watch user interfaces. These are three main areas: (1) Information needed to be displayed to the caretaker (Connection, Battery percentage). (2) A button that displays all the scheduled reminders that have been set to the watch. (3) A hold emergency button that will make the device call for help when it is pressed.

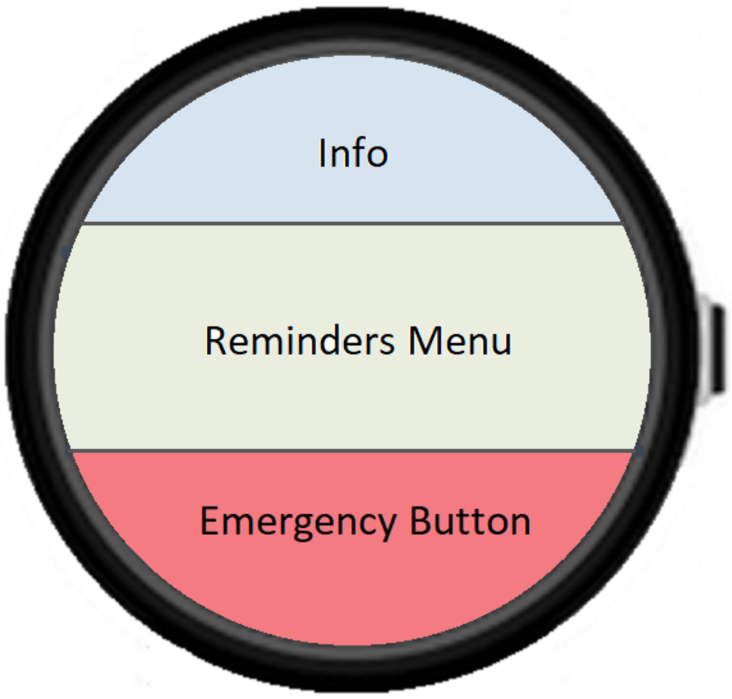


**Figure 7.6:** Android application main areas

### 7.2.3 Router

Modern web applications, built with the latest technologies, enable more powerful interaction with the users. Their features are very close to desktop applications in terms of complexity and management. These applications are called RIAs (Rich Internet Applications).

A very important aspect in the development of RIAs, is the use of AJAX for the communication between the client and server. This enables the generation and presentation of content without forcing the web browser to reload a new web page. JavaScript source code running on the user's browser is responsible for issuing AJAX calls to the server as well as manipulating the Document Object Model and presenting the data.

As JavaScript applications are being used to manipulate all of the interfaces presented to the user, they are becoming more complex. Two major problems have arise from this complexity. Firstly, the change of user interface using JavaScript needs special treatment in order to allow the browser's history support. This appears because the web browsers tend to push the URLs to history
and reload a new one whenever a change is performed. In RIAs this is not always the case. The second problem is the ability of users to create bookmarks for a specific user interface screen, which is not possible if the application does not propagate the interface change to the browser's URL.

In our system, these issues are handled by the Router component, which takes care of both the history of the browser and the mapping between the different screens of our application and their URLs. Our Router is based on the Router object provided by the AngularJS library.

### 7.2.4 Controller

In AngularJS a Controller is a JavaScript constructor function that is used to augment the Angular Scope. When the "ng-controller" directive is used in the DOM, Angular creates a new Controller object, using the specified Controller's constructor function. In our application we use controllers to respond to the user input and interact with the Views in order to perform any change on the user interface. Furthermore, they are used to maintain the Model and change it appropriately.

### 7.2.5 Services

The services in our application are responsible for handling all the data communication between the client side and the server side. Apart from those in the server side, there are also services in the client side. These services issue the AJAX calls to the server and get the responses.

### 7.2.6 Maps

For the maps used in our user interfaces, we used an open-source JavaScript library from Google Maps for developers.

# Chapter 8

# Graphical User Interface

This chapter presents the methodology followed for designing the user interfaces of the applications, as well as the final product. Section 8.1 describes the phase of designing and user interface prototyping, while Section 8.2 presents the final outcome of the graphical user interfaces as they have been implemented.

## 8.1 Prototypes

At the initial stages of the design process of the application, we sketched some prototypes of the user interfaces to visualize and organize our ideas. Sketches enable better visualization of the user's interface, interaction and the navigation between the application screens before the actual creation of the application.

The following sections present some early user interface prototypes of the application. More specifically, Section 8.1.1 presents the user interface prototypes of the application on desktop, while Section 8.1.2 provides the user interface prototypes of the application on mobile phone and Section 8.1.3 presents the user interface prototypes of the android application.

### 8.1.1 Desktop Application

The figures of this section present some user interface prototypes of the application running on a large screen like a desktop computer which have been sketched in the early stages of the design process. Each prototype has a title beneath it, indicating the purpose that the user interface serves.

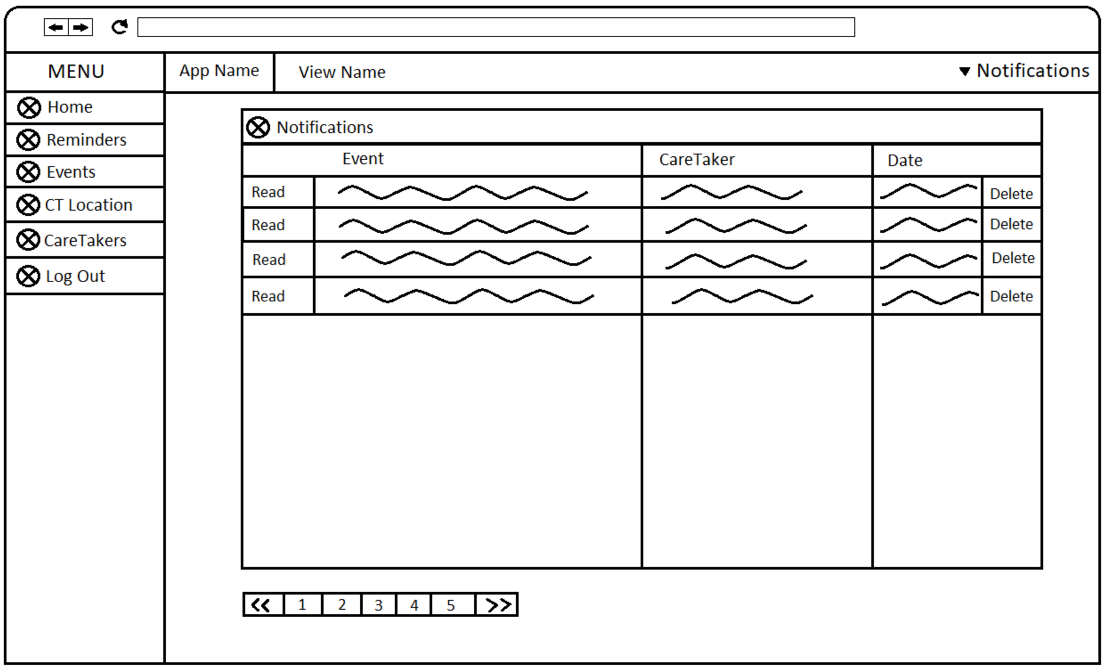


**Figure 8.1:** Show all careTakers' Notifications.

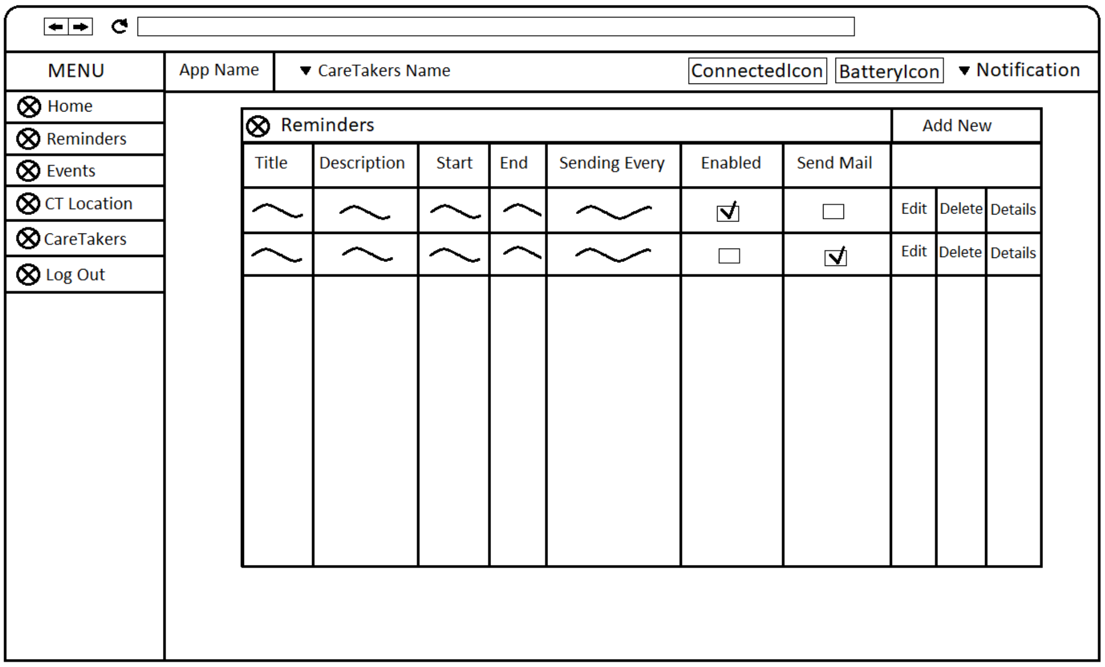


**Figure 8.2:** Show the selected careTaker's Reminders.

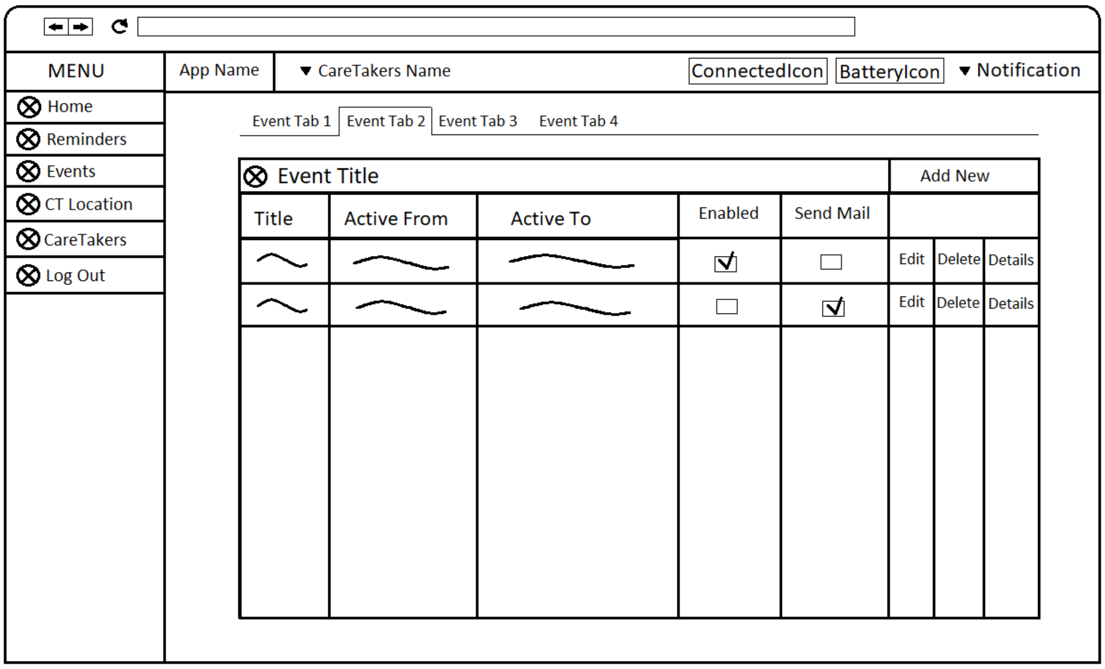


**Figure 8.3:** Show the selected careTaker's Events.

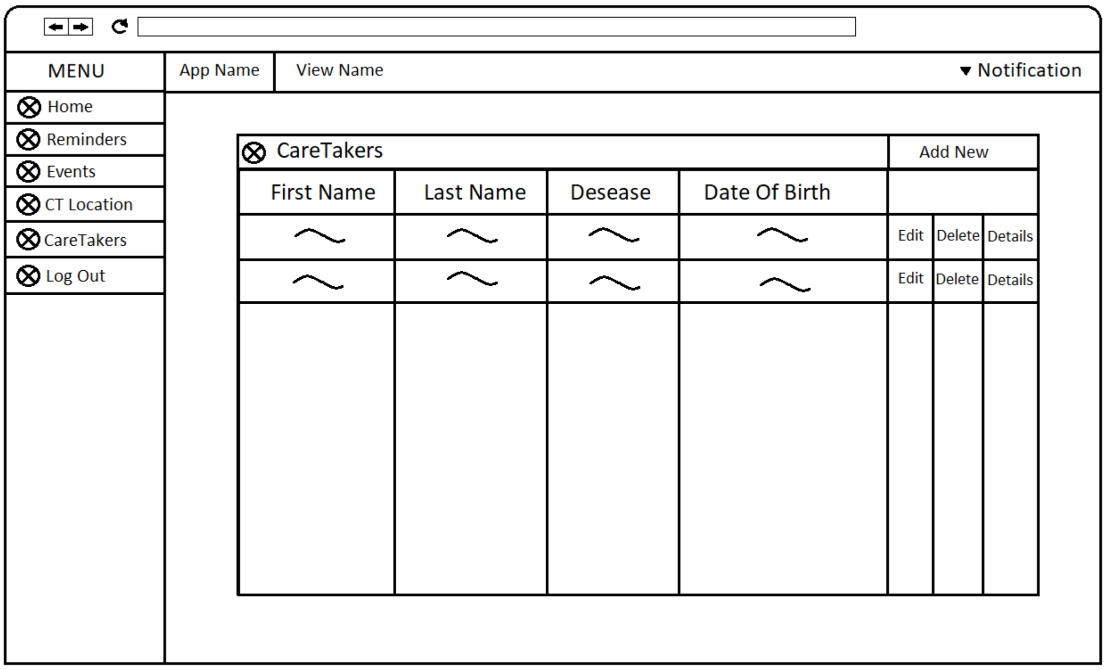


**Figure 8.4:** Show all careTakers.

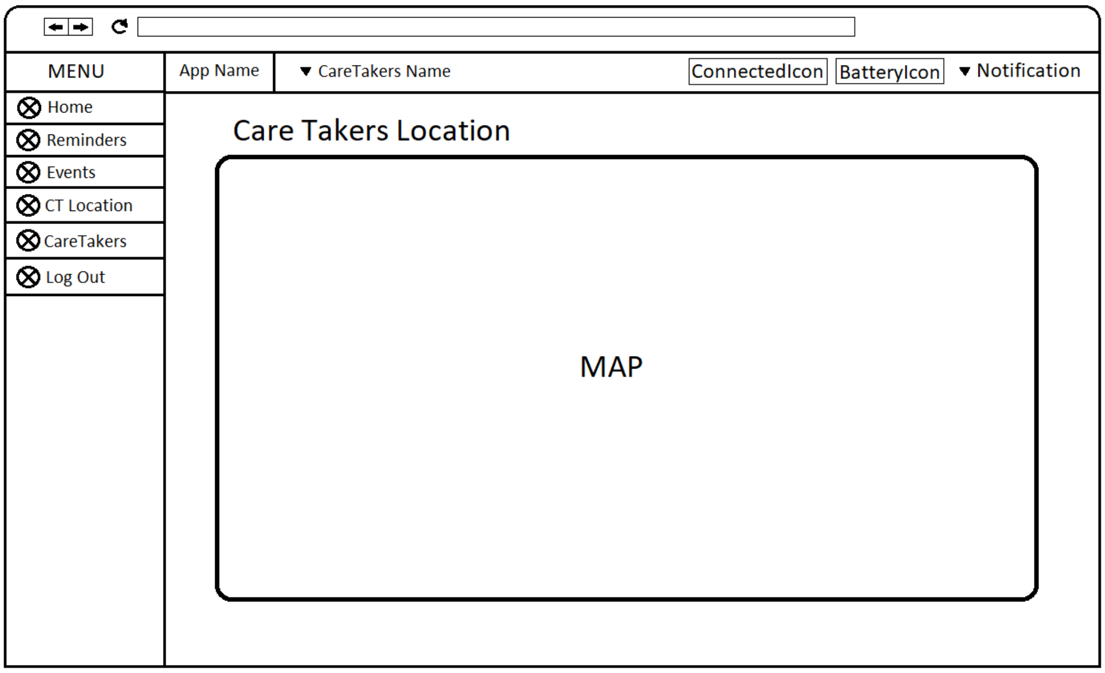


**Figure 8.5:** Show the selected Caretaker's Location.

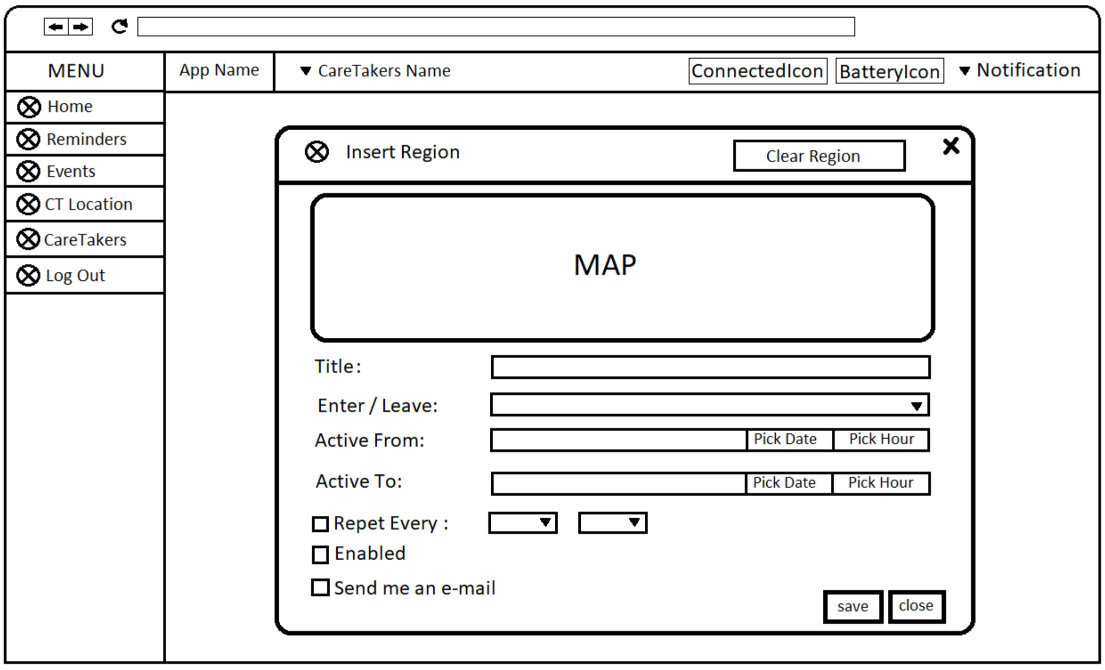


**Figure 8.6:** Insert/Edit Region Event.

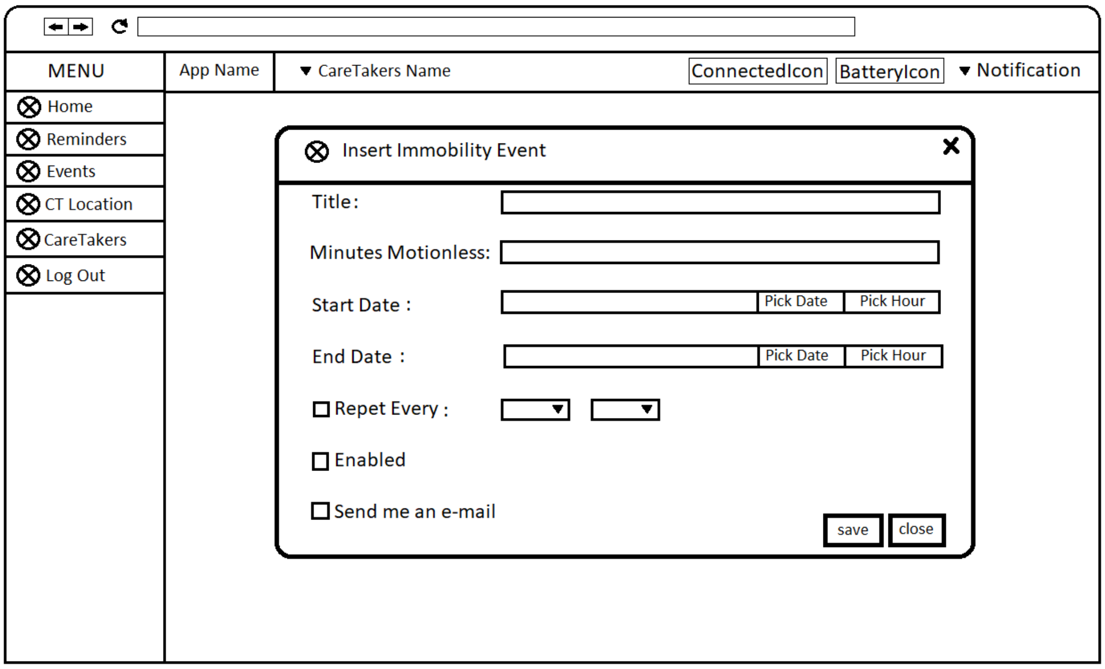


**Figure 8.7:** Insert/Edit Immobility Rule.

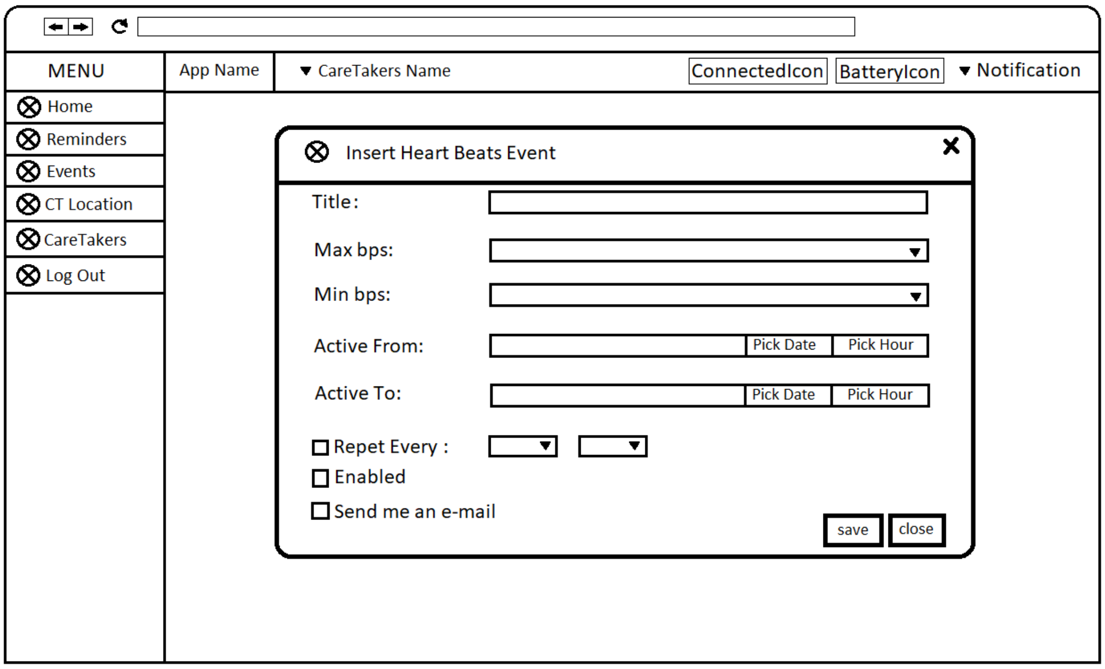


**Figure 8.8:** Insert/Edit Heart Beats Event.

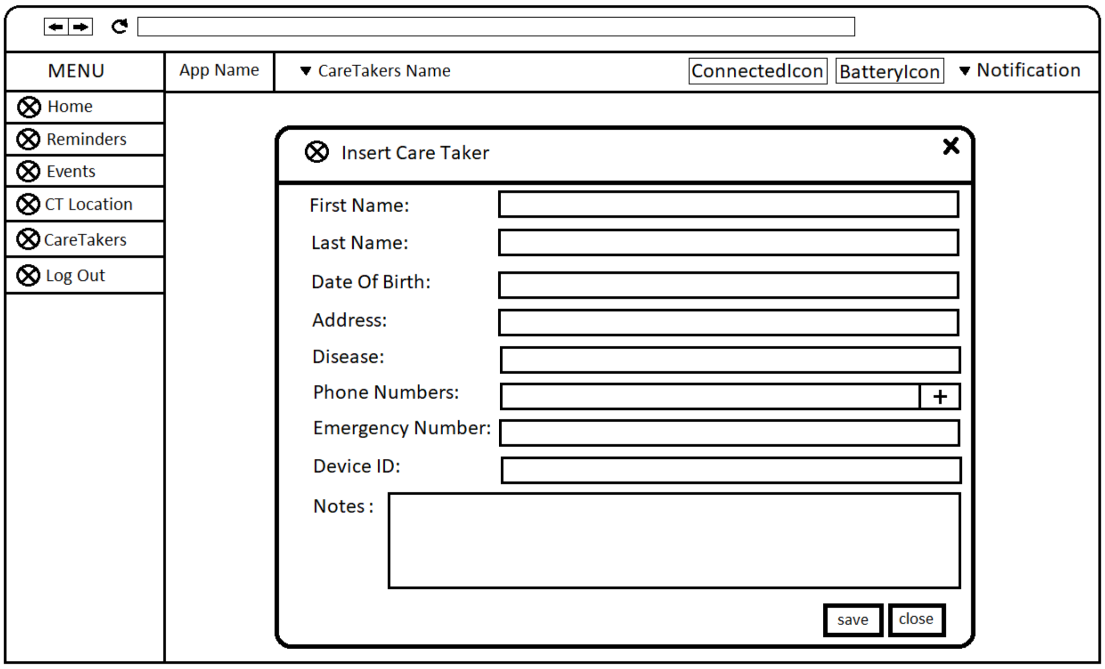


**Figure 8.9:** Insert/Edit CareTaker.

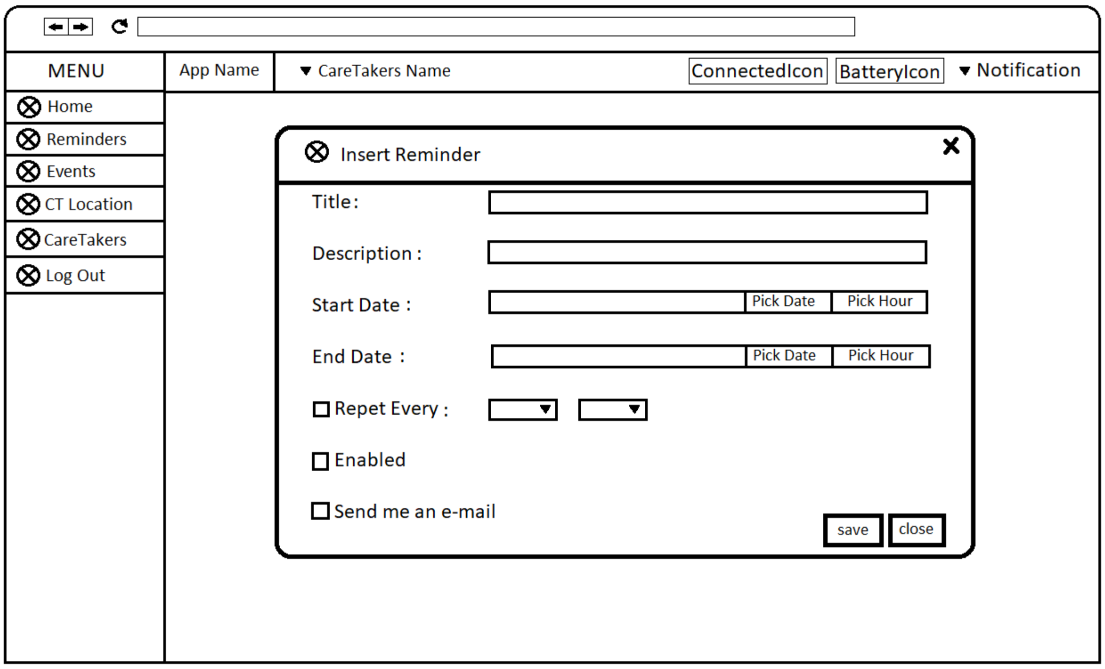


**Figure 8.10:** Insert/Edit Reminder.

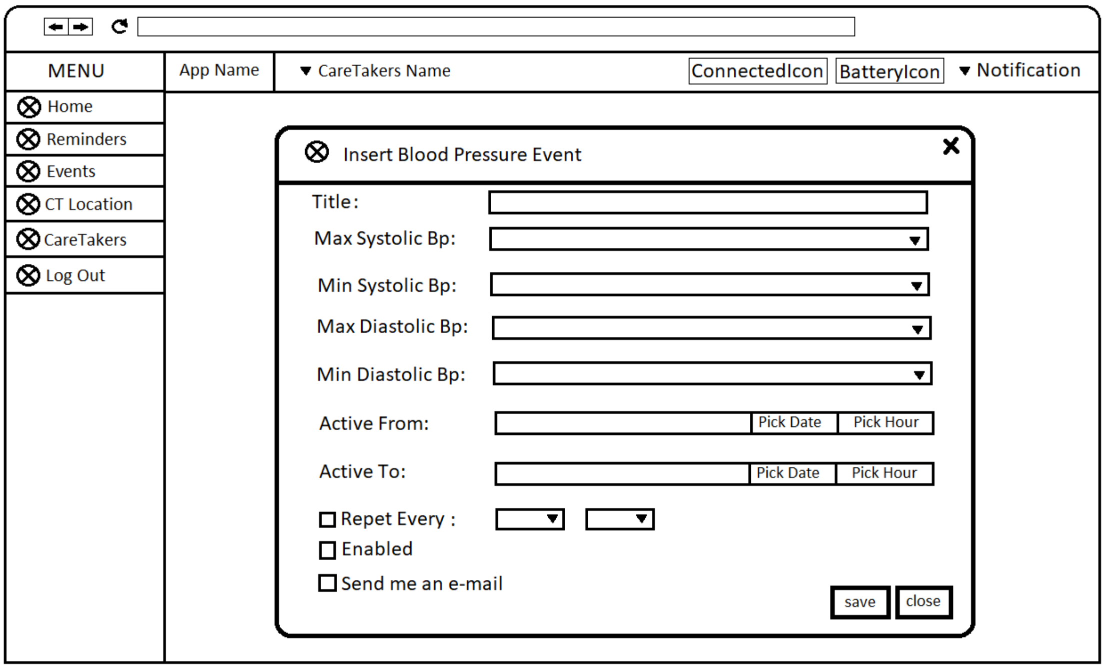


**Figure 8.11:** Insert/Edit Blood Pressure Event.

## 8.1.2 Desktop Application on mobile

The figures of this section present some user interface prototypes of our application running on a mobile device which screen size is smaller than computers.

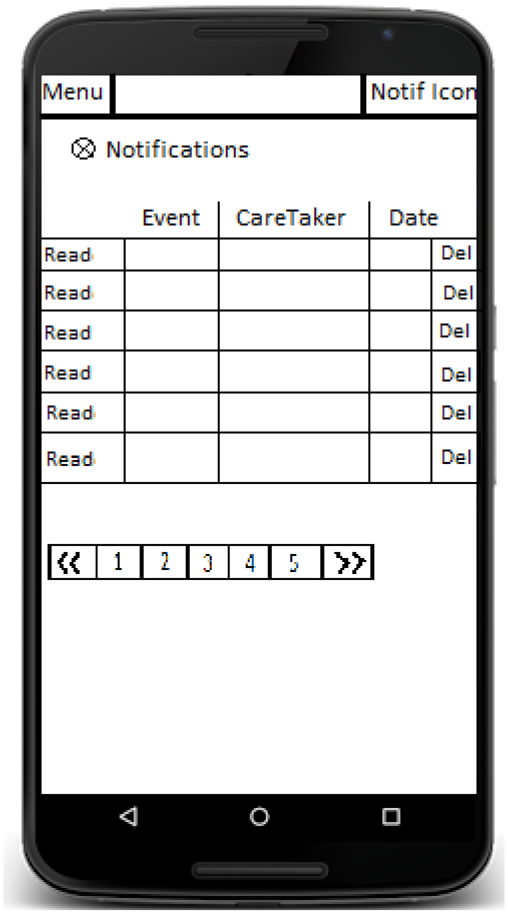


**Figure 8.12:** List of notifications.

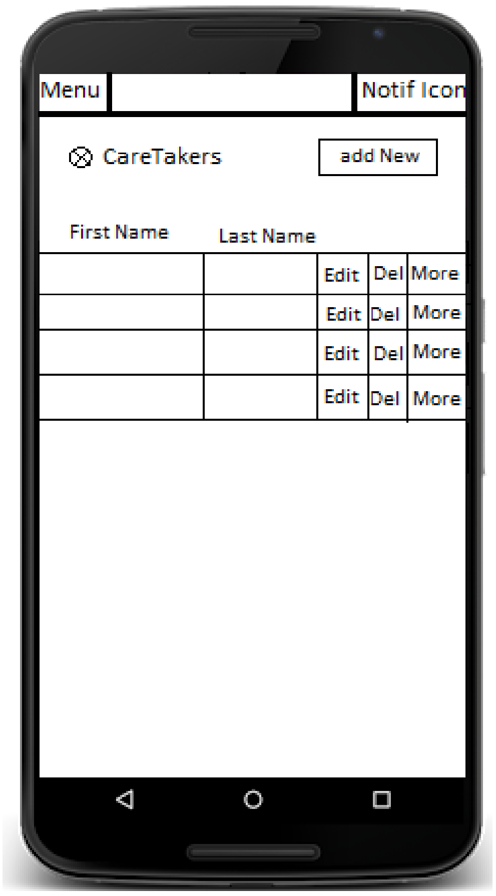


**Figure 8.13:** List of caretakers.

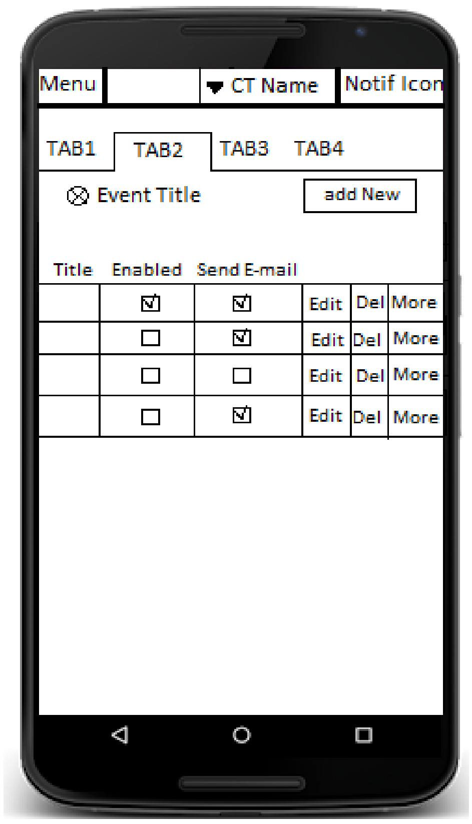


**Figure 8.14:** List of Events.

**Figure 8.15:** The Caretaker's map location.

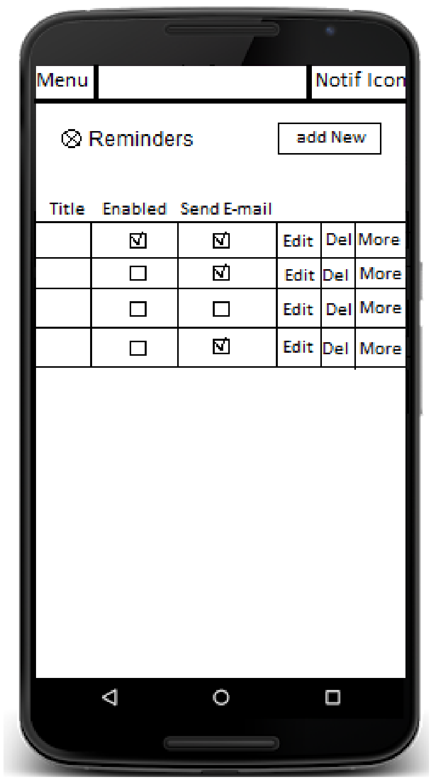


**Figure 8.16:** List of reminders.

## 8.1.3 Android Application

The figures of this section present some user interface prototypes of our android application which is used by the caretaker.

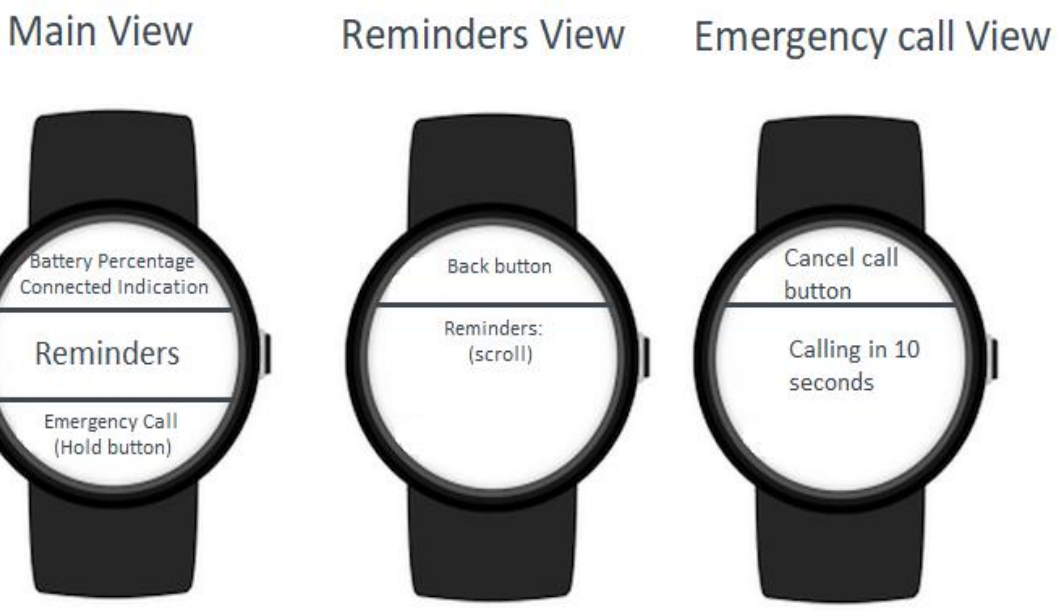


**Figure 8.17:** Android application prototypes.

## 8.2 Usability Evaluation

The following sections present the evaluation methods that have been used to improve usability of our applications. More specifically, Section 8.2.1 describes the Heuristic Evaluation method, while Section 8.2.2 describes the Think- Aloud method.

### 8.2.1 Heuristic Evaluation

A heuristic evaluation [17] is a usability inspection method for computer software that helps to identify usability problems in the user interface (UI) design. It involves evaluators examining and interacting with the interface, judging its compliance with the usability principles. The main goal of heuristic evaluations is to identify any problems associated with the design of user interfaces. The heuristics that were used in this thesis were from Nielsen's book [18] and are the following:

- Visibility of system status.
- Match between system and the real world.
- User control and freedom.
- Consistency and standards.
- Error prevention.
- Recognition rather than recall.
- Flexibility and efficiency of use.
- Aesthetic and minimalist design.
- Help users recognize, diagnose, and recover from errors.
- Help and documentation.

Heuristic evaluation requires only one expert, so the evaluation was made by us. It was held at the early stages of design and we examined the system using the rules of Nielsen. Firstly we tried to find out problems that naive users might have. Then we examined the system having in mind the users' primary goals and their usability targets.

### 8.2.2 Think Aloud

After the corrections of the heuristic evaluation we put users to perform the same tasks on the real system. The aim was to see how people react with the real application. So we evaluated the application with the method of think-aloud. Think-aloud protocols [19] are used to gather data in usability testing of user interfaces. Think-aloud protocols involve participants thinking aloud as they are performing a set of specified tasks. Users are asked to say whatever they are looking at, thinking, doing or feeling during the whole procedure. This enables observers to see first-hand the process of task completion. Observers of this test are asked to objectively take notes of everything that the users say, without answering most questions during the test since we want to see if the system can be used without help. The purpose of this method is to determine user's expectations and identifying what aspects of the system are confusing.

To run a basic thinking aloud usability study, you need to do three things:

1. Recruit representative users.
2. Give them representative tasks to perform.
3. Let the users do the talking and tell what they think and do.

**Procedure**

We chose seven adults / middle-aged persons of different personal computer skills. Some of them were naive users while others had an experience with personal computers and touch devices. According to the method we allowed the users to follow all the steps needed to reach the completion of some tasks. We were next to them to see whether they make mistakes or when they are confused. Also an important requirement was that the user should tell us what he was thinking. Furthermore, we wanted to measure the efficiency of use so we defined the estimated time that every task needs to be completed and then we also calculated the average time that the users needed to complete the tasks. The tasks that were performed by the users to evaluate the web application were the following:

| **Tasks:** | **Estimated Time:** | **Average Time:** |
|---|---|---|
| 1. Create an account. | **(50 sec)** | **(45 sec)** |
| 2. Add a Caretaker. | **(1 min 40 sec)** | **(2 min 27 sec)** |
| 3. Add a Reminder. | **(1 min)** | **(1min 9 sec)** |
| 4. Add a Region Event. | **(1 min 20 sec)** | **(1 min 34 sec)** |
| 5. Add a Heart Beat Event. | **(40 sec)** | **(37 sec)** |
| 6. Add a Blood Pressure Event. | **(40 sec)** | **(35 sec)** |
| 7. Add an Immobility Event. | **(30 sec)** | **(21 sec)** |

| | | |
|---|---|---|
| 8. Edit a Reminder. | **(15 sec)** | **(13 sec)** |
| 9. Edit an Event. | **(15 sec)** | **(13 sec)** |
| 10. Delete a Reminder. | **(8 sec)** | **(9 sec)** |
| 11. Delete an Event. | **(8 sec)** | **(8 sec)** |
| 12. View all Notifications. | **(5 sec)** | **(4 sec)** |
| 13. Log out. | **(5 sec)** | **(7 sec)** |

For the evaluation we also needed some elderly people and young children to use and interact with the android application at the smart watch. Three individuals took part for this evaluation. Because the android application is quite simple to use, there was no need for a greater number of users. The tasks that were performed by the users to evaluate the android application were the following:

| **Tasks:** | **Estimated Time:** | **Average Time:** |
|---|---|---|
| 1. Open the application. | **(10 sec)** | **(8 sec)** |
| 2. View scheduled reminders. | **(3 sec)** | **(3 sec)** |
| 3. Make an emergency call. | **(3 sec)** | **(4 sec)** |

As we can see the estimated times were very close to the average time that it was needed for the completion of the tasks. This was very encouraging because it means that the system was quite easy to use. The most difficult task that it took quite long to be completed was adding a new Caretaker. This was because the message explaining how to connect the smart watch was not very clear which is something that it was corrected afterwards.

The users were also asked to complete a questionnaire and we calculated the average score of each question. The scale of results ranged from 1 to 10, as we approach 1, the more negative the answer, the closer we get to 10, the more positive. The results of the questionnaires are presented below:

| | |
|---|---|
| • I am overall satisfied with the system's usability. | **9.3** |
| • I was capable of completing effectively the tasks needed using the system. | **9.6** |
| • I was capable of completing the tasks needed quickly. | **9.4** |
| • I feel comfortable using the system. | **9.9** |
| • It was quite easy to learn how to use the system. | **9.6** |

| | |
|---|---|
| I think that I became productive quite quickly using the system. | **9.9** |
| The system provides messages explaining its functionality. | **8.1** |
| Whenever I made a mistake using the system I returned easily to the previous state. | **10** |
| I was easy to find the information that I needed. | **9.4** |
| The information provided by the system was easy to understand. | **9.3** |
| The information provided by the system was sufficient to help me complete the tasks. | **7.9** |
| The organization of information provided by the system was clear. | **9.6** |
| The user interface was pleasant. | **9.7** |
| I like using the user interface of the system. | **9.7** |
| This system includes all the capabilities and functionalities that it should. | **10** |
| I am overall satisfied with the system. | **9.4** |

**Conclusions:**

From the results above, we can assume that the users found the system satisfactory in most of its sectors. As we see none of the users, despite been asked to answer objectively and without leniency, did not rate a question with a grade of less than seven. This may be because users generally do not judge a system easily when asked to, because they do not want to reduce the work of the creators. Nevertheless, the questions that were given the lowest score were the ones including the messages and the information provided by the system. The lower scores on those questions rely on the fact that the system's messages were not sufficient and the information provided to the user should be clearer. As a result we added more tips to the system helping the user understand easily each of the system's functionality, as well as more error messages so that the user will not be confused. By analyzing the results, we also saw that the users of the android application think that the system is very easy to use. This is very important considering that the smart watch application will be used mainly by people with special needs.

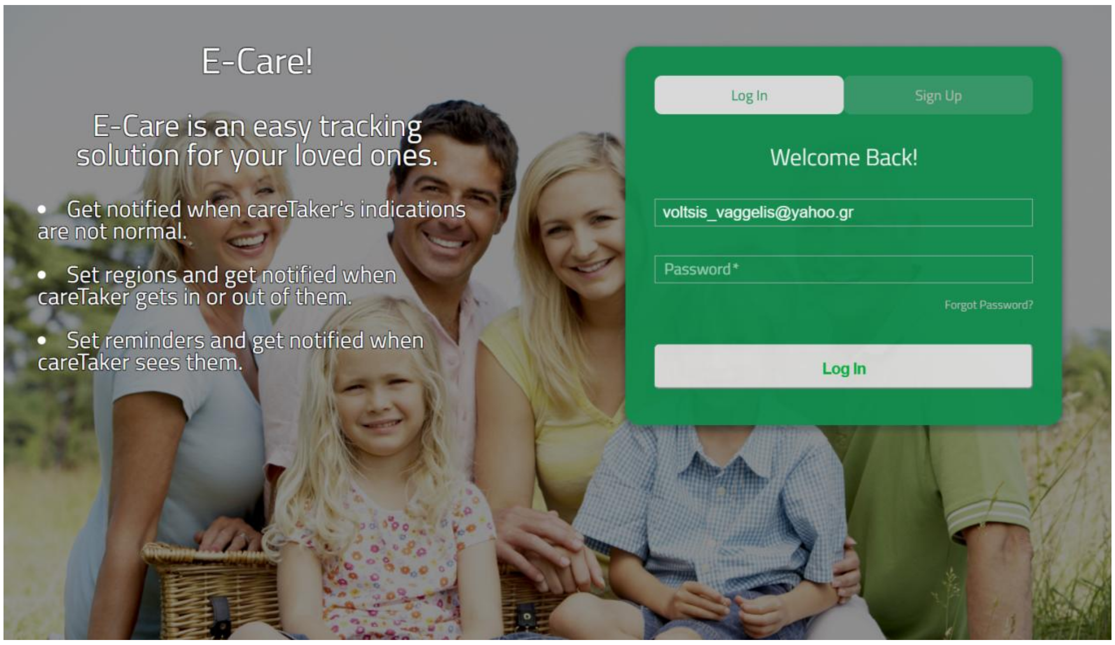
## 8.3 User Interfaces

The following sections present some graphical user interfaces of the system as they have been implemented. In more detail, Section 8.2.1 presents user interfaces of the web application on desktop, Section 8.2.2 provides the user interfaces of our web application on mobile devices and Section 8.2.3 presents user interfaces of the android application on the smart watch.

### 8.2.1 Web application on desktop

#### Log in

When the caregiver opens the application and he is not logged in the log in screen is displayed (Figure 8.18), prompting him to enter his credentials. The user also has the option to create a new account by clicking the sign up button.


E-Care!

E-Care is an easy tracking solution for your loved ones.

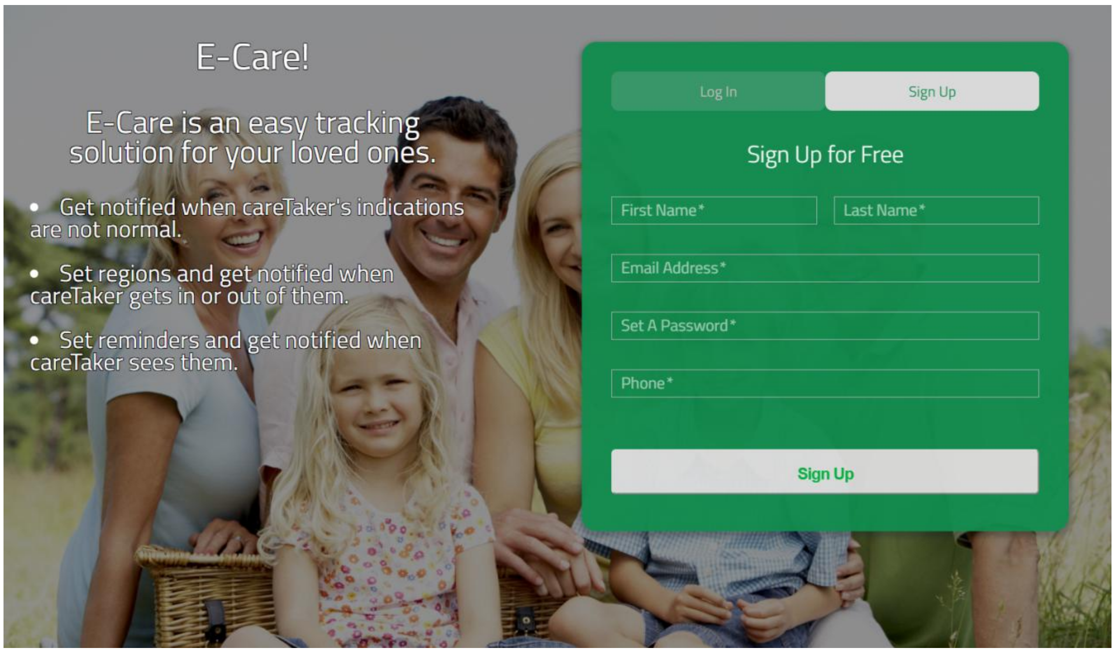
- Get notified when careTaker's indications are not normal.
- Set regions and get notified when careTaker gets in or out of them.
- Set reminders and get notified when careTaker sees them.

voltsis_vaggelis@yahoo.gr

**Figure 8.18:** Log in page.

## Sign up

When the user selects the sign up button the screen of Figure 8.19 is displayed, prompting him to enter some personal information for signing up. The user also has the option to log in if he already has an account to the system.


E-Care!

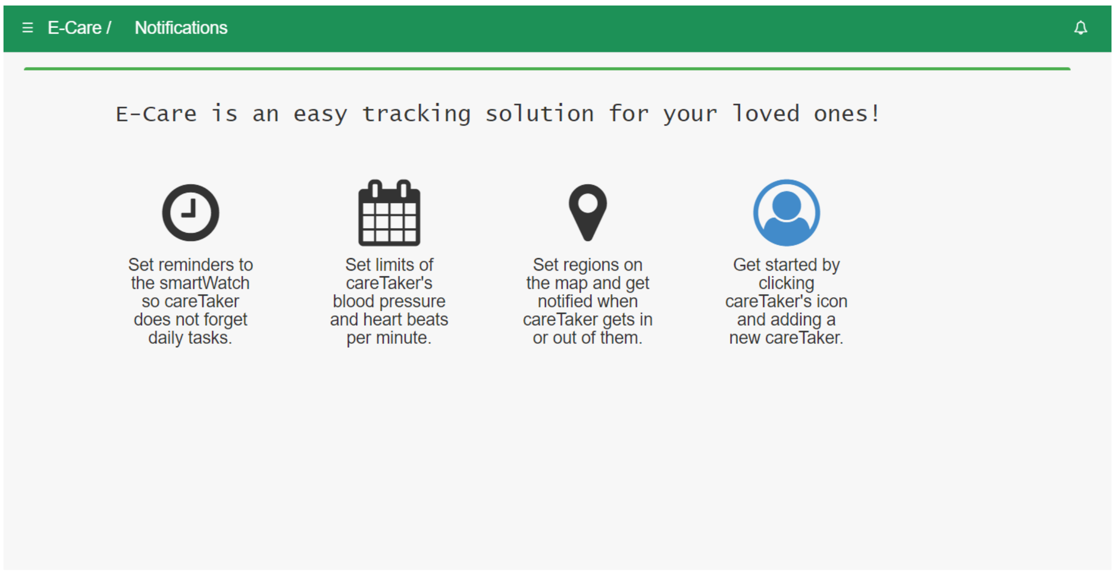
E-Care is an easy tracking solution for your loved ones.

- Get notified when careTaker's indications are not normal.
- Set regions and get notified when careTaker gets in or out of them.
- Set reminders and get notified when careTaker sees them.

**Figure 8.19:** Sign up page.

## Home Page

When the caregiver logs in the application for the first time and there are no caretakers added, the home page is presented (Figure 8.20), describing the main capabilities of the application. The user also has the option to add a new caretaker.



E-Care is an easy tracking solution for your loved ones!

Set reminders to the smartWatch so careTaker does not forget daily tasks.

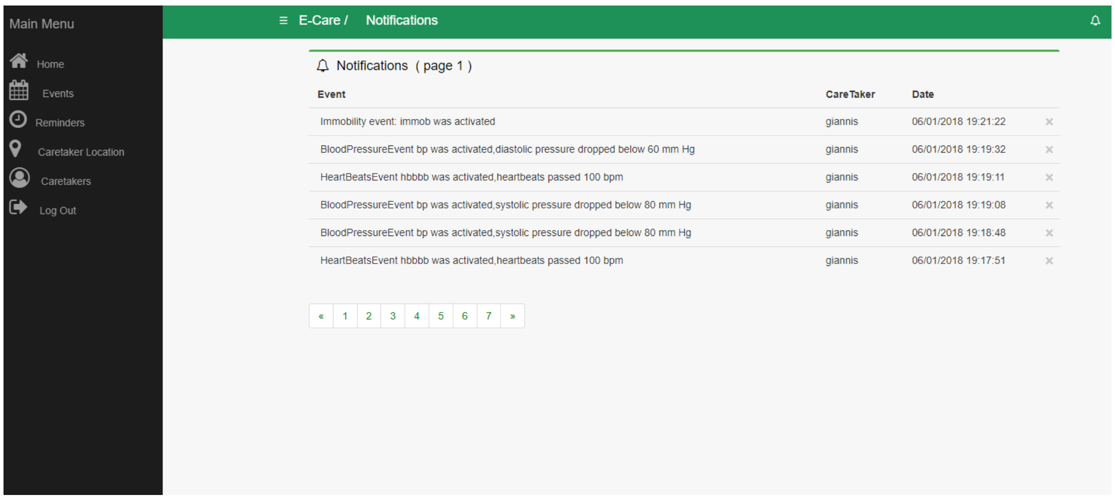
Set limits of careTaker's blood pressure and heart beats per minute.

Set regions on the map and get notified when careTaker gets in or out of them.

Get started by clicking careTaker's icon and adding a new careTaker.

**Figure 8.20:** Desktop - Home page 1.

If the caregiver has at least one caretaker added in the application then Figure 8.21 is displayed, showing the notifications of every caretaker.



Notifications ( page 1 )

| Event | CareTaker | Date |
|---|---|---|
| Immobility event: immob was activated | giannis | 06/01/2018 19:21:22 |
| BloodPressureEvent bp was activated,diastolic pressure dropped below 60 mm Hg | giannis | 06/01/2018 19:19:32 |
| HeartBeatsEvent hbbbb was activated,heartbeats passed 100 bpm | giannis | 06/01/2018 19:19:11 |
| BloodPressureEvent bp was activated,systolic pressure dropped below 80 mm Hg | giannis | 06/01/2018 19:19:08 |
| BloodPressureEvent bp was activated,systolic pressure dropped below 80 mm Hg | giannis | 06/01/2018 19:18:48 |
| HeartBeatsEvent hbbbb was activated,heartbeats passed 100 bpm | giannis | 06/01/2018 19:17:51 |

**Figure 8.21:** Desktop - Home page 2.

**Top bar**

- ***Menu hide/show:*** Hides or shows the left side menu.
- ***Home page button:*** Loads the home page of the application.
- ***Selected caretaker:*** Drop down button with which the caregiver selects the caretaker that is displayed.
- ***Connection icon:*** Shows if the caretaker's device is connected to the application.
- ***Battery icon:*** Shows the battery percentage of the caretaker's device.
- ***Notification button:*** Drop down button that shows all the notifications.

**Left side Menu**

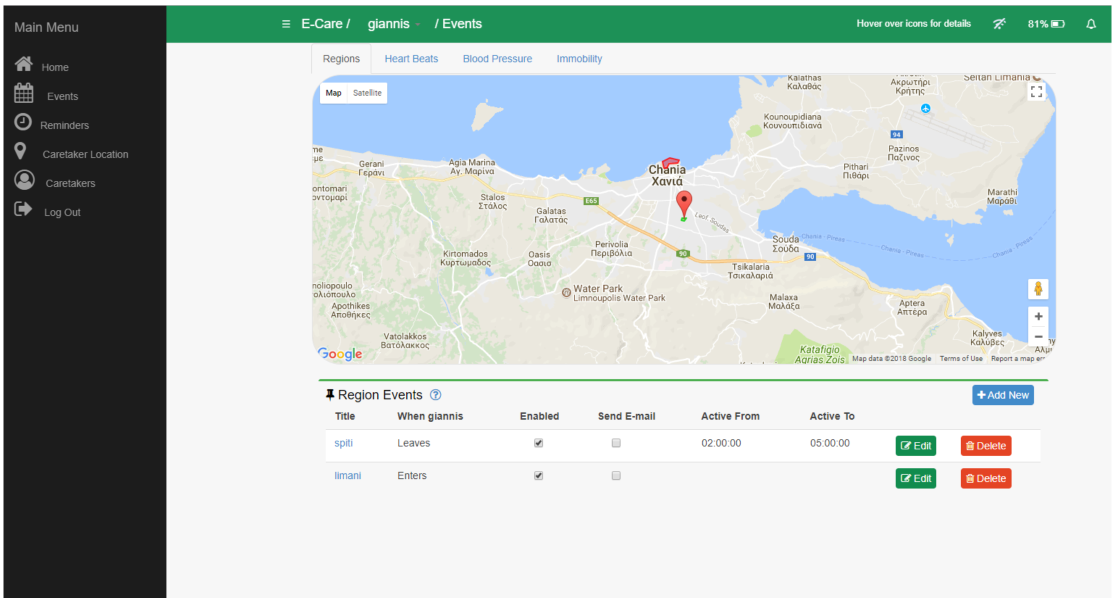
- ***Home:*** Presents the homepage of the application.
- ***Events:*** Shows the events of the selected caretaker.
- ***Reminders:*** Shows the reminders of the selected caretaker.
- ***Caretaker location:*** Shows the location of the caretaker on the map.
- ***Caretakers:*** Shows all the added caretakers.
- ***Log out:*** Logs out the caregiver.

**Management buttons**

- ***Edit button:*** Edits the information of the selected event, caretaker or reminder.
- ***Delete button:*** Deletes the selected event, caretaker or reminder.
- ***Show more button:*** Shows the information if it can not fit in the page.
- ***Add new button:*** Adds a new event, caretaker or reminder.
- ***Info/Question mark button:*** Shows some information about the present view.

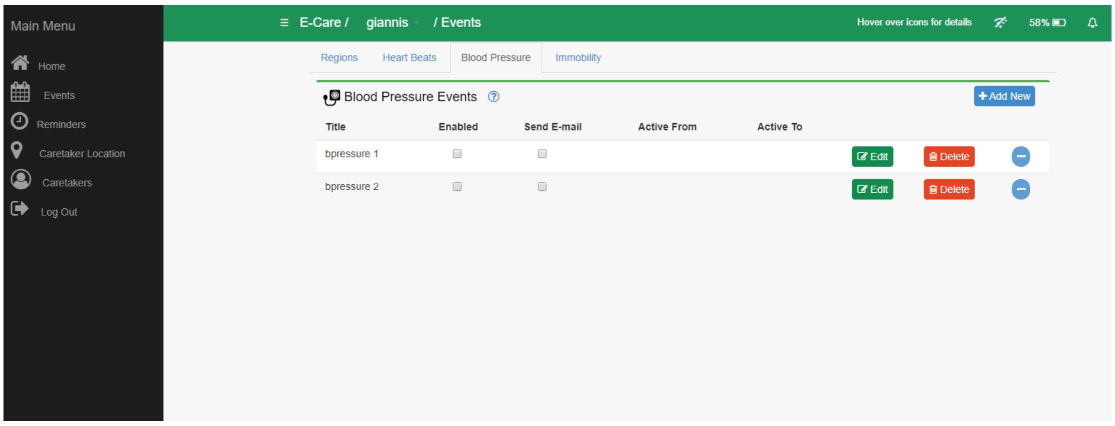
## Events

Figure 8.22 presents the Region Events page, Figure 8.23 shows Blood Pressure Events page, Figure 8.24 shows Heart Beat Events page and Figure 8.25 presents Immobility Events page.



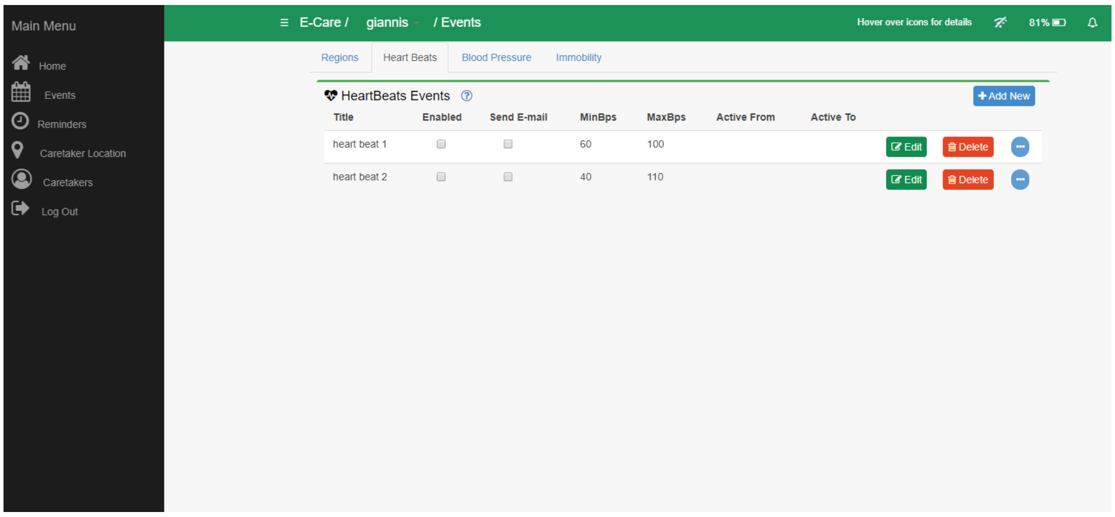
| Title | When giannis | Enabled | Send E-mail | Active From | Active To |
|---|---|---|---|---|---|
| spiti | Leaves | ✓ | | 02:00:00 | 05:00:00 |
| limani | Enters | ✓ | | | |

**Figure 8.22:** Desktop - Region Events page.



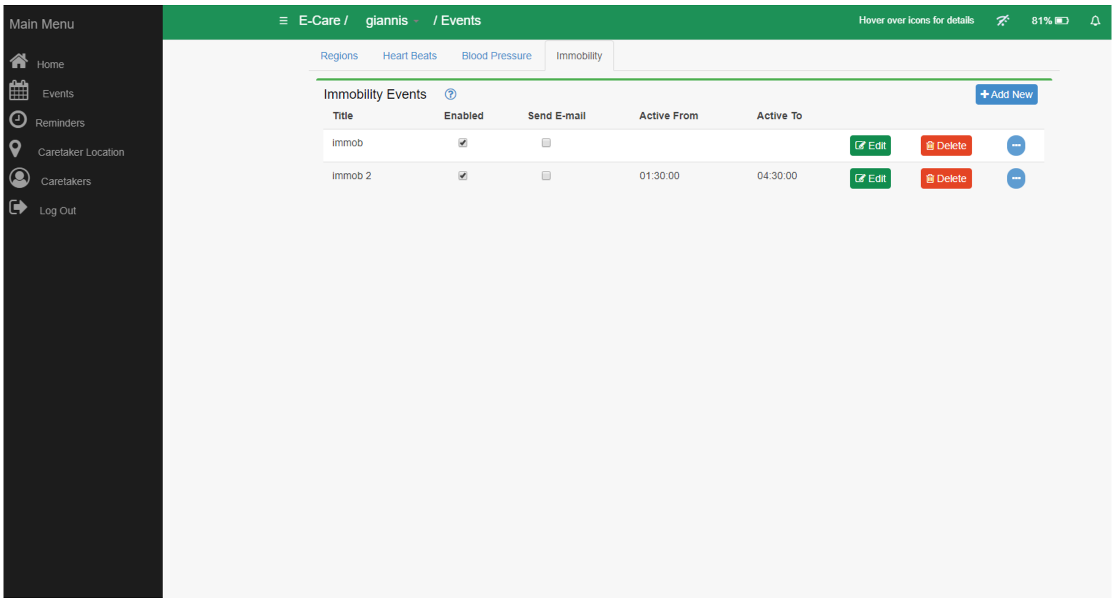
| Title | Enabled | Send E-mail | Active From | Active To |
|---|---|---|---|---|
| bpressure 1 | | | | |
| bpressure 2 | | | | |

**Figure 8.23:** Desktop - Blood Pressure Events page.



| Title | Enabled | Send E-mail | MinBps | MaxBps | Active From | Active To |
|---|---|---|---|---|---|---|
| heart beat 1 | ☐ | ☐ | 60 | 100 | | |
| heart beat 2 | ☐ | ☐ | 40 | 110 | | |

**Figure 8.24:** Desktop - Heart Beats Event page.

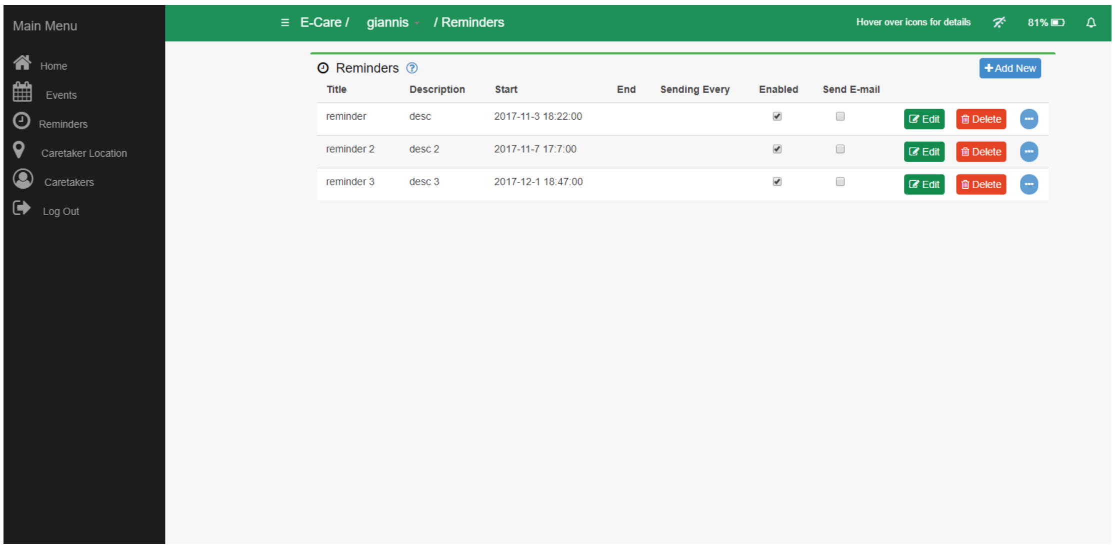


| Title | Enabled | Send E-mail | Active From | Active To |
|---|---|---|---|---|
| immob | ✓ | ☐ | | |
| immob 2 | ✓ | ☐ | 01:30:00 | 04:30:00 |

**Figure 8.25:** Desktop - Immobility Events page.

## Reminders

Figure 8.26 presents the Reminder page of the application in which all the added reminders are displayed to the caregiver.



| Title | Description | Start | End | Sending Every | Enabled | Send E-mail | | | |
|---|---|---|---|---|---|---|---|---|---|
| reminder | desc | 2017-11-3 18:22:00 | | | ☑ | ☐ | Edit | Delete | ⊖ |
| reminder 2 | desc 2 | 2017-11-7 17:7:00 | | | ☑ | ☐ | Edit | Delete | ⊖ |
| reminder 3 | desc 3 | 2017-12-1 18:47:00 | | | ☑ | ☐ | Edit | Delete | ⊖ |

**Figure 8.26:** Desktop - Reminders page.

## Caretaker Location

Figure 8.27 presents the caretaker location page in which the caretaker's last known location is displayed on the map.

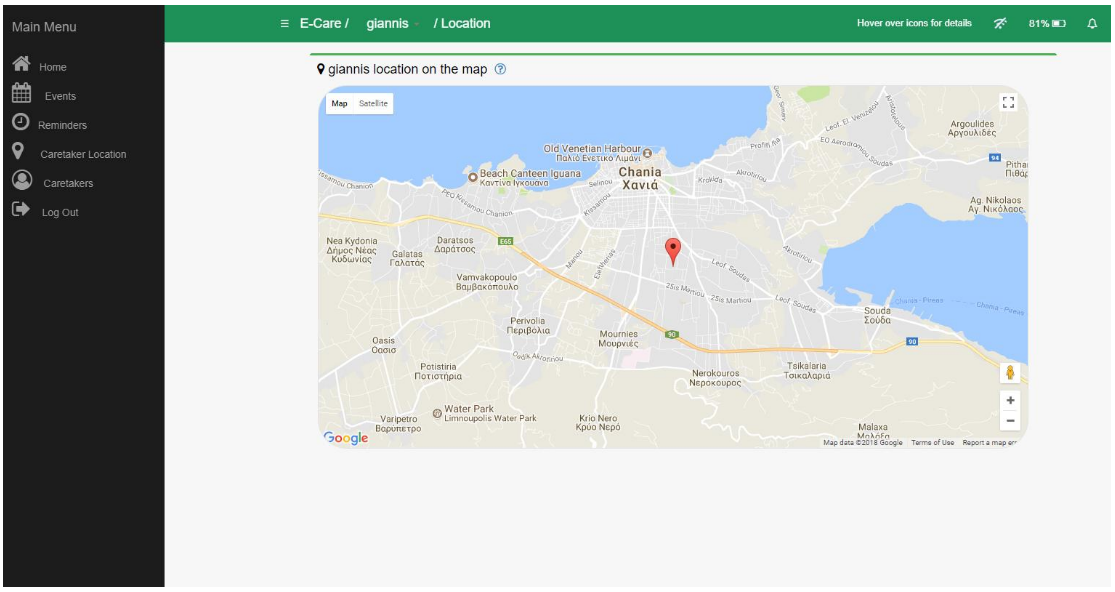
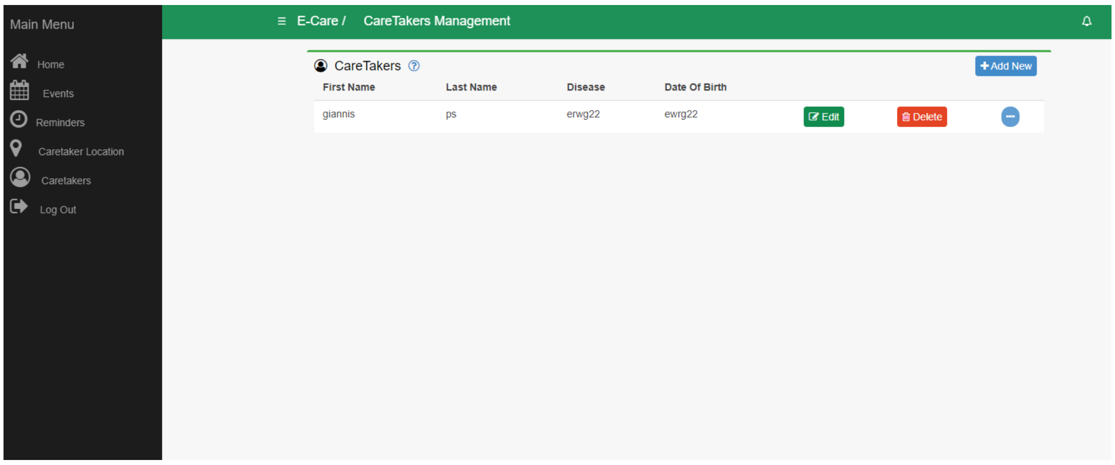


**Figure 8.27:** Desktop – The Caretaker's location page.

## Caretakers

Figure 8.28 presents the caretaker's page in which all the added caretakers are displayed. In the figure only one caretaker is added to the system.

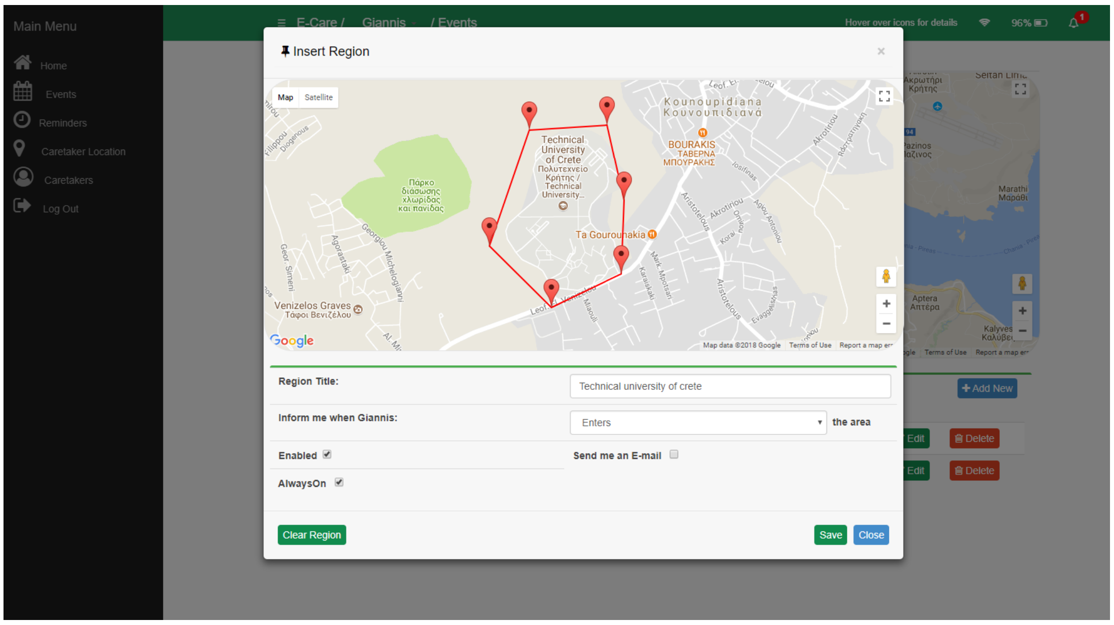


| First Name | Last Name | Disease | Date Of Birth |
|---|---|---|---|
| giannis | ps | erwg22 | ewrg22 |

**Figure 8.28:** Desktop – The Caretaker page.

## Insert/Edit Region Event

Figure 8.29 presents the user interface of the application when inserting a new Region event. The Caregiver defines the peaks of the area and also if he wants to get informed when a caretaker enters or leaves the area. The Caregiver has also the option ,like any other event or reminder, to enable or disable the event, get informed by e-mail in case that a caretaker leaves or enters the area and define hours that the event will be enabled. The edit screen is the same as the insert screen with the difference that the areas are filled with the event information.



Region Title: Technical university of crete

Inform me when Giannis: Enters the area

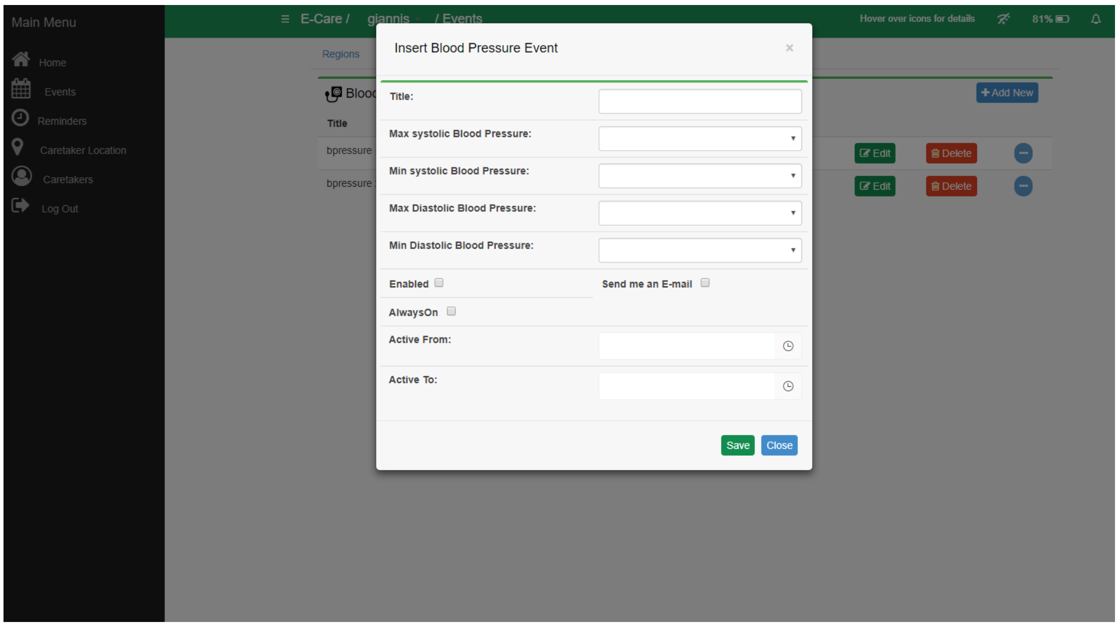
Enabled ✓ Send me an E-mail ☐

AlwaysOn ✓

**Figure 8.29:** Desktop – Insert/Edit Region Event page.

### Insert/Edit Blood Pressure Event

Figure 8.30 presents the user interface of the application when inserting a new Blood Pressure event. The Caregiver defines the title and also the caretaker's min and max blood pressure allowed limits. The edit screen is the same as the insert screen with the difference that the areas are filled with the event information.



Title:

Max systolic Blood Pressure:

Min systolic Blood Pressure:

Max Diastolic Blood Pressure:

Min Diastolic Blood Pressure:

Enabled ☐ Send me an E-mail ☐

AlwaysOn ☐

Active From:

Active To:

**Figure 8.30:** Desktop – Insert/Edit Blood Pressure Event page.

## Insert/Edit Heart Beat Event

Figure 8.31 presents the user interface of the application when inserting a new Heart Beat event. The Caregiver defines the title and also the caretaker's min and max heart beats per minute allowed limits. The Caregiver has also the option, like any other event or reminder, to enable or disable the event, get informed by e-mail and define hours that the event will be enabled. The edit screen is the same as the insert screen with the difference that the areas are filled with the event information.

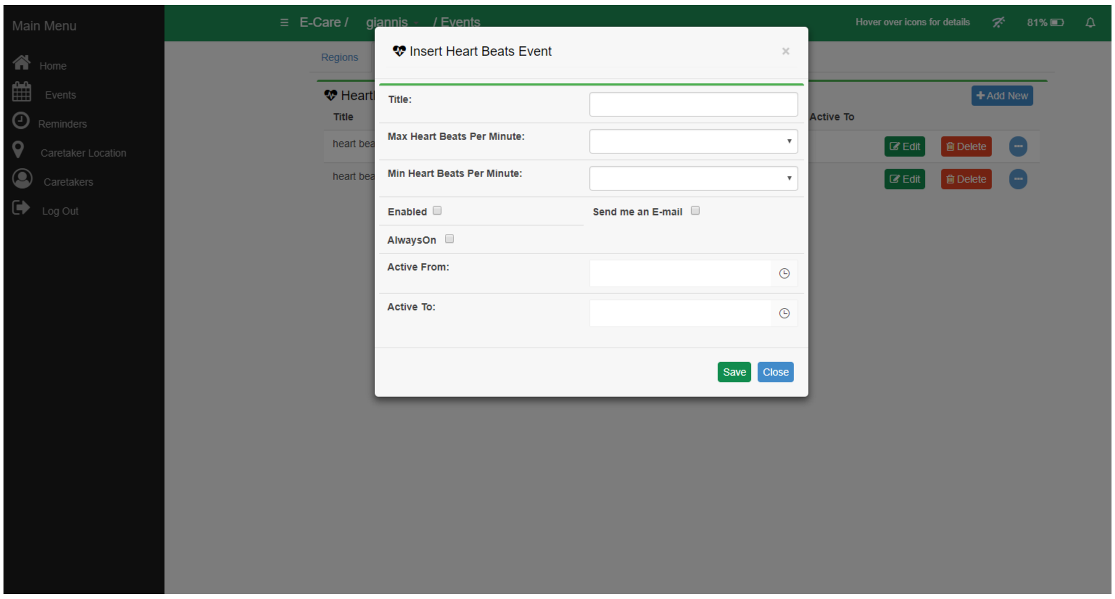


**Figure 8.31:** Desktop – Insert/Edit Heart Beats Event page.

## Insert/Edit Immobility Event

Figure 8.32 presents the user interface of the application when inserting a new Immobility event. The Caregiver defines the title and also the caretaker's max allowed minutes of immobility. The edit screen is the same as the insert screen with the difference that the areas are filled with the event information.

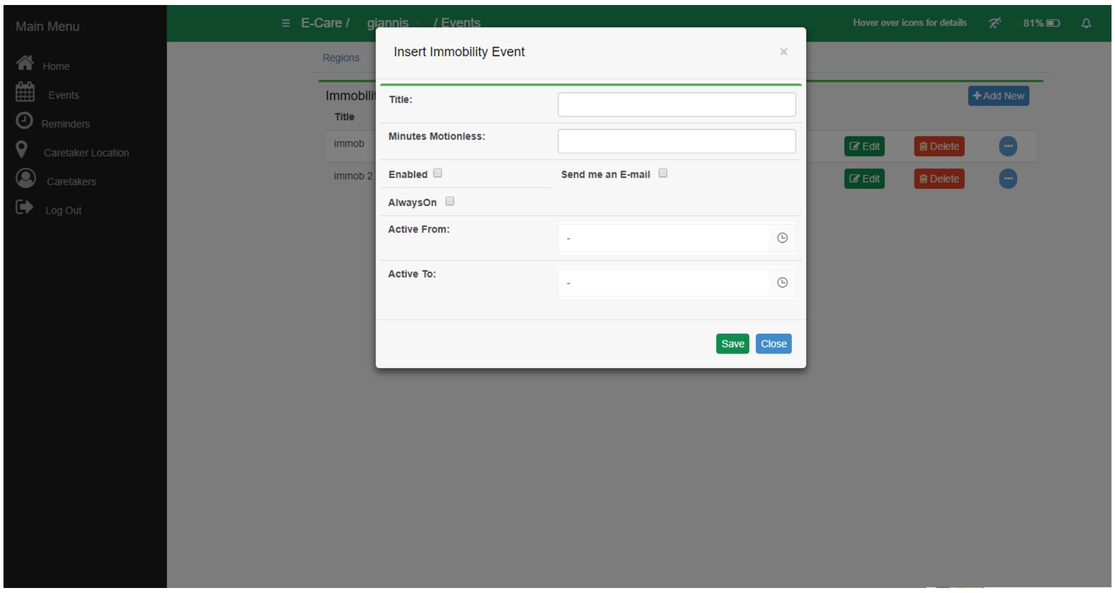


**Figure 8.32:** Desktop – Insert/Edit Immobility Event page.

### Insert/Edit Reminder

Figure 8.33 presents the user interface of the application when inserting a new Reminder. The Caregiver defines the title and the description of the reminder. He also has the option to enable or disable the reminder and to define a repetitive reminder that will stop at the date of his choosing.

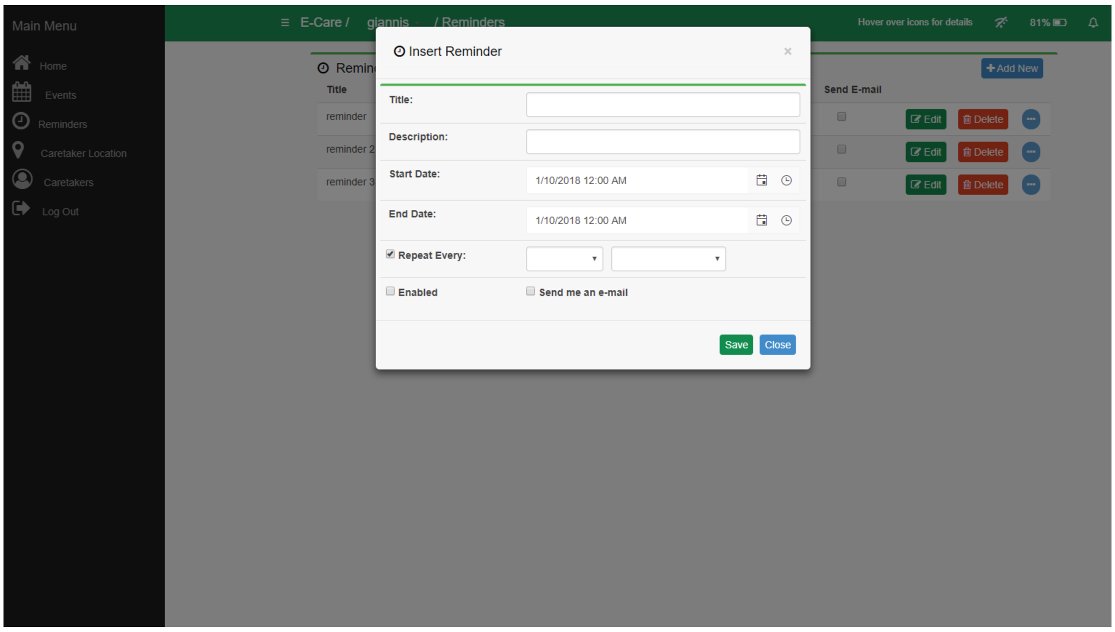


**Figure 8.33:** Desktop – Insert/Edit Reminder page.

## Insert/Edit Caretaker

Figure 8.34 presents the user interface of the application when inserting a new caretaker. The Caregiver has the option to insert the caretaker's first name, last name, date of birth, address, disease, and phone numbers as well as an emergency phone number that the smart watch will call in case of an emergency. The Caregiver must also insert a device ID which is displayed on the android application if the smart watch is not imported in the database. This ID is used for the connection between the android application and the web application. The edit screen is the same as the insert screen with the difference that the areas are filled with the caretaker's information.

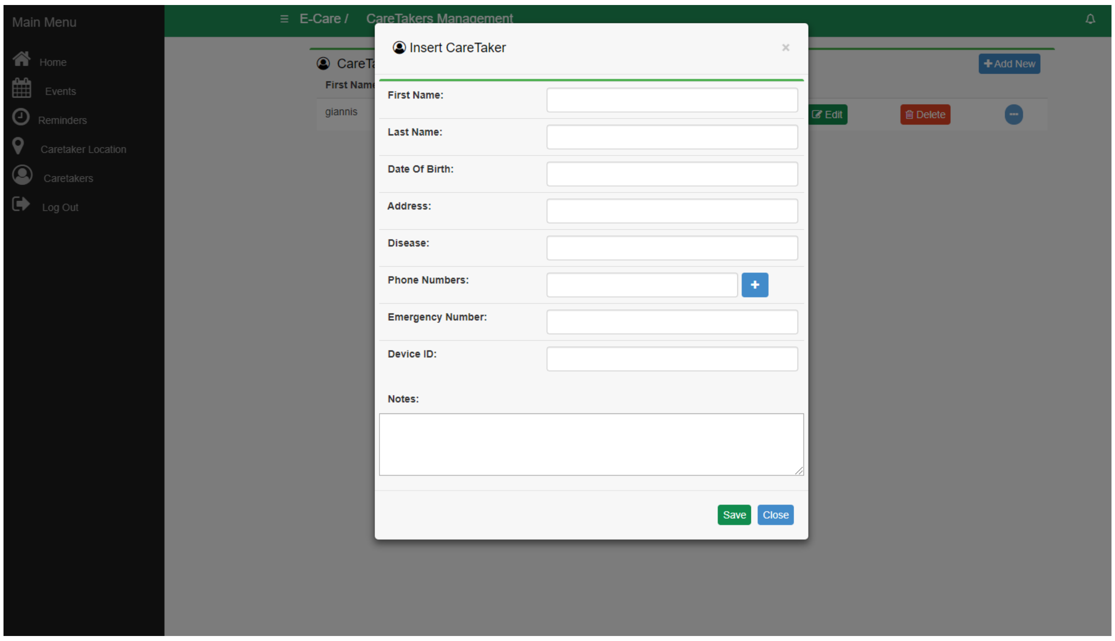
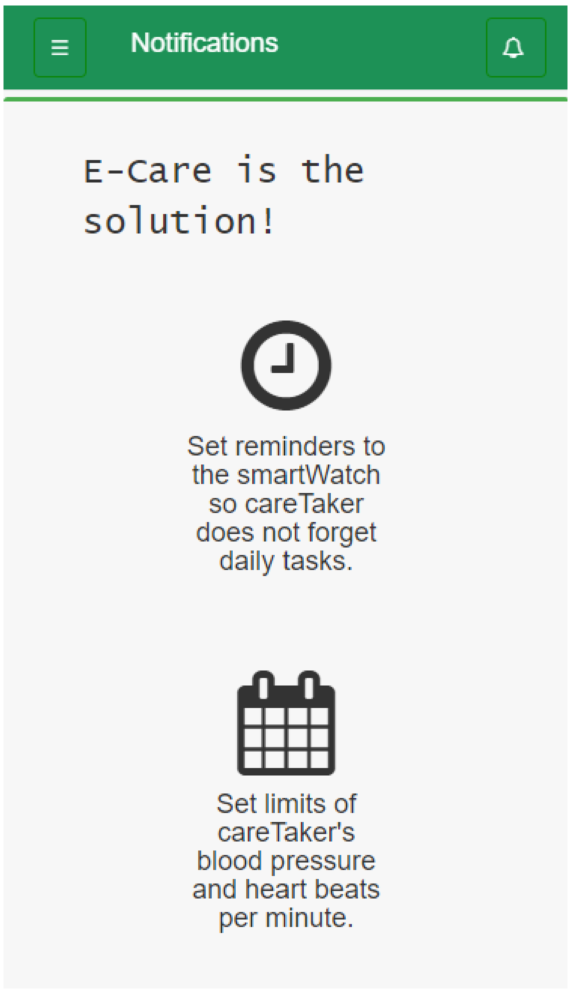


**Figure 8.34:** Desktop – Insert/Edit Caretaker page.

### 8.2.2 Web application on mobile device

#### Home Page

When the caregiver logs in the application for the first time and there are no caretakers added, the home page is presented (Figure 8.35) similar to the desktop application, describing the main capabilities of the application. The user also has the option to add a new caretaker. If the caregiver has at least one caretaker added in the application then Figure 8.36 is displayed, showing the notifications of every caretaker.

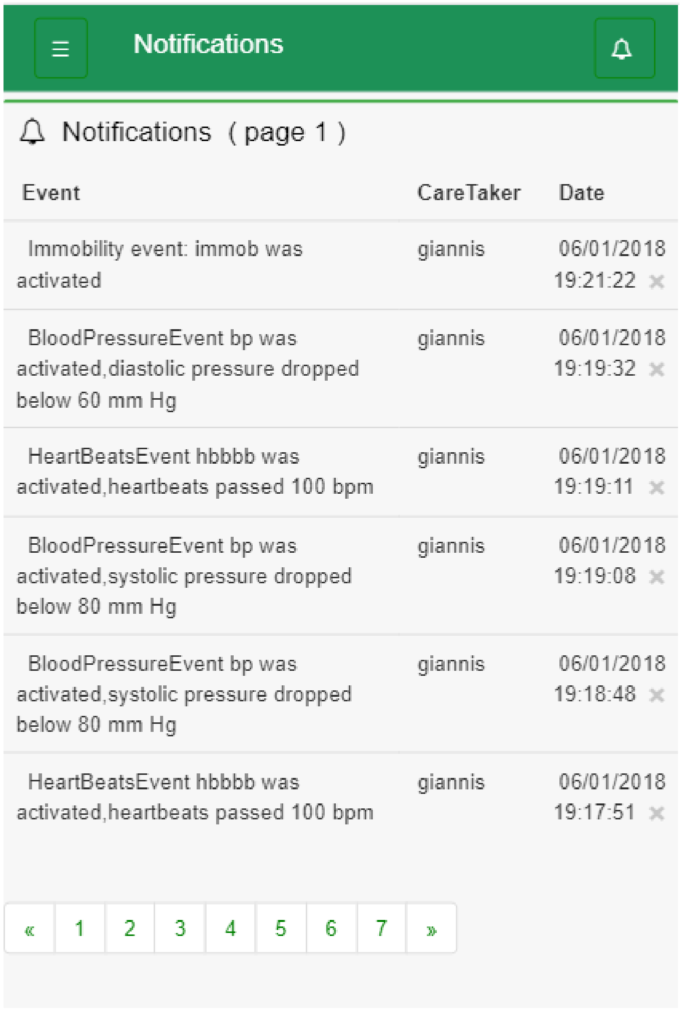


E-Care is the solution!

Set reminders to the smartWatch so careTaker does not forget daily tasks.

Set limits of careTaker's blood pressure and heart beats per minute.

**Figure 8.35:** Mobile device – Home page 1.



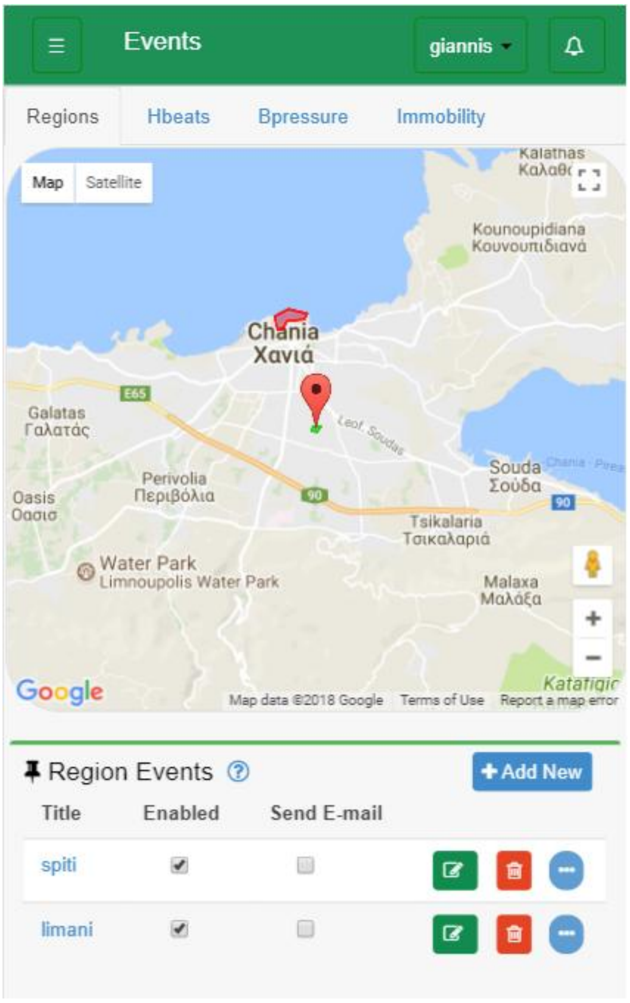
Notifications ( page 1 )

| Event | CareTaker | Date |
|---|---|---|
| Immobility event: immob was activated | giannis | 06/01/2018 19:21:22 |
| BloodPressureEvent bp was activated,diastolic pressure dropped below 60 mm Hg | giannis | 06/01/2018 19:19:32 |
| HeartBeatsEvent hbbbb was activated,heartbeats passed 100 bpm | giannis | 06/01/2018 19:19:11 |
| BloodPressureEvent bp was activated,systolic pressure dropped below 80 mm Hg | giannis | 06/01/2018 19:19:08 |
| BloodPressureEvent bp was activated,systolic pressure dropped below 80 mm Hg | giannis | 06/01/2018 19:18:48 |
| HeartBeatsEvent hbbbb was activated,heartbeats passed 100 bpm | giannis | 06/01/2018 19:17:51 |

**Figure 8.36:** Mobile device – Home page 2.

## Region Event

Figure 8.37 presents the user interface of the application showing the region events of the selected caretaker.

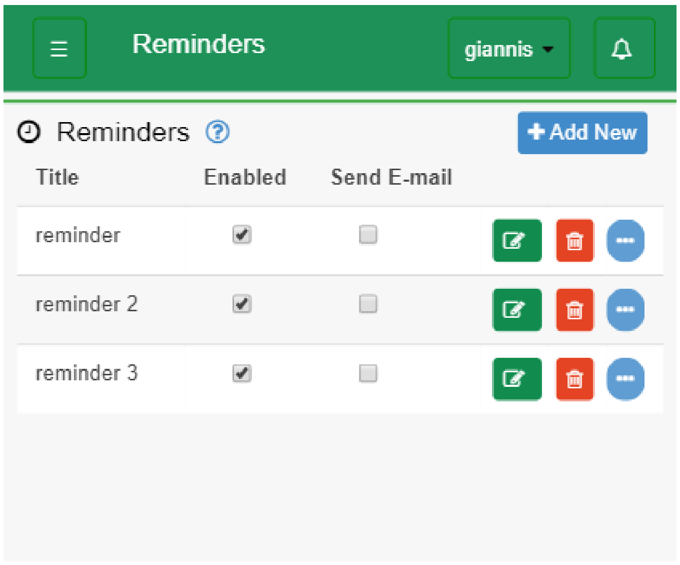


| Title | Enabled | Send E-mail |
|---|---|---|
| spiti | ✓ | ☐ |
| limani | ✓ | ☐ |

**Figure 8.37:** Mobile device – Region Events page.

**Reminder**

Figure 8.38 presents the user interface of the application showing the reminders of the selected caretaker.

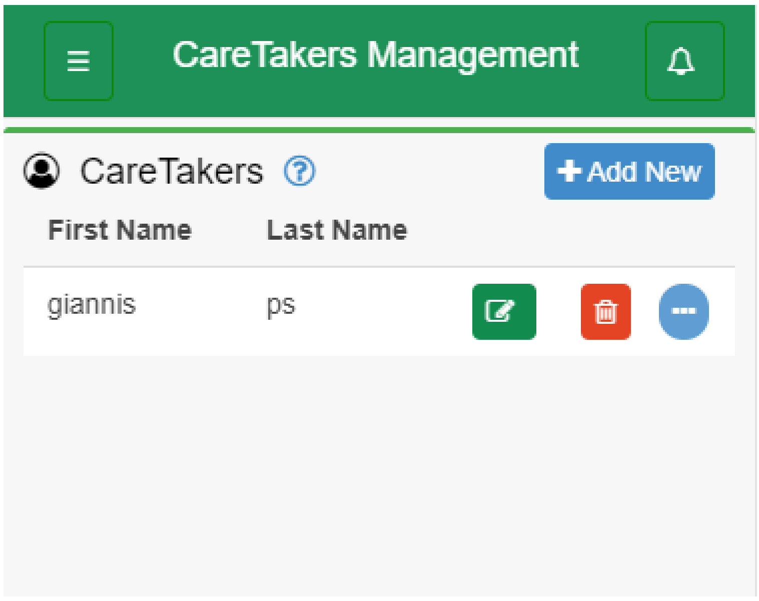


| Title | Enabled | Send E-mail |
|---|---|---|
| reminder | ✓ | ☐ |
| reminder 2 | ✓ | ☐ |
| reminder 3 | ✓ | ☐ |

**Figure 8.38:** Mobile device – Reminders page.

**Caretakers**

Figure 8.39 presents the user interface of the application showing all caretakers.



| First Name | Last Name |
|---|---|
| giannis | ps |

**Figure 8.39:** Mobile device – The Caretaker page.

### Insert/Edit Region Event

Figure 8.40 presents the user interface of the application when inserting a new region event. The edit screen is the same as the insert screen with the difference that the areas are filled with the event information.

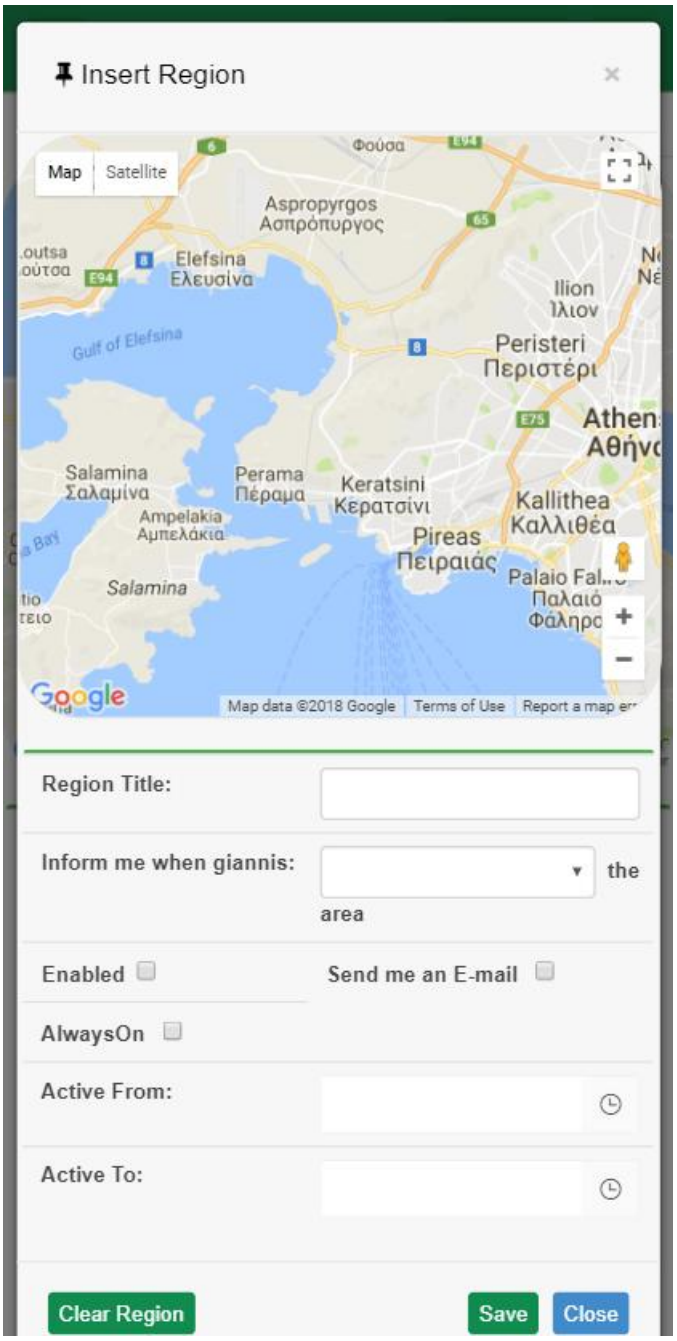
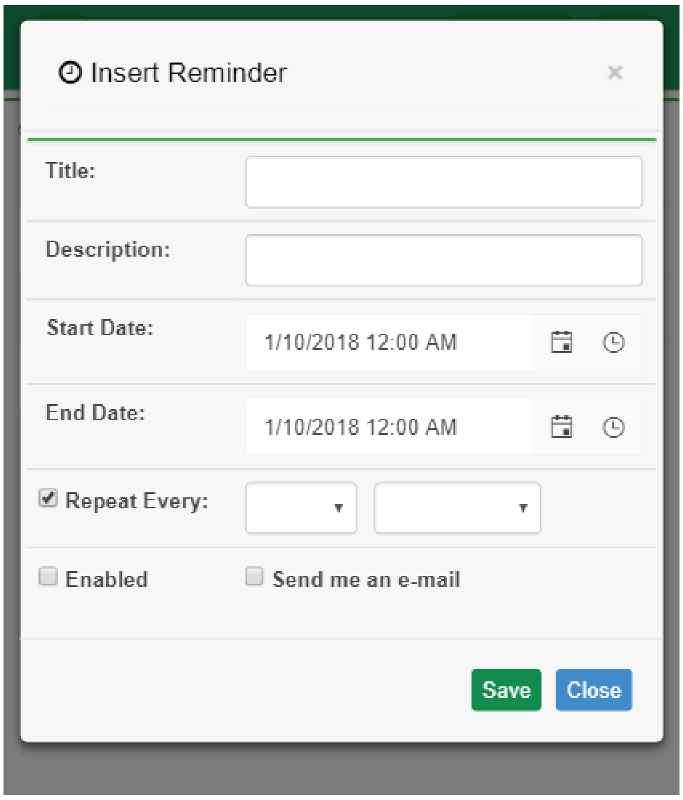


**Figure 8.40:** Mobile device – Insert/Edit Region page.

## Insert/Edit Reminder

Figure 8.41 presents the user interface of the application when inserting a new reminder. The edit screen is the same as the insert screen with the difference that the areas are filled with the reminder information.


Insert Reminder

Title:

Description:

Start Date: 1/10/2018 12:00 AM

End Date: 1/10/2018 12:00 AM

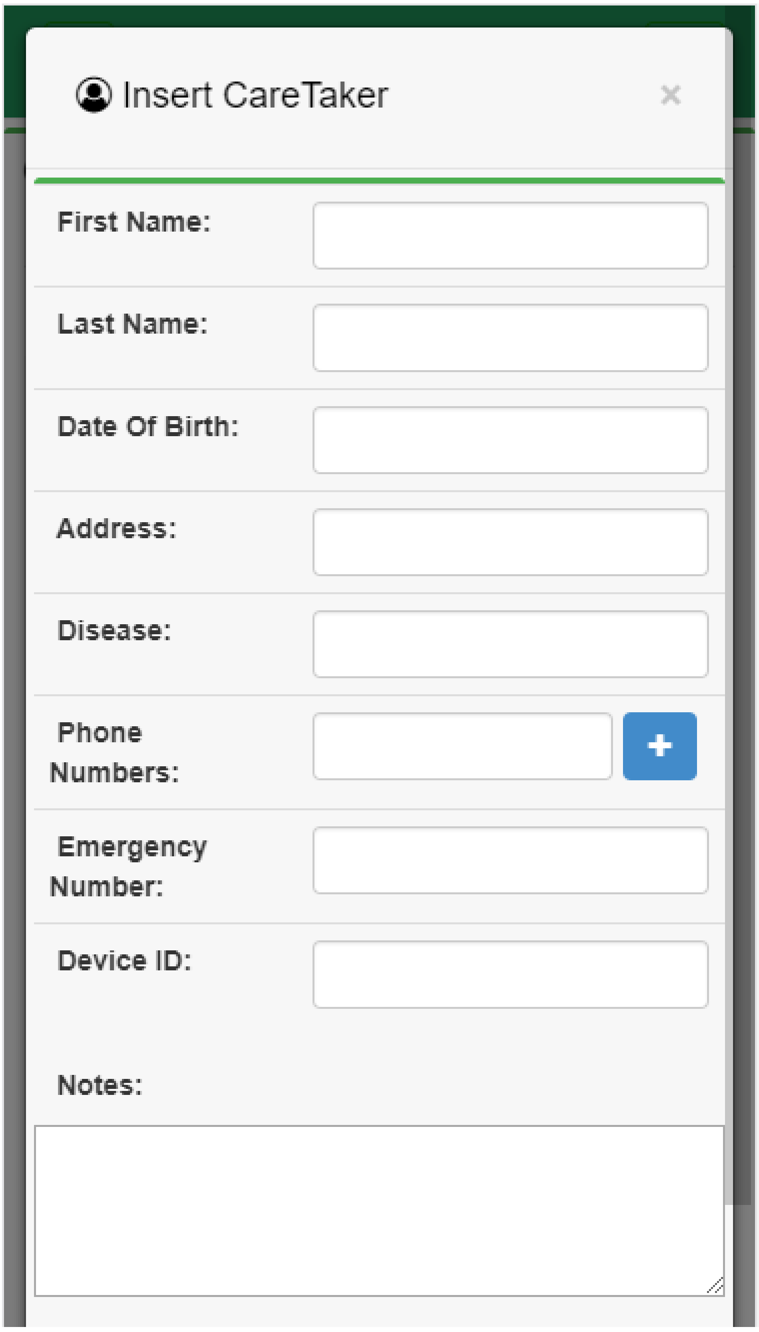
[✓] Repeat Every:

[ ] Enabled [ ] Send me an e-mail

**Figure 8.41:** Mobile device – Insert/Edit Reminder page.

**Insert/Edit Caretaker**

Figure 8.42 presents the user interface of the application when inserting a new caretaker. The edit screen is the same as the insert screen with the difference that the areas are filled with the caretaker's information.


Insert CareTaker ×

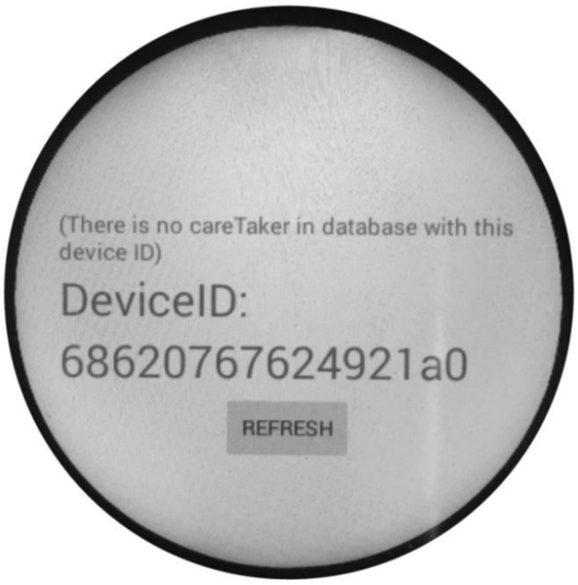
| | |
|---|---|
| First Name: | |
| Last Name: | |
| Date Of Birth: | |
| Address: | |
| Disease: | |
| Phone Numbers: | + |
| Emergency Number: | |
| Device ID: | |

Notes:

**Figure 8.42:** Mobile device – Insert/Edit Caretaker page.

### 8.2.3 Android Application

Figure 8.43 presents the user interface of the android application when the device is not submitted in the system by the caregiver, showing the ID of the device that is used for the connection with the web application and Figure 8.44 presents the user interface when the user is at the main screen. Figure 8.45 shows the scheduled notifications scroll screen and Figure 8.46 shows the emergency call screen.


(There is no careTaker in database with this device ID)

DeviceID:

68620767624921a0

**Figure 8.43:** Android application – First run screen.

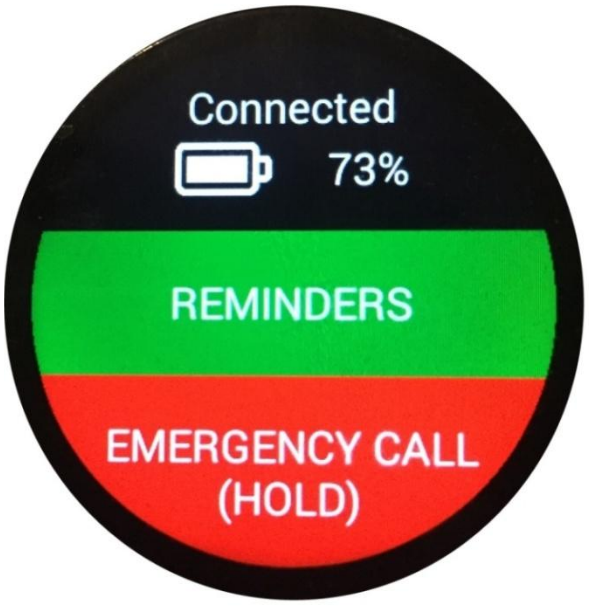
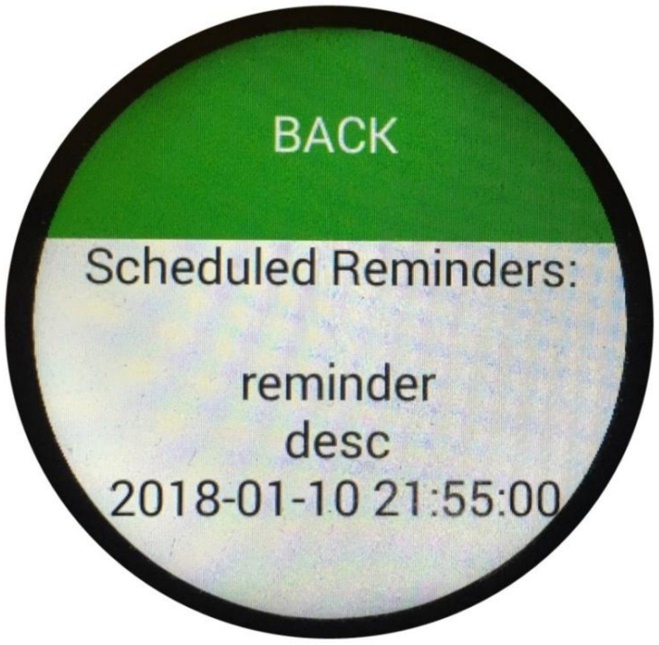


**Figure 8.44:** Android application – Main screen.



Scheduled Reminders:

reminder
desc
2018-01-10 21:55:00

**Figure 8.45:** Android application – Reminder screen.

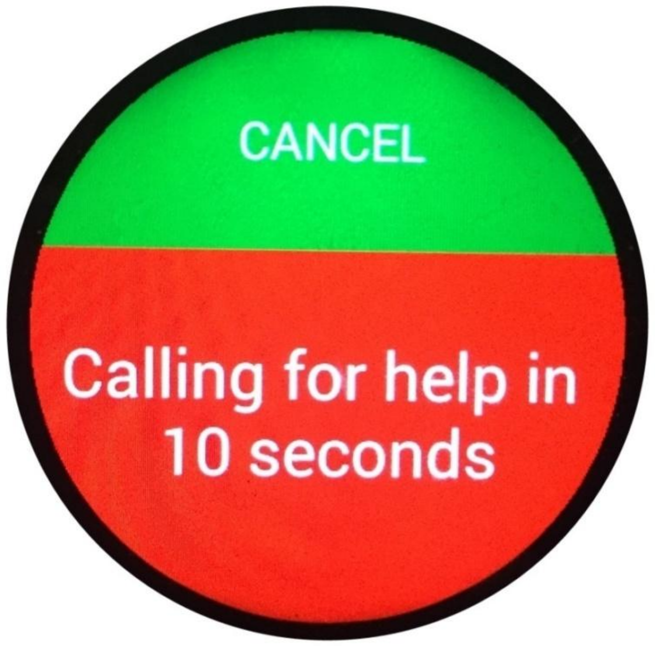


**Figure 8.46:** Android application – Emergency screen

# Chapter 9

## Conclusion & Future Work

In this thesis we presented the design and implementation of a system for helping families and persons with disabilities. The main objectives of our application were to: (1) bring peace of mind to families that have family members with disabilities or children. (2) help professionals that deal with a lot of patients and would like to monitor them constantly. (3) make the persons that will wear the smart watch feel free and more safe.

More thoroughly, we designed a web application supporting the creation and management of some events with which the caregiver is able to know if something is wrong with the caretaker. Moreover, we developed an android application that is running on android devices and especially in smart watches. The web application is compatible with most mobile devices/platforms and of course with laptops and desktop computers.

Both the web and android applications have been designed with flexibility and extensibility in mind. Also, they have been evaluated for their usability from some users that used the application for this purpose and extensive paper prototyping. Our future work includes the following: (1) replacing simulation of heart beats and blood pressure of the caretaker with the actual measurement from the smart watch device; (2) giving caregiver the option to view the caretaker's blood pressure or heart beat graphs (per month or per year); (3) giving caregiver the option to use smart watch camera in case of emergency; (4) falling detection of the caretaker; (5) direct connection of smart watch with other home devices; (6) option to make complex search between the registered caretakers or events.

# Bibliography

**[1]** AngularJS https://docs.angularjs.org/guide

**[2]** Pro AngularJS Authors: Adam , Freeman

**[3]** A. Cockburn. Writing Effective Use Cases. Addison-Wesley Professional, 2001.

**[4]** Android https://developer.android.com/index.html

**[5]** W3schools https://www.w3schools.com/

**[6]** A. van Kesteren, J. Aubourg, J. Song, and H. R. M. Steen. XMLHttpRequest Level 1. World Wide Web Consortium, Working Draft WD-XMLHttpRequest2-20080930, January 2014.

**[7]** MySQL https://www.mysql.com/

**[8]** Websockets www.websocket.org

**[9]** Stack Overflow https://stackoverflow.com/

**[10]** R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology 2002.

**[11]** M. D. Network. Introduction to Object-Oriented JavaScript.

**[12]** M. W. Newman and J. A. Landay. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In Symposium on Designing Interactive Systems 2000.

**[13]** T. Reenskaug. The Model-View-Controller (MVC) Its Past and Present, 2003.

**[14]** Icons http://fontawesome.io/

**[15]** A. L. Hors, P. L. Hégaret, L.Wood, G. Nicol, J. Robie, M. Champion, and S. Byrve. Document Object Model (DOM) Level 3 Core Specification. W3C Recommendation, April 2004.

**[16]** G. Alonso. Web Services: Concepts, Architectures and Applications. Springer, 2004.

**[17]** Heuristic Evaluation. https://en.wikipedia.org/wiki/Heuristic_evaluation

**[18]** Jacob Nielsen, R.L. Mack. Usability Inspection Methods 1994.

**[19]** Think Aloud protocol. https://en.wikipedia.org/wiki/Think_aloud_protocol

**[20]** Wikipedia https://www.wikipedia.org/