Design and development of a mobile system for clinical data exchange in the MADep project

Author: Jaume Ollés Barberán
Advisor: Juan Vera del Campo

June 19, 2012
"Victory is for the most persevering"

*Napoleón Bonaparte*
Resum

MADep és un sistema de monitorització per ajudar els pacients depressius, per tal de millorar el seu tractament utilitzant informació de la seva vida quotidiana. Aquest projecte mèdic vol ser una eina útil per comunicar els metges amb els pacients, creant una nova experiència per tractar la depressió. Aquest projecte final de carrera és responsable de la part d’intercanvi de dades clíniques d’aquest complex projecte que involucra diferents entitats treballant juntes seguint un objectiu comú.

Com a un sistema de monitorització que vol ser el més fiable possible, MADep es centra en la vida del pacient, monitoritzant el seu comportament a casa. Els metges consultaran la informació al seu lloc de treball, sense implicar el pacient en aquesta acció. L’intercanvi de dades permet a MADep ser un sistema complet de treball, separat en ambdós llocs però treballant com un sol sistema.

En aquest projecte final de carrera es detallen els passos seguits i les tecnologies utilitzades per tal que aquesta comunicació sigui possible. Smartphones i servidors s’utilitzen per crear aquesta interacció entre la casa del pacient i l’hospital per tal de fer aquesta comunicació tan flexible i forta com sigui possible.

Abstract

MADep is a monitoring system to assist depressive patients in order to improve their treatment using information about their daily lives. This medical project wants to be a helpful tool to communicate doctors with patients, creating a new experience to treat depression. This master thesis is the result of the clinical data exchange part of this complex project that involves a few entities working together pursuing the same target.

As a monitoring system which wants to be as reliable as possible, MADep is focused on the patients’ life, monitoring their behaviour in their house. Doctors will check the information from their workplace, but will not involve the patient in this action. Data exchange permits MADep to be a complete system working separately in both places as a single system.

This master thesis will detail the steps followed and technologies used to make this communication possible. Smartphones and servers are used in order to create this interaction between patient’s house and hospital in order to make this communication as flexible and strong as possible.
Partners

MADep (www.madep.cat) is a project with the support of the Generalitat de Catalunya and ACCIÓ. These partners participate in MADep:

Microart. www.microart.cat

Sant Joan de Déu - Serveis de Salut Mental (SJD-SSM). www.pssjd.org

Fundació I2CAT. www.i2cat.net

Centre de Recerca Biomèdica - UPC (CREB-UPC). www.creb.upc.es


Acknowledgments

I would like to thank all those people who made this trip possible, beginning in September 2005 and finishing today. Without every one of them it wouldn’t have been possible. These 6 years have become one of the most important periods of my life and I have been involved in some changes, meeting many people who have enriched my life.

First of all I would like to thank those people who helped me in this master thesis, guiding and helping me in each step, making this experience possible, Juanvi and Josep, my director and draftsman.

To my parents, Manel and Neus, and my sister Anna, who have helped me since I was a child and are responsible for what I am today, especially as person.

To my girlfriend, Neus, who has supported me every day and given me encouragement to continue even in the worst moments.

To my friends, colleagues and flatmates who have supported me during this trip and beyond and made me grow.

Thanks again to all of you, it would never have been the same if anyone of you hadn’t been there. This master thesis is as much yours as mine.
# Contents

1 **Introduction** 1
   1.1 Main targets ........................................... 2
   1.2 Document structure ................................... 2

2 **Scenario: MADep** 5
   2.1 What does MADep treat? ................................. 5
   2.2 How does MADep work? ................................ 6
   2.3 System at patient’s home ............................... 6
      2.3.1 Sensors ........................................... 7
      2.3.2 Virtual Assistant ................................. 8
      2.3.3 Data Management ................................. 9
      2.3.4 Indicators ....................................... 9
   2.4 System in the Hospital ................................. 10
      2.4.1 Data Store ....................................... 10
      2.4.2 Medical interface ............................... 11
   2.5 How to connect both systems? ....................... 11

3 **Technologies** 15
   3.1 Android ............................................... 15
      3.1.1 Android Architecture ............................. 16
      3.1.2 Components ..................................... 17
      3.1.3 Intents .......................................... 18
      3.1.4 The Manifest File ................................ 19
   3.2 Databases ............................................. 19
      3.2.1 MySQL and SQLite ................................ 20
      3.2.2 Hibernate ....................................... 21
   3.3 Web Services .......................................... 23
      3.3.1 Tomcat ........................................... 23
      3.3.2 HTTPS ........................................... 25
      3.3.3 JSON ............................................ 26

4 **System at the patient’s home** 29
   4.1 Use case ............................................... 30
      4.1.1 Description of the use case .................... 30
      4.1.2 Targets and Requirements ...................... 31
   4.2 System at the patient’s home Architecture .......... 32
      4.2.1 Main structure .................................. 33
1 Introduction

Nowadays, technologies are all around us, providing entertainment, useful services for our daily lives, new medical treatments, security, and so on. In particular, Smartphones are one of the most important devices invented in the last decade, and seem to be a great potential device capable of working on any idea coming from the human mind.

Smartphones are more than basic voice-communications devices with gaming capabilities. During the last years, they have become similar to pocket computers and allow developers to create new and complex applications that make our life easier. Smartphones let us access the Internet to check our mail, chat with many other people in real time, consult a worldwide geographical map, take photos, organize our lives, and many other services.

On the other hand, medical science is evolving too with the passing of the years, the numbers and efficiency of medical treatments have increased over the last decades, developing new vaccines and treatments to save lives. Many physical illnesses have been studied all these years and some of them have been definitely cured.

Unfortunately, mental disorders are still difficult to study and treat. They are not visible and doctors can’t act in front of them as they do in front of physical illnesses. The human brain is one of the most complex mechanisms in the universe and just a small section of its internal secrets have been revealed. This fact makes mental disorders much more difficult to treat and eventually, cure.

In many cases, doctors only provide treatments that soften the symptoms of mental disorders. Usually, these treatments involve interaction between doctors and patients in order to find a solution to their problem and help them to cure themselves. Doctors have become experts on mental disorders along the years and have developed some treatments that are really helpful to the patients, but they are not enough to give the patient a personal and more concise treatment.

Depression is one of the most extended mental disorders all over the world. The standard treatment includes meetings at the hospital in order to understand the source of the problem, study the current status of the patient, track the evolution of the patient, and try to find a path to follow in order to improve this behaviour.

Depression is a long-term mental disorder and these meetings between doctors and patients must be periodically arranged. Many of depressive patients are not admitted in a hospital and live at their own home. In this case, doctors don’t know how the patient’s life is between the periodical meetings, they can’t imagine what the current situation is and whether or not the treatment is working. The lack of a daily-based update of the patient’s conditions is a huge handicap in the treatment of depression.

Monitoring depressive patients while they are at their home is the main objective of the MADep project.
1.1 Main targets

The objective of MADep project is the design and development of a computational tool to support and assist the remote treatment of people with depression. With the support of the "Generalitat de Catalunya" and "ACC1Ó" and the participation of "MicroArt", "Sant Joan de Déu - Serveis de Salut Mental", "Fundació I2Cat" and "Universitat Politècnica de Catalunya", this project will become a useful tool for doctors to monitor depressive patients in their own houses.

MADep provides a bridge in the communication between the patient and his doctor. MADep is a monitoring system placed in the patient’s house in order to keep the doctor informed of the evolution of the patient with depression. The system lets the doctor know how the patient behaves at his home, what he does and track the evolution of his physical and mental state.

MADep uses smartphones as the core of the monitoring system due to its flexibility and potential. The different partners of the project will work together in order to obtain a whole system able to control data collection, data transfer and a user interface to manage all the information that is collected at the patient’s home. Electronic sensors will create a network that collects information about the behaviour of the patient and his environment. The data collected at the patient’s home is saved in a smartphone and sent to the doctors who, finally, access the information from the hospital. This system approaches the patient to the doctor, so in between meetings the medical team will be informed of the evolution of the patient. Extra helping tools are used in MADep too, such as alerts, evolution graphics and a Virtual Assistant.

1.2 Document structure

The structure of this document is as follows:

Chapter 2: Scenario  The first chapter shows us what MADep really offers to doctors and patients, explaining the whole system in detail. The structure of the system will be shown and the structures of every section will be clearly divided, showing the part that is especially developed in this master thesis, responsible of the clinical data exchange between the smartphone and the main server.

Chapter 3: Technologies  Technologies and protocols used in the MADep project are carefully listed and detailed in this chapter, explaining their advantages and their utility in the system.

Chapter 4: System at the patient’s home  MADep data exchange system is divided in two main parts, System at the patient’s home and System at the Hospital. This chapter details the first part, the System at the patient’s home. It focuses on the work done in the smartphone since the data is received from the sensors, using a database to store all this information and explaining how we manage it. This information, previously serialized, will be sent to the MADep main server using HTTP protocol. All these points will be explained in detail in this chapter.
Chapter 5: System at the hospital  Second part of the MADep data exchange system is built as a server. This system at the hospital will receive data from all the patients included in the MADep treatment. This server is in charge of this data and will work as a listening server and an information provider as well. Smartphones will send information to the server, which will store it into a central database. When doctors want to check this data, the server must provide them the information. Database technologies for servers, website languages and server structures will be detailed in this chapter.

Chapter 6: Demonstrator  This chapter will show how the whole MADep system works, detailing every step and explaining how the system must be used. Screenshots of the smartphone application and the website will help us to understand the interaction between user and system.

Chapter 7: Conclusions  This chapter will show the achieved targets, new ideas, and future improvements for MADep. As well as personal achievements, lessons learned and future lines.
2 Scenario: MADep

Monitoring and Assistance for the Depression (MADep) is an autonomous system consisting of sensors, different kinds of networks, a smartphone, a private application, and so forth. This system monitors the behaviour of a depressive patient in his house using a network of sensors. The information collected by these sensors is stored and transferred to the hospital. Doctors will check this information in the hospital itself in order to know how their patients are evolving in their houses.

This project is supported by the "Generalitat de Catalunya" and "ACC1Ó". Many entities collaborate in this project, such as "MicroArt", "Sant Joan de Déu - Serveis de Salut Mental", "Fundació I2Cat". Many departments of the "Universitat Politècnica de Catalunya" participate in this project too, some of them are "Centre de Recerca en Enginyeria Biomèdica (CREB-UPC)" , "Soft Computing Research Group (SOCO-UPC)" and "Information Security Group (ISG)". This master thesis consists on the communication module and data exchange of MADep, and it has been developed by the ISG group of the UPC.

2.1 What does MADep treat?

Depression is a mental illness that affects between 5% and 10% of the population of the Western Europe, and many people have suffered it in their lifetime [1]. The WHO (World Health Organization) has reported that the suicide is one of the 10 most common causes of death all over the world, and that the suicide rate in depressive patients is estimated between 6% and 15%. In our country, the results from ESEMeD and SAMCAT [2] show that 23,7% of Catalan people older than 18 years will suffer from depression during their lives, particularly, the 15,2% have a higher depression. And after 35-64 years, this problem is more frequent.

Nowadays the treatment for depression consists of drugs and psychotherapies that the patient should include in his daily life. This is an external treatment that does not include a patient participation. Ideally, this patient should be involved in an active way in all the phases of his treatment, and be capable of identifying any situation or bad habit that lead him to relapse or a risk phase [3] [4]. It is known that every patient is affected by depression in a different way [5], that's why each one has different needs. The ESEMeD study [2], cited above, shows that 38,9% of Catalan patients are treated only with drugs, 6,9% of the patients are treated using psychological treatment and the rest receive both forms of treatment.

Unfortunately, the treatment of depression doesn’t always seem to be effective. Apart from the 40% of the patients that recover and the 20% who show an important improvement [6], the rest of them have no response to the treatment, or just a little. The reason for the failure of the treatment have not yet been identified, but it seems that the premature termination of the treatment has been a major cause.

There are new methods that are being used to try to solve this problem. New therapies
focused on the technology placed on the patient’s location [7], such as self-help websites that help patients to improve their behaviour, seem to work well in certain cases. But sometimes, leaving all the responsibility to the patient doesn’t work at all, and the patient should be under medical supervision.

This is where MADep has an important effect. It was developed to be a bidirectional system that allows the patients to receive support for their behaviour, to know what their bad habits are in real time and how to solve them. But it also includes a system that allows the doctor to be aware of the conduct of their patients. This compound structure permits a better relation between the patients’ lives and habits, and his medical treatment. As well as avoiding communication problems when the patients go to the hospital. All their behaviour has been monitored during the period since their last session.

2.2 How does MADep work?

MADep would be a complete system with an autonomous architecture, consisting of several smaller parts developed by different project partners, including electronic sensors, Bluetooth networks, smartphone technologies and applications, web services through the internet network (Wi-Fi and 3G), local servers in the hospital and so on.

The main target of MADep is to collect information about the habits and behaviour of these patients in their own homes, to manage this information in each particular case, and display it in an easy and understandable way for the medical staff in the hospital itself.

To manage all these technologies and fix them in a whole singular system, MADep consists of two major parts:

- System at patient’s home: Involving a few sensors distributed around the house that will collect the information about the patient’s lifestyle and send it to the smartphone.

- System in the hospital: Consisting of an understandable interface for the medical staff, that will show tables, medical history and special messages for each individual patient.

In particular, the work done in this master thesis is the connection of both parts of MADep. It is the interface that allows the smartphone to talk to a server placed in the hospital. Without this interaction of both parts, the MADep project makes no sense. It would be a sensor network storing information without a target and an isolated static server working without new information to be treated and studied.

2.3 System at patient’s home

The data acquisition module of MADep works inside the house of the patient. It includes many parts gathering data in different ways, but all of them work together in order to get a complete report of how the patient lives. Obviously, MADep is absolutely not a control system of people’s life, it only gets information from the habits related to the conduct and lifestyle of the depressed person. So, with the authorization of the patient, the house is equipped with sensors in many important areas and furniture, such as the bed, the weight balance, the fridge
and so on. All these sensors, as it is said, work in an independent way, but they send the collected information to an access point, a central brain, the smartphone.

### 2.3.1 Sensors

There will be a few sensors installed in different parts of the house. Initially it is planned to include these sensors:

- **Bed sensor**: this sensor will tell us how many times per day the patients lie on the bed, and the amount of time they spend in it. A person who sleeps more than 8 hours per day tends to increase their depression habits, and so does sleeping in short and several times every day. Doctors will evaluate their sleeping habits and decide if they are consistent with an improving behaviour.

- **Weighing scale sensor**: patients should weigh themselves everyday in order to collect information about how their weight varies during the treatment. A sharp increase or decrease in the patient’s weight should be registered and shown to the medical staff as an important change.

- **Fridge sensor**: this sensor will show how many times the patient opens or closes the fridge, and when it happens. This will tell the doctor how patients feed themselves. Particularly, this will be an indicator used to study the alimentary habits of the patient and compare them with the desired habits.

- **Moving sensor**: the smartphone itself is a potential sensor. It can be programmed to collect data about the movement of the phone. So if we are able to study the phone’s movement and process it in a smart way, we will have the best sensor in the patients’ home, “themselves”. It is true that the patients have to carry the smartphone with them all day. But, if we can acheive this, it will be the best way to know what the patient’s motor activity is. The medical staff could study how much time the patient walks, stays seated, does exercise and so on.

Most of these sensors are not in the smartphone itself so we have to design a system to transfer their data to the smartphone.

Transferring all the information collected by the sensors, distributed around the house, should not involve any user interaction. This would create a refusal attitude by the patients, and would lead to a lack of interaction. This is a not an optimal way to work because MADep, apart from being a monitoring system, tries to include patient’s activity in this process. A person who has homework to do everyday, such as weighing himself or answering the Virtual Assistant’s questions, tends to increase their activity compared to a person who has nothing to do all day.

To solve this problem, the system is ready to work as a Wireless Network System, specifically assembled as an ad-hoc network. This method permits MADep to work in an area without an access point, solving the problem of setting up a more complex WLAN in each patient’s house. In addition, MADep tries to be an autonomous system, as being independent from a phone line or electrical system, makes it more robust.
One of the best options for this particular case is to include Bluetooth communications in each sensor and transfer data to the smartphone using its own Bluetooth interface. Since the sensors try to measure the patient’s activity they will be near the smartphone when the interaction takes place. So, in that moment, the data exchange could be done by Bluetooth technology without any problem. Every sensor will be equipped with Bluetooth, some of them do include it already, as the smartphone does. An ad-hoc network implemented by Bluetooth will be in charge of data exchange in the inner house system including sensors and the smartphone. Figure 2.1 shows the interaction between the patient and the sensors and the data return using Bluetooth technology.

![Figure 2.1: Image showing the interaction Patient-Sensors and Bluetooth connection](image)

2.3.2 Virtual Assistant

Apart from sensors, MADep includes another method to report the habits and the emotional condition. We call it the "Virtual Assistant". It consists of an interface developed on the smartphone that will try to emulate a human being. It will communicate with the patients in a familiar way, show them methods to improve their lifestyles, give them some advice, and so on. In addition, from time to time patients will be asked some questions referring to their mood and emotional condition.

The Virtual Assistant will study the answers and the data collected by the sensors, so it should be able to talk to the patients in a coherent way according to their mood. This will create a better environment for the patients to be honest with their answers and their treatment.
monitoring. This will also allow the Virtual Assistant to communicate with the patients in the best way related to their treatment phase: initial, intermediate or advanced.

This Virtual Assistant will collect information about the emotional condition and habits of the patient and, considering some aspects, will study the case and issue a diagnosis. Specialists working on the psychological area already develop all of these tests used. So, this test will be collected, studied and inserted as MADep standard tests.

2.3.3 Data Management

All the data collected by the smartphone, both sensor and smartphone itself, are saved on a local internal database. This information is not understood by anybody if it shown as it comes to the smartphone. Consisting of different kinds of sensor impulses, amount of times, switches and answers coming from the tests, which don’t give us any idea of what is happening. So this information should be managed, studied and shown in a simple and visual way.

This data management will consist of different kinds of studies, some of them will use mathematical processes, and others need a signal processing and many others a "human" interpretation. For example, the weight collected from the weighing scale will be just shown if it’s daily information, or an average weight should be obtained if there are many captures in a day.

Some of this data contains just electric signals coming from sensors. Obviously electrical signals don’t make sense to a doctor, they need to see what is really happening there. So a signal processing is needed in these cases. The system should be able to recognize patterns of conduct, exceptions and problems in the patient behaviour and present it in a few numbers or characteristics, which can be included in a comprehensible document.

Other data doesn’t need this kind of processing but a psychological study. Tests and conversations used by the Virtual Assistant come from a huge work previously done by the medical staff. The Virtual Assistant, using many algorithms included in the application, studies the response to these kind of indicators. It considers aspects such as previous answers in the same and different tests, the evolution of the patient’s treatment, external measures coming from the sensors, the patient’s medical history, etc. The tests shown by the Virtual Assistant have been obtained from studies already done on the psychological area. These tests are standards, they can be shown to depressive patients, in order to get information on their mood.

2.3.4 Indicators

MADep is a new and revolutionary system working in patients’ houses to monitor depressive conduct. First tests only serve a small number of patients, but future plans include dozens of patients at the same time. A doctor can manage one, two or three patients simultaneously but, what if these three patients become 300?

Doctors need more than numbers and signals, they need indicators that can show evolution, alarms, good behaviour, etc. All the signals, treated by the smartphone, should be shown in an understandable way.

Some indicators have been accorded with the medical staff to show them what is happening in a fast and dirty way. Graphics of evolution, for example, could show how the patient’s weight is evolving. All the answers from the test, and the reaction of the patient in front of the Virtual Assistant could be transformed into a simple phrase or paragraph explaining to the doctor what
the patient’s emotional state is. Or even other indicators could be processed as a few "traffic lights" just showing whether the evolution is positive, negative or stable. Allowing the doctor to see what is happening in the patient’s treatment in a short period of time makes it possible to treat a higher number of patients at the same time.

2.4 System in the Hospital

While the system in the patient’s house works as a self-contained network, another system has been developed in the Hospital, where the doctors query and study the patient’s information. 

This system will have two parts, one consists of the collection and storage of data, and the other is the interface that doctors will consult. These kind of systems try to be transparent to the final user, so doctors will only see what they are really interested in. Every backwards system should be able to provide this information to the medical interface when doctors ask for it. So this brings us to another autonomous system placed in the hospital. Both parts together make the hospital system, used by doctors to see how the patient’s treatment results. Doctors should be able to control a few patients at the same time without leaving their office. As it has been said, this system consists of two parts, Data storage and Medical interface.

2.4.1 Data Store

All the data collected in every patient’s house should be stored on a main server, and consulted by the medical staff. A database containing every patient’s medical history, data received from MADep in the house system, alarms, messages, treatments, and so on, will be included in this server. This database must be "anonymous" and secure, because it contains every patient’s private medical data. Doctors will access the database with an identifier number for every patient, which only they will know, or not even that.

Constructed as a huge database, this will link patients to their medical history. It will include past and present data related to the treatment, a calendar which shows the evolution of the
patient throughout the phases, incidents involving this patient, alarms being activated, messages showing how the treatment is evolving, and many others aspects. This database must be easily extendable in the future, to ensure expansion and improvement of the system.

2.4.2 Medical interface

Doctors need to see the information from their patients in a quick, easy, and comprehensible way. They don’t really need the patient’s weight everyday during the evolution. Or even how many minutes the patient sleeps every day, or when it happens, but if it is included in proper behaviour.

It is necessary to develop an effective interface that can be used by the medical staff without much trouble. For this purpose, some indicators and graphics will be created by this interface with the information coming from the database. Medical staff and developers will, again, work together to create a useful interface consisting of a website or an internal program, containing what doctors need to know and the backwards code that permits access to the database, as well as the code that will convert patient data into graphics and messages.

This Interface will contain graphics showing the evolution of certain parameters and traffic lights indicating if the evolution is positive or negative. Messages will show the doctor what the emotional reaction and evolution of the patient is. And the system will permit the medical staff to save notes about how the treatment evolves, and what the next step to follow would be.

There will be alarms which would be treated as special events too. These will react with a different action, such as sending an email or an SMS, indicating the urgency of the case. Doctors should determine these alarms previously.

In a future improvement of the system, some other ideas could be included. For example, Doctors may call the patient to arrange a visit. Or even send them some advice and personal messages related to the treatment.

Another improvement could be to consult and compare a group of patients and see how differently they evolve, which parameters seem to follow a common way of reaction, or even if one patient is far away from the group average evolution.

2.5 How to connect both systems?

We got two self-contained and autonomous systems working separately. They are able to work without any problem with the data they have stored, and they can upgrade new information coming from their own system. But we need every patient’s information in the main system at the hospital, we need it regularly and in a short period of time. So, what is the solution?

Can you imagine the patient going to the hospital with the information and transferring it physically to the main server? Or even the doctor going to every patient’s house to collect this data and bring it to the main database? It was previously said that the whole system should be transparent for both patient and medical staff. So involving any of these parts in the data exchange is not an optimal solution. We need to solve this problem by developing the data exchange layer in order to transfer this data in a comfortable way for both patients and medical staff.
This is where my master thesis comes in. Like the rest of the MADep project, this work consists in two main parts too. The patient’s house part will be covered by a smartphone working on Android’s OS, and the hospital’s system will consist of a server receiving data from all the patients through the internet network.

The Android OS will contain a Service working in background, which will contain alarms, clocks, data exchange protocols, security, database access and privacy methods to preserve patient anonymity. The data stored on the local database, coming from the sensors and the Virtual Assistant, will be consulted by this part of the MADep application, managed, stored in data exchanging packets and sent through the internet to the system at the hospital.

The server, which will receive the data from all the patients, will consist of a listening port receiving HTTPS messages containing all the information from a patient’s message. This information should be extracted from the package and included in the main medical database, containing all the patients’ information, medical history, evolution and treatment data. Allowing the rest of the hospital system to query this database in order to create the medical interface. Figure 2.3 shows the complete structure of MADep, presenting the data exchange area developed in this master thesis.
Figure 2.3: Complete MADep structure and Data Exchange Area
3 Technologies

This chapter includes a detailed description of the technologies that are directly involved in this data exchange project. It is good to remember that the project architecture contains two main parts, System at the patient’s home and System at the hospital. Some technologies take part in the System at the patient’s home architecture (Android, SQLite), and others are just included in the System at the hospital (Hibernate, MySQL, Servlets). But other kinds of technologies affect both parts, these include HTTP and JSON.

First of all, we will talk about Android, the OS supported by the smartphone that contains basically all the work done in the patient’s home system. As it was previously presented in section 2.3, sensors distributed around the patient’s house collect data and then send them to the smartphone, where the information is stored, managed, and finally sent to the system at the hospital. All this work on the smartphone is carried out by Android.

On the other hand, we’ll take a look at the technologies used in the system at the hospital. Some of them, such as Servlets and JSP, are in charge of receiving data from the network, while others (Hibernate) are in charge of the storage of this data in the database. They are explained in more detail in sections 3.2 and 3.3. We will also take a quick look at web technologies that allow us access to information through a browser interface. Finally other technologies involved in the data exchange protocol such as HTTPS or JSON will be presented too, together with the ones that allow the data storage in databases, SQLite and MySQL.

3.1 Android

In October 2003, Andy Rubin (Co-founder of Danger), Rich Miner (Co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (Head of design and interface development at WebTV) founded Android Inc. in Palo Alto, California (USA). It began operation secretly, only showing the idea of working on mobile phone technologies.

Two years later, on August 17th, 2005, Google acquired Android, assuming the idea of a possible incorporation into the mobile phone market. Key employees, including some founders of Android Inc., stayed at the company when this became part of Google Inc.

On November 5th, 2007, the Open Handset Alliance was presented, consisting of several companies including Google, HTC, Intel, LG and many more. A year and two months later, on December 9th, 2008, 14 new members joined the Open Handset Alliance.

On the same day of its creation, the Open Handset Alliance, presented Android, a mobile device platform built on the Linux kernel, with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony.

The idea of Android as an open source platform was to grow up thanks to the third-party developers, and let anyone create their own application using standard methods to assure compat-
ility with any Android platform. This was regulated by AOSP (Android Open Source Project), which include SDK and NDK, allowing any Android platform to run any application written by third-party developers. Nowadays Android has become one of the most important platforms used by smartphones all over the world. In the United States, the market of smartphones based on Android platform increased from the middle of 2010, to getting the first position with a 43.6% of sales in the third term of 2010 [8]. Worldwide, it reached 50.9% of the market in the last term of 2012, more than twice the second operation system (IOS from Apple) [9].

![Android versions distribution](image)

**Figure 3.1:** Image showing the percentage distribution of Android versions on April 2nd, 2012

Android has improved its version through these years. A study shown in the Android developer’s page [10], gives us an idea of the version distribution of Android all over the world. It is easy to observe that a large part of the percentage allows version 2.x. We should observe on section 4.2.2 that a version of Android 4.0 or later is one of the requirements of MADep’s system. On figure 3.1 we can observe the amount of smartphone models different versions of Android OS nowadays.

### 3.1.1 Android Architecture

Java programming language is the one that has been chosen to write Android applications. Using a developers environment, such as Eclipse, we have Android SDK tools available to create and develop an entire Android project. This Android SDK contains data and resource files, as well as Android libraries and many useful tools that permit the developers to work in a comfortable environment. When the Android project is compiled, an archive file with an .apk suffix contains the code which can be launched on an Android Emulator or a smartphone. The .apk file itself is large enough to contain all the code, is considered a single application and is the file used to install the application. When it is already installed, we do not need the developers environment anymore, the application lives in its own sandbox.

The application is launched when any one of the components that it contains is called by the user or another application. From then on, the application is active, and begins a lifecycle running all through the application components and their interaction. This process is shut
down when it is no longer needed or when the system needs memory. The decision as to which process will be shut down lies on a list of preferences depending on the privileges of the different kind of components and their importance in the process or the whole system.

All applications have a list of privileges. Previous to the final implementation of the application, we should be able to identify which privileges the application should have and ask for them in the manifest file. This ensures a secure environment accepted by the user, where the application has no access to those parts of the system to which it is not given permission.

However, it is possible to share information between applications and the data stored in the system itself. The application can request permission to use external devices, storage and even components of other applications. Some of these examples are cameras, SD cards, Bluetooth, maps and so on. The user has to accept this permission when the application is installed.

Android applications are composed of different kinds of components, each one performing a different role in the application. The interaction between them, the lifecycle running through the application, the calling of each component and the interface shown by each one, make the whole application.

Each component is a potential access point to the application from the system. Normally the components are called on between themselves to create all the application options, menus, and different ways to work on them. But some of the components can be initialized by an external process if they have the permission to do so and if the component allows it.

### 3.1.2 Components

There are four different types of application components. Consisting of different structures and purposes, they are the skeleton of the whole Android application. They work together, and each one has its own unique and necessary role. The dependency and interaction between them defines the application itself and the possibilities that this will give to the user.

- **Activities**

  They are the main components of the Android application. A Activity present a User Interface (UI) consisting of a single screen where the user can interact. When an Activity is called, the main screen related to it is opened, and the interaction with the user changes (in most cases), new buttons, options, menus, lists, and other Objects appear. This Activity usually contains actions too, which can call another Activity, close it, call another component, play a video or a song, and lots of other potential actions. An application use to be composed by some of these Activities that interact between them and that eventually form the application itself.

- **Services**

  A Service is a component that runs in the background, invisible to the user. It doesn’t have a user interface, and works while other Activities take place on the screen. The Services are used to perform long-running operations or to access remote processes. They are called on when we need the system to do a job while we use the smartphone for another purpose, and we cannot allow the terminal to be blocked by a component. Some examples of Services are playing music in the background while we are using the smartphone, or the upgrade of applications every certain period of time.
• Content Providers

A content provider allows all of the application to share data, and provides the tools to access this data in a more comfortable way. The data could be stored on a SQLite database, on a web server, on the smartphone or any other location. Examples of Content Providers are; access to the photos included on the SD card, accessed by the camera, a mail application or a file viewer. Content Providers are also used in private applications to store the data and share it among their components.

• Broadcast Receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements. When a broadcast occurs, for example when a download is finished or a the battery is low, the Broadcast Receivers usually does a little job, such as calling an Activity or a Service to act as a reaction to this broadcast. Or even to show notification bars or launch alerting messages.

As said before, Android applications have a few entry points, not just one. So there is no main() method in the code, just different and single self-initializing components. All of them connected in a complex structure of calling methods to create the impression of a self-containing application. What the user sees is a system that provides an application that can work in different ways, and permits the use of external components in the current application without any extra effort.

3.1.3 Intents

Activities, Services and Broadcast Receivers are initiated as individual components, they start each other and then, the thread comes back to the "caller", or it goes to another component. The developer decides which path will be followed in each case. And this creates the application flow.

The way to travel from one component to another and then come back, or follow another path, takes us to another important component of the Android architecture, the Intents.

Intent Object defines where we want to go or what we want to do. If we know the component we need in that situation, we can just include the URI in that Activity, Service or broadcast receiver. This called component will bring us back to the previous component with a result, if we were asking for it. But what really happens is that we just launch the new component and we wait for the return of the main thread. The called component will do its own work, and when it finishes, the thread will return or follow a new path created with a new Intent in the called component.

So, an Intent can be created in any Activity, Service or broadcast receiver in order to call another component that we need in that situation. Depending on the type of intent, we will wait for a response or we just call it and leave it to work.

The action of starting external components is carried out by the intents themselves. When we use an intent we specify whether we are calling an inner component or we need external application’s help to do that kind of work. The intent will do the work of searching for the kind of Activity we need, and it will start it for us.
In the case of Broadcast Receivers, the intent just launches an announcement of what is happening. So every application listening will receive the message and the chosen one will act as a reaction.

The last case, the content provider components are not called on by an intent. They don’t contain a running code but methods we will use to work with the information they contain. We access a content provider through different methods, such as query(), ContentResolver Object and so. Intents do all the work we need to interconnect different components along the lifecycle of our application. We just develop the Activities, Services, Broadcast Receivers and Content Providers and then we interconnect them using intents. This kind of architecture simplifies the structure of the application and leaves the door open to potential upgrades of the initial code.

### 3.1.4 The Manifest File

A manifest file is necessary to declare all the components that will be used by this application and the requirements they need to work on it, as well as the API required and any hardware configurations that will be used. This is reflected in the AndroidManifest.xml file included in all Android projects and is constantly checked by all the components.

AndroidManifest.xml file must be included in our Android project. This .xml file contains all the information needed to identify all the components that take part in the application, their ID’s, references, permission, etc.

It is important to know that any Activity, Service and content provider not included on the manifest file doesn’t exist to the system. That will never be called, or if they are, will never be found. In case of Broadcast Receivers, they can be dynamically created on the code itself, so they are a special case that avoid the manifest file.

### 3.2 Databases

Databases support all the data stored both in the smartphone and the server. They consist of a structured group of tables containing information of several patients (weight, sleeping hours, tests answers, ...) being edited by the application on the smartphone and by the Servlet in the system at the hospital. This information comes from outside the code. In the smartphone the data is collected from the sensors by Bluetooth and then inserted in the local database of the smartphone, a SQLite DB. When the data exchange begins, this database is queried, and the new information is obtained and sent to the system at the hospital.

In the server, the data comes from outside the Servlet through an HTTPS packet, as will be shown in section 5.3.1. Then it will be stored in a MySQL DB placed in the server itself, or in a remote one, accessed from the MADep main server.

After all, databases are the baseline that support all the information contained in MADep. Two of them are used, but why not use the same database in both parts of MADep system? The differences between MySQL and SQLite will be explained in the following section.
3.2.1 MySQL and SQLite

As just introduced, MADep will include two types of databases, MySQL and SQLite. Both of them are based on SQL databases but they are so different in many ways.

SQL is a declarative language to access relational databases, which allows you to specify several types of operations in these. The main structure of SQL consists of storing different tables not in the same file but in several separated tables. Using a file that manages the relationship between tables, SQL can give us the impression of a single huge database. When the user queries a database using SQL, he obtains a single answer coming from different tables which are presented in a single file that is shown to him.

MySQL

MySQL is a multiuser and multi-thread SQL. It is commonly used on Internet servers to assure the multiple access of several users simultaneously [11]. This allows MADep to be able to receive information from different sources at the same time. This is essential in this kind of system because the information exchange sequences are not controlled by the system at the hospital but by the smartphone. Try to imagine hundreds or thousands of patients trying to send their own information to the same server at the same time, it is not an option to keep them waiting on a buffer or, even worse, to lose the information. The system at the hospital should be able to store an amount of information coming from several patients simultaneously. This is possible thanks to MySQL work and its special characteristics as compared to other SQL languages.

Another benefit of MySQL is the option of having the data stored in a few servers placed in different places and even better with multiple storing motors. This makes it possible to increase the size of MADep to include a large number of users.

In addition, MySQL is open source software GPL (GNU General Public License), available to anyone and used in many Internet applications. Its effectiveness is reflected in its extensive use in the most important applications today. MySQL ensures a secure connectivity too, an important characteristic essential to the MADep system due to the privacy of the patient’s information.

SQLite

SQLite consists of a SQL based query language contained in a relatively small library written in C. The self-content form of SQLite makes it possible for it to be placed in an individual terminal, such as a smartphone, and to use it to manage all its internal data in a relatively easy way. In contrast to other SQL systems, SQLite is linked to the main project and works as a part of it. The use of SQLite in a project is carried out by inner calling functions and methods; this makes the interaction quicker and more efficient. All the data is stored in the same file blocked against any interaction between it and the main application [12].

The use of SQLite on the smartphone makes it possible to include a database in a small amount of memory inside the smartphone, which makes it the best choice if we consider the low memory available in a mobile terminal. In addition, we should consider that no more than a single user will access the smartphone database simultaneously, so the multiuser advantage
that other SQL systems provide is, in fact, not needed. As well as MySQL, SQLite is open source software. This one is already included in the Android SDK which is needed to develop Android applications. What is really included in this SDK is the library that has been previously explained [10]

### 3.2.2 Hibernate

SQL systems use their own language consisting of calling methods to the database. These methods have their own structure and characteristics depending on the type of SQL. Most of them are similar, but there are a few differences between them.

Few examples of MySQL calls:

```sql
INSERT INTO "table" ("column1", ["column2,..."] ) VALUES ("value1", ["value2,..."])
```

```sql
DELETE FROM "table" WHERE field2 = 'N'
```

```sql
SELECT * FROM "table"
```

The syntax of these methods is heavy, not very efficient, and very strict in the punctuation. It doesn’t fit at all in a Java code, it’s like an outside language included in a common one. The same thing happens when we talk about HTML inside a Servlet. So, is it good to include the MySQL code in an entire Java language program? It’s not impossible, not bad at all, but it is strange and, of course, inefficient. One solution comes using Hibernate [13].

Some of these methods change in a few parameters, characteristics or type of language and syntax. Nowadays, MADep is prepared to work using MySQL on the system at the hospital but in the future we are not sure if it will work under these conditions. What happens then if something changes inside the MADep structure? Would it be necessary to change all the code lines that affect the database storage? This problem is solved using Hibernate too [13].

### So, what is Hibernate?

Hibernate is an open source distribution of an Object-relational mapping library for Java. It provides a framework to work with Java classes and Object-oriented domain instead of database language. What Hibernate really does is map an Object-oriented model to a relational database (SQL in our case) [13].

Hibernate provides us classes with variables instead of tables and columns, and also provides query methods as well as insert, delete and update methods. It seems like a Java interface which works with databases in an easy and comfortable way. What Hibernate really solves too, is the problem of which SQL system is to be used. As will be explained in detail in the next section, Hibernate can change the SQL system just adapting the "translation layer", being very adaptable to changes in the SQL database.

### How does hibernate work?

As shown in the last section, Hibernate consists of an Object-relational mapping library for Java. In this library we can find different kinds of classes, methods, configuration and mapping files and Hibernate tools that make all this system work as an efficient interface between Java classes and databases. Hibernate basically contains three essential files needed in a project which includes Hibernate:
• Hibernate class

Hibernate classes contain all the information of a table mapped on the database. The name of the class is usually the database table name. Inside this class, every column contained in the table has its own variable and its own methods for each one. Getter and setter methods are written for each one of the columns/variables in the Hibernate class, inserting or removing the value of these. It is common to include as many Hibernate classes as tables in the database.

• Mapping file

As its name suggests, this file maps every one of the variables listed in the Hibernate class to its corresponding column in the database, including the table name. This file, written in XML language, makes a description of the characteristics of the columns, such as the type of variable (int, timestamp, ...). Each Hibernate class has its own entry in the mapping file.

• Configuration file

The configuration class consists of a number of lines written in XML, like the mapping file, which shows the type of database that will be used and the drivers needed for it. The configuration file includes the information of where the database is placed (URL), and the user and the password that will be used to access it.

Figure 3.2: Image showing the structure of Hibernate, its 3 main files and their interactions

Apart from these 3 main files, Hibernate uses methods and support files. For example, when Hibernate is going to be used, a class could be called on to create the Hibernate environment condition through a support class usually called HibernateUtil.class, which creates a Hibernate session to work on a certain database. From then on, we just include the data on the table class and then we use Hibernate tools to commit the whole class into the database. This method of
work will be explained in section 5.3.4. Figure 3.2 shows the 3 essential files of Hibernate and their interaction.

3.3 Web Services

A Web Service is a method of communication between two electronic devices over the Web [16] [18]. Web Services use protocols and standards to exchange data between applications. Different software applications developed by different parts using different languages are able to communicate and exchange data through the network due to the open standards used in Web Services.

As they work using standards and they are just in charge of data exchange on the network, Web Services are independent from the final application; they are like a lower layer which is not related at all to the tasks done over the rest of the application. This permits total independence between the application development and the use of Web Services. This is very important nowadays, where the development of a complex application comes from little independent developed parts.

Some standards such as HTTPS are part of Web Services most commonly used protocols. HTTPS will be used in MADep in order to transfer data from the application to the system at the hospital and from the server to the medical interface. A server running environment called Tomcat, which permits the Servlets running, is in charge of these HTTPS connections. Inside the HTTPS methods, many technologies will be used in order to manage data inside the Servlet. JSON will help us to serialize and deserialize data contained in the HTTPS message, and to organize this data into a mapping Object.

3.3.1 Tomcat

Tomcat is a server running environment that provides tools to manage connections, dependencies and libraries needed to run a server. Developed by Apache [17] is one of the most common running environments used today containing Servlets using an open source distribution. Using Java language, Tomcat can contain Java Servlets as well as JSPs, working as a web server.

Tomcat version 6.0.35 is the chosen to work as the MADep server container. As it is a standard Servlet container and a server running environment, Tomcat just provides, in our case, an external connection presenting HTTPS requests access, supported by any Web Service.

The compatibility between applications, servers and methods is essential when we work using a distributed system. MADep must use Android tools and protocols, and should use standards to work with the server, the databases and the final medical interface. So, supposing that everything will use these standard protocols, Web Services and Tomcat provide the developers with an essential freedom and independence from other parts involved in the system development. Everyone must know the standards and the key-value pairs, but the rest of the code written behind the connection established is not important to the rest of the developers, just the protocols used to connect both subsystems.

Tomcat as a server running environment is organized in a known structure containing configuration files, libraries, start and stop configuration files, resources used and, of course, the
Servlets and the JSPs which provide the core of the server tasks. To get an idea of what Tomcat does, figure 3.3 shows us the structure which is used in Tomcat.

Figure 3.3: Tomcat architecture and files on every directory

Using Tomcat configuration files we decide which port will be used for each purpose, which class will be called on as the initial index file shown to the user. We can restrict the user access to this server, asking for a password to use these Servlets.

On the bin directory we manage the starting and stopping Tomcat files, scripts in charge of the Tomcat running environment life. As in conf directory all the logs are saved in order to provide the developer with an idea of what is happening in the server.

The most important part of this structure is the web-apps file, where all the developers’ codes will be placed to take part in this architecture. Here is where the server keeps all the codes that gives it its identity, all the Servlets, the JSPs, classes, user libraries and application configuration files are inside this file, zipped in a Tomcat .war file, imported from the developer’s environment.

Servlets are contained in tomcat, so the developer should provide the system with the path to follow when a method of this Servlet is requested by an external client. Tomcat will execute this Servlet in the JRE of the physical server computer, and then provide this Servlet the tools needed to execute its tasks. When a response is generated, Tomcat will be responsible for sending this response to the client from the Servlet, using its own tools and the user’s ones together.
All these actions and tools used by Tomcat to generate this flowing along the Servlets life is known as the server running environment, permitting developers to create the server code and to insert it as a single .war file into Tomcat, and to avoid server configurations and tools.

### 3.3.2 HTTPS

HTTPS is the secure brother of HTTP (Hypertext Transfer Protocol), the protocol used in the World Wide Web (www) transactions. This is one of the standards used by Web Services to communicate applications working all over the Internet network.

It is a transaction-oriented protocol and uses the request-response schema, used by system at the patient’s home and system at the hospital parts. The client requests something from the server, and the server provides the client with a response according to their request. The client needs a URL (uniform resource locator) to send the request to the server, making the direction to this server unique to the server request method.

The HTTP transactions has an standard structure:

- **Client request:** The message coming from the client, responsible for the request, sent to the server, is structured as a heading followed by a blank line and, sometimes, some data. The heading specifies things like the action required of the server, or the type of data returned, or the status code. This heading provides flexibility to the protocol, so these fields allow descriptive information to be sent in the transaction, enabling authentication, encryption and user identification.

- **Server response:** The server sends back a response message saying if the request was correct or not, including a detailed error message if the communication has failed for any reason. The message also contains the data that has been requested by the client, as well as information about the returned Object.

In figure 3.4 we can see the request-response protocol taking place on a server using HTTPS methods. Every call needs its time to process request information and to generate an associated response and send it to the client, depending on the request characteristics. But, apart from the process time involved in the action, the protocol is an open standard, known by everybody and used in the Internet data exchange as a Web Service protocol.

Inserting security in this protocol involves HTTPS standards. The methods are the same, but a certificate is needed to work with on these secure methods. Usually connected to other ports which are different from the HTTP ones, HTTPS provides the application privacy and security on the connection.

As will be detailed on sections 4.6 and 5.3.1, a HTTPS certificate must be created in the system at the hospital to provide secure certification of the clients using this server HTTPS methods. To obtain this certificate, the developer must generate a key and then generate the Certificate Signing Request (CSR). This consists of a file containing characteristics from the server, ready to be signed to obtain the Signed Certificate. This final Certificate will be shown to the client to be accepted, to assure the requesting client that this connection is secure.

Anyone can sign his own certificate which means however, that not all signatures are secure. Nobody can ensure that a connection is secure if that connection is self-signed, only if the owner
Figure 3.4: An example of a HTTPS server listening to client requests and sending responses of the server is trusted by the client, which is what happens in MADep. In MADep system, the client is the MADep application developed by the same team as the system at the hospital has been set, who signed the self-signed certificate. In this case, the same team develops both parts of the communication, so HTTPS self-signed communication ensures security and privacy in front of external users.

For other purposes, Certifying Entities exist to provide accepted Signed Certificates to public servers. These certificates are recognized as secure, as the Certifying Entity is previously recognized as being reliable and secure by the users. The work of these Certifying Entities is just to sign the CSRs, providing the server with a connection which is supposed to be reliable.

3.3.3 JSON

JSON is a key-value pair mapping technology, which permits the developer to create structure in order to organize an amount of data. Instead of having several disorganized parameters, JSON provides tools to generate a mapped Object to include all this information into a simple Object. This JSON Object consists of a continuous key-value pair relation, included into a file which can be serialized and deserialized to send it through a data transfer protocol, as will be done in the MADep data exchange protocol.

In MADep we use databases with an amount of data that could be sent through the network value by value, but this is neither secure nor efficient. Instead of that we use JSON. When we want to zip data into a JSON Object we use JSON libraries which provides us with useful methods to insert data, just calling a simple method and passing the variable we want to insert.
into this Object. What we have to do is to generate a single key identifying this value and call the put() method to insert this just generated key-value pair into the previously created or opened JSON Object.

When the JSON Object is already created from the simple parameters, these can be extracted later. To extract a parameter from the JSON Object we need to know the key related to the parameter we want to extract. This implies that the developer has to know which key is assigned to each parameter. As we did in the put() method, we just call the get() method to extract the parameter, but now we just send the identifying key, and we obtain its paired value as a return.

JSON libraries can provide a method to serialize JSON Objects. In MADep we need to send this JSON Object from the system at the patient’s home to the system at the hospital, as will be explained in the following chapters. To do this we will use a single String to include it in the HTTPS package, so these libraries will help us to serialize the JSON Object to a JSON String.

All these tasks will be explained in detail in sections 4.5 in the System at the patient’s home chapter and in section 5.3.2 in the System at the hospital chapter.
4 System at the patient’s home

MADep is composed of two main parts, as introduced in section 2.2, System at the patient’s home and System at the hospital. This chapter includes every section on the system at the patient’s home, explaining in detail every one of them. A use case, the internal structure of the system at the patient’s home and the technologies used to construct are some of the aspects that will be detailed in the following sections. The system at the hospital will be studied in the next chapter 5.

First of all, a use case will be presented as a MADep environment, extracting the main targets that need to be covered by the MADep application, as well as the requirements needed for this application to work.

After the use case, a main structure of the system at the patient’s home will be detailed in 4.2.1 to give an overall idea of what this system really does offer. In section 4.2.2 the requirements of the smartphone needed to work with MADep will be studied, and finally a Flowchart of MADep application is detailed and explained in section 4.2.3.

The Android operating and the configuration of this application will be developed in section 4.3, as well as the lifecycle of an Android application, explaining the relationship between Activities, Services and Intents, as well as the work done in the foreground and the background, an important issue for this project.

Finally, the work done by different technologies in the background of the application will be explained in detail. The creation and interaction of SQLite Databases will show how the system stores data and how can this be shared with other actives, Services or even applications. JSON Objects are used in the encapsulation of data when this will be sent throughout the network, making it possible to include all the data of a patient to be sent in a single block of information, instead of seeing parameters one by one.

HTTPS protocol is the data exchange protocol used to send MADep data using 3G or Wi-Fi, it is important to ensure that a listener will be waiting for our message, and that this one will be understood. These last ideas will be detailed in sections 4.4, 4.5 and 4.6.
4.1 Use case

MADep is a monitoring system that allows a medical team treating a patient from depression to know the behaviour of all the patients in their own house. This system is composed of two main subsystems, the system at the patient’s house and the system at the hospital. These two parts are used in different situations and by two kinds of people, the patients and the doctors. Each group uses the system in a very different way, and both patients and doctors have different objectives and types of technology available to them.

4.1.1 Description of the use case

A depressive patient usually spends most of his time in his own home, sleeping, watching TV or just lying around without any intention of doing anything. Many times, even if the patient doesn’t show this attitude, he does not usually go outside for a walk. Most depressive patients don’t go to work, so they are all day wandering around their homes. A common depressive patient lifestyle is one that includes full day behaviour characterized by actions carried out in his own house, from the bedroom to the kitchen, including the bathroom and the living room. So, we are talking about a whole house space, but limited to finite square meters.

A depressive patient usually sleeps most of his time in his own home, watching TV or just lying around without any intention of doing anything. Many times, even if the patient doesn’t show this attitude, he does not usually go outside for a walk. Most depressive patients don’t go to work, so they are all day wandering around their homes.

A common depressive patient lifestyle is one that includes full day behaviour characterized by actions carried out in his own house, from the bedroom to the kitchen, including the bathroom and the living room. So, we are talking about a whole house space, but limited to finite square meters.

A MADep patient wakes up at 8:12 a.m. in his bed, he has been sleeping all night from 10:32 p.m. until now, so that means that the patient has been lying on the bed for about 9 hours and 40 minutes, this information is collected using a sensor placed in the bed. Most of the time, the patient has just been thinking and lying down, and for many of these hours he has really been sleeping. When this patient wakes up, his human reaction could be to go to the bathroom in order to take a shower, then to weigh and dress himself, when the patient weighs himself the information is sent to the smartphone using Bluetooth. Once he is dressed, the patient should go to the kitchen to eat something, it’s 9:00 a.m. and he has not eaten anything since 3:00 p.m. the say before, he is hungry. He spends 5 minutes because he only drinks a cup of coffee. After this lightning breakfast, he goes to the living room to turn on the TV and spends several hours watching 2 films and 3 episodes of an unknown TV series. This activities and movements around the house are collected by the smartphone itself and will be used to study the patient’s activity. It’s 2:26 p.m. on the smartphone clock, let’s fill the MADep test before going to eat something, he is not really hungry. "The doctor says to spend a few minutes on this test, but it tends to be a little boring. Oh! Today it looks different; it seems like it knows how I’m feeling now. It is interesting, let’s answer these questions: how many hours have you really slept today? Just 5 hours. Have you woken up before 11:00 a.m.? Yes, of course! ....", the data inserted in these tests will help the medical team to study the sensors data, and will provide an idea of the patient’s mood and lifestyle, together with sensors. Once the patient finishes the
test, he goes to the kitchen to eat something. He spends 2 hours in the kitchen preparing lunch and eating, he is not as hungry as he thought he was before so, he just plays with the food and thinks a lot, again it will be reflected on the smartphone sensors as the patient’s activity. After this situation, he feels tired, so he decides to go for a sleep, after all he has slept just 5 hours the previous night, so he spends about 4 hours lying on the bed, until 9:00 p.m., collected again by sensors in the bed. Now he is not really hungry to eat anything so he decides not to access the kitchen but to sit on the sofa again, it’s time for his favourite TV program, until 10:00 p.m., this will be again reflected as spent time on the sofa. Today they are playing "Seven" on the TV, he will lie on the sofa to watch it, but he falls asleep, until he wakes up and goes to the bedroom again, the sensors in the bed will tell us when does patient lies down on the bed tonight. The sun will rise again tomorrow.

4.1.2 Targets and Requirements

MADep should be able to adapt the system to the patient’s lifestyle. The doctor in the hospital should know the patient’s every action, and the evolution of the patient’s behaviour must be expressed in comprehensible graphics and phrases. The MADep subsystem located in the patient’s house must be able to achieve these main targets:

• Alimentary habits: Doctors need to know if patients feed themselves following a correct diet, schedules, and good alimentary habits. A patient’s feeding habits must be studied and monitored, and their weight must be reflected on an evolving graph.

To manage this situation many sensors will be placed on many objects in the patient’s house. Beginning with the fridge, a sensor captures when the patient opens the fridge to obtain food, how many times it happens during the day and when it happens, comparing it to the recommended feeding habits. Apart from the fridge sensor, which shows the alimentary habits of the patient, another sensor will help the doctors to evaluate the feeding situation of the patient. Another sensor will be placed on the weighing scales. Patients must weight themselves every day in order to obtain a graphic showing the evolution of the health of the patient.

• Movement and activity: The activity of the patient during the day is essential to identify the patient’s lifestyle. Doctors want to know the amount of hours the patient lies on the bed, on the sofa, and if this patient walks around the house. The movement of the patient all through the day is useful to determine if the patient has an active behaviour or if he just walks from the sofa to the bathroom, and from the bedroom to the kitchen.

To capture this data, two sensors will be used in the MADep system. A sensor placed on the bed will tell the doctors the amount of hours that the patient lies on it. An evolution graphic could be developed from this information, to show the doctors the sleeping habits in a comprehensible interface. Another essential sensor will be used for this target, the smartphone itself will be a movement sensor providing information about the patient’s movement throughout the day, this will tell us how much time the patient spends in every room, and what his most common behaviour is.
• Mood and lifestyle study: The patient talks to the doctor from time to time, but it is good to know how the patient feels more frequently. The patient can provide us with some extra information about his lifestyle through an automatic interaction. This interaction should be attractive to the patient in order to obtain real information in an easy and comfortable way.

MADep will provide a Virtual Assistant (VA), the most similar to a doctor’s behaviour, which will interact with the patient. The Virtual Assistant will raise a few questions to the patient in order to obtain this information from him. The interaction between the patient and the VA should be attractive to increase patient’s willingness to answer the Virtual Assistant questions.

• Flexibility and Independence: No extra effort should be requested of the patient to interact with MADep, apart from the Virtual Assistant. Patients won’t be asked to enter their sleep hours, their alimentary habits or their daily weight measurement. They should be free from any extra medical monitoring.

MADep will be implemented as a wireless monitoring system working as an automatic system, allowing the patient to behave in a normal way, without interaction with the system, except the VA. This will provide the doctor real lifestyle information collected periodically and sent to the system at the hospital in an automatic action. We need wireless systems in order to collect this data without interaction between the patient and the sensors.

• Data transfer: All this data obtained by MADep patient’s system should be available to doctors, so data must be transferred from the smartphone to the hospital. The patient, because of the MADep target involving flexibility and independence, cannot do this action. So, MADep itself must do this data transfer.

We need an extra system to be in charge of transferring this data from the smartphone to the system at the hospital in an automatic way. This part of the system will provide the patient independence from the data transfer action, and will show the doctors all the data coming from all their patients, concentrated in a single central database, using an easy and comfortable interface.

4.2 System at the patient’s home Architecture

MADep’s application for Android include, as was introduced in the Scenario chapter 2, consists of a few smaller structures developed by different project partners, and integrated into a single covering system, and zipped in an Android application. This master thesis involves the data exchange development of the complete application. As this part works in the background most of the time, an interface has been developed. This includes a few Activities showing the patient an idea of what MADep should be in a final version. In the following section, a structure of the application that was developed is presented, as well as a flowchart showing the behaviour linked to the patient’s interaction.
4.2.1 Main structure

Figure 4.1 shows the modular architecture of the Android application for MADep. This application contains two parts. (1) User interface involving the foreground work built in Activities. (2) Background work and actions, including most of the data exchange code, built as Android Services.

![Figure 4.1: Modular architecture of the MADep application developed](image)

What figure 4.1 shows is the complete modular structure separated in its main Activities and Services. In the foreground work area, Activities show the user the interaction permitted in MADep. When the user starts the application, he is welcomed in the first Activity, and then asked to insert his name or patient ID.

After this welcome Activity, the patient goes to the main menu Activity, being able to choose between different data types. Choosing between data types, the user is taken to another Activity, presenting a list of the last data collected. In this screen, the user can insert his own data by hand, or wait until data is received from the external sensors. This work will be done through the application’s life, at every point of it. But this is a simpler application, not including the final architecture of MADep. The electronic team should complete this part of the project in order to ensure that sensors send data using Bluetooth and that the application receives it and inserts it to the database. The reception of data is done in the AsyncTask background method, which is listening to an external data sending. This method refreshes the foreground activity in order to show the last data received.

On the other hand, if the user chooses the Virtual Assistant Activity, a medical test is launched in order to contact the patient and identify their mood and lifestyle. As explained
in section 2.3.2 in the Scenario chapter, the Virtual Assistant will be developed in detail by both medical and software development teams working together in order to obtain the optimal Virtual Assistant work. The VA included in this simpler application is just an idea of what it should be in the final MADep application. When the patient finishes the test, a Service (working in background) is called. This Service includes most of the data exchange protocol, including queries to SQLite database, encapsulation into a JSONObject, and data transfer through the network.

An important part of this application consists of permitting background work while the foreground one is presented to the user in a transparent way. While the user is interacting with the application, he shouldn’t know what is happening in the background. So, when information is being sent, the user can use the smartphone in a normal way.

4.2.2 Smartphone requirements

Smartphones working with Android OS are usually prepared for any kind of application requirements. MADep is not different from other applications, it doesn’t really need any special technology or version of the system. Our smartphone must fulfil the following requirements:

- Android version 4.0 or later: Previous Android’s versions didn’t include Bluetooth 4.0 [10], Bluetooth Health Profile is only included in Bluetooth 4.0 [14], so it is necessary in MADep data exchange between sensors and the smartphone. So in the final prototype of MADep the application should be developed in Android version 4.0 or later to ensure medical data privacy.

- Internet Connection: MADep application needs Wi-Fi at least, 3G as a second option. Assuming that the main work will be done inside the patient’s home, Wi-Fi will cover the Internet connection requirements that MADep needs to be able to work. 3G is recommended because of a possible failure of the patient’s home Internet connection. In this case, the smartphone will use 3G to send the data, instead of Wi-Fi.

- Memory space of 260KB: MADep application included in the smartphone will initially need an amount of 260KB free space on the phone memory. If MADep is improved and expanded to include other project partners’ work, more free space will be needed. Another aspect to take into consideration on the subject of memory, is the SQLite DB memory space. The DB could be included and mapped into an SD card, this in only an issue that should be decided when the final MADep application would be developed. This extra requirement must be specified to the user if the final version of the application includes the need for an SD card.

- Wireless chargers: The medical team is planning to provide the user with a smartphone charged using wireless technology, like induction chargers. It is known that some depressive patients use cables in a dangerous way, so doctors want to avoid the existence of cables in the patient house in order to prevent unexpected accidents.

All of these requirements and technologies needed for MADep application must be explained to the user before they install the application on the smartphone. Android requires that the
developers show the user the technologies that will be used for this application, and the user is asked to accept the terms and conditions.
4.2.3 MADep Application’s Diagram

MADep works on two execution levels, the foreground (user interface) and the background level. As it is possible to appreciate in the flowchart, when the Virtual Assistant Activity is finished it doesn’t just come back to the previous Activity but it initiates a background thread in order to do a background work while the user keeps using the smartphone.

Figure 4.2: Flowchart of the MADep application behaviour
4.2.4 Class Diagram

On figure 4.3 a class diagram from MADep is shown, detailing which classes are created in order to develop the MADep application running in the patient’s smartphone. It’s detailed which classes contain Activities responsible of the user interface, the Service one containing the background tasks and DataDbAdapter class in charge of helping the developers to access the database.

![Class Diagram](image.png)

Figure 4.3: Class Diagram of the MADep application

4.3 Android application

Android’s system, as explained in section 3.1 is formed by Activities, Services, Content Providers, intents, manifest file, ... The interaction between them is what really gives the application its final shape. The architecture performed inside the application is a previous study of what we want the application to do, as detailed in sections 4.1.1 and 4.2.4. When we put Activities, Services, intents, AsyncTasks and the manifest file together, we are creating the activity flow, as explained on section 4.2.3.

4.3.1 Android configuration and operation

If the requested requirements are covered by the smartphone, no extra configuration is needed to install an application, we just need the installation file. What the developer really does is to create a self-contained application, with all the resources needed, using the Android provided tools (on Android SDK), and to export it as a single .apk file. The .apk file will be installed
on the smartphone and, if the user accepts terms and conditions. The application should work without any extra configuration.

The doctor carries out the first step on the MADep application. When a new patient is included in the MADep treatment, this will be registered in the database. When we initiate the MADep application for the first time, we will be asked to introduce the new patient information. The doctor is responsible for filling these fields out in the right way. This will be the MADep medical sheet of this patient, including the MADep patient ID as well as the patient’s personal information. After this action, MADep is prepared to work in an automatic way.

4.3.2 Lifecycle

The flowchart shown above, in section 4.2.3, gives an idea of the previously fixed connection between components, classes and intents in the MADep application. The idea of traveling from one Activity to another, pausing the previous one and then coming back, or even starting a Service and keeping on working in the foreground while this Service is doing its work in the background, is called the lifecycle.

Every application and every component has a lifecycle, conforming to its lifetime from its birth until it is destroyed. What it really means is that an Activity is just a Java class waiting to be called by any external component, such as another Activity, a Service, an AsyncTask or other options. When an Activity is called, it begins its lifecycle, it is created and then begins its work. This Activity can then start another Activity (starting its lifecycle), so the parent Activity will enter in the pause state, until it is called back. The lifecycle of an Activity can be seen in figure 4.5, detailing the difference between the main application’s life, and the inner Activity lifecycle.

This brings us to the existence of a main user interface thread, where the user sees the lifecycle of this main thread. What is really happening is that components flow from one to another and are kept waiting to be called back. Sometimes this main thread creates secondary threads, responsible for background work. Here is where Services and AsyncTasks appear. Services and AsyncTasks have a lifecycle that is similar to the Activities lifecycle, but with a few differences. When the Service finishes its work, it usually disappears, it is self-killed. Similar behaviour happens with the AsyncTasks, but being included as part of a bigger component (Activity or Service), and are controlled by their father component’s lifecycle. In figure 4.4 we can see many screenshots of the MADep main thread.

Activities vs Services

Now we know that a main thread is responsible for the user interface work, and follows the main application’s lifecycle. But, the idea of a secondary thread which is responsible for the background work has already been shown. What really happens in the background isn’t the existence of a single thread doing all the work. For each different task, we often create a single thread. When the task is finished, this thread tends to be self-killed. Before this happens, it usually sends back some important information to the main thread.

Figure 4.5 shows the lifecycle of a common Service, it is important to notice that some Service return to the main thread, or send data back to it. But some others are just called and
do their work in an independent way from the main thread. When they finish, they normally die.

This secondary thread, usually implemented as Services, is responsible for the long-term actions. When the system has to send or receive data, such as downloading or package sending (our case), it tends to spend few seconds on it, setting the connection, transferring the amount of data and closing the connection. These actions shouldn’t disturb the user’s interaction with the smartphones, so these kinds of actions must be included at the background level. Here is when the developer should include the secondary thread using Services. In MADep there’s a main Service responsible for the whole connection and data sending, invisible to the user, but promoting the automatic essence of the MADep system. This thread will be explained in detail in section 4.3.3, in figure 4.5 there’s a graphical idea of the secondary thread.

**Intents**

The user decides the flow driving of the application, it is not pre-set. Not every Activity is able to call whichever other, there’s a pre-set structure that provides the possible connections in the system, but it is the user who really decides in every case, depending on the options.

Intents provide the user with the impression of feeling free to decide where he wants to go inside the application. What the user is really given is some choices presented, usually in buttons. The step made from one Activity to another is known as Intent. Intents provide us with the path decided by the user, choosing between the options provided by the developer.

In MADep application just a few Intents are used. Six main Activities (Welcome, New Patient, Data Type List, Data List, Manual Data insert and Virtual Assistant) are pre-set, and can be accessed using internal Intents, which can initiate another Activity or return to the parent Activity. Another extra Intent is included at the end of the work done by the Virtual Assistant Activity. This Intent is responsible for calling the main MADep Service. This Intent just starts this Service and then calls the previous Activity (Data Type List Activity). While this happens and the user keeps on interacting with the smartphone, the main Service works to send data to the system at the hospital.
What really happens is that an Intent starting the Service also initiates a new thread for this Service, the background thread mentioned on 4.3.2. It is easier to appreciate in figure 4.5.

![Figure 4.5: Detailed lifecycle of standard Activity and Service, and foreground and background threads](image)

### AsyncTasks

Similar to Services there are AsyncTask, defined threads created inside Activities in order to control background work, like Services do.

In contrast to Services, AsyncTasks are not called using an Intent but using the execute() method. A new thread will be created, as also happens when a Service is called. The main thread keeps on its own lifecycle while the background task does its work.

AsyncTasks are though to be little Services in order to execute simple background actions, which will take a long time. They tend to be simple in code extension and complexity, they aren’t usually able to create new threads, new life cycles or to do complex actions. The usual work done by an AsyncTask is to receive an amount of data and store it in memory (such as
a download), or wait until something pre-set happens and then call a method into the father Activity.

It is usual to control the AsyncTask life from the Activity who calls it. In the onStart() method, the AsyncTask could be initiated, and kept alive until it finishes its work, depending on the type of task assigned to it. If the task consists of waiting for an external signal, or similar task, maybe we don’t need this AsyncTask to work outside the Activity lifecycle. In this case, we are able to shutdown the AsyncTask from inside the Activity, an option which could be in the onStop() method.

While the DataList Activity is running, a secondary thread implemented with an AsyncTask will be waiting until external data is received from the supposed weighting scale. When this data has already been received, these will be notified from the AsyncTask to its father Activity, the DataList, and updated. In this case, the AsyncTask responsible for this task will be initiated in the onResume() method, and then stopped on the onPause() method. While the Activity is running, the AsyncTask will do it too, updating data every time it receives it. It is important to remember that this won’t be included on the final MADep system but will be replaced with Bluetooth technology and external electronic sensors.

4.3.3 MADep main Service

A single Service carries out most of the background work in the MADep application. When the user has travelled along the application, there is a point when the information is sent, this will actually happen periodically. When the user has already completed the Virtual Assistant test (it is supposed to happen every 15-30 days, depending on the medical team’s final decision), data will be sent using an essential component, the main Service. This task could take a few seconds or so, in order not to disturb the user, it has been decided to include it as a background work, initiating a secondary thread.

In contrast to AsyncTasks, the Services have more complex structures, could include more tasks, and are accessed by external components. They are individual classes, separated from other Activities or Services, and access is visible to anyone who wants to use them. AsyncTasks tend to be called as internal methods included in Activities, as explained previously in section 4.3.2.

The main Service of MADep is as the principal motor of data exchange in this system. All the work done previously by the rest of the application, from the data collected by sensors to the Virtual Assistant, passing along data management and storage, is recovered, packaged and transmitted at this point. What the Service really does is collect data from the SQLite database, encapsulate all of them into a single JSON Object, and initiate a HTTPS connection in order to send the data through the network to the target point, the hospital server.

As it is the main body of the MADep application data exchange system, it is important to separate it into three essential parts: (1) SQLite querying, (2) JSON mapping and (3) HTTPS connection, as shown in figure 4.6. The SQLite block will be responsible for querying data, and translating it from the SQLite database to Java private variables. When the data is contained in the Java class, it’s time to encapsulate them into a single Object, this is done in the JSON block. Last, but not least, this single Object will be included as a parameter into the HTTPS request, responsible for the data sending to a known pre-set target.
4.4 SQLite in Android

SQLite is the SQL language used by Android to deal with databases. On a smartphone, we will only store the user data in this application. It won’t be a problem to include MADep data in a database based on SQLite. We should remember that SQLite databases are the only ones available in Android, and are enough for the database sizes used in the MADep patient subsystem.

This is the tool that Android’s SDK gives to developers but, in fact, it is the best choice. All the developers working in MADep will use SQLite, permitting other project partners to manage the data they include in the common database. So, it is essential to know how the database will be in order to access it from one point of the system, to query data, while from another point this data is inserted into the database by another Activity or Service, developed by other project partners.

As it will be explained in the next section, a helper class will be accessible for developers to work with the database so, instead of typing all the code, they will decide things for themselves or manage the database at a lower level, the developers should only have to call on a few methods in order to insert, query or update the database, or even create a new one. This provides a level structure to access the database, as shown in figure 4.7.

4.4.1 DataDBAdapter class

We will detail the difficulty of using SQL language directly on sections 4.4.2 and 4.4.3, and the necessity of including a helper class to solve this problem. Here is where DataDBAdapter class appears. It is similar to the tasks done by Hibernate in the system at the hospital, we’ll see it in
detail in section 5.3.4.

This class just offers the system simple methods to insert, query, update and delete data from the database. Even the class creates the database for us, we just call the DataDBAdapter() constructor and the class itself identifies if the given name database exists or if it is necessary to create a new one, and the class does that too.

When the developer wants to insert new data, it is only necessary to call the insertData() with the data included as parameters. The same happens when we want to query data; a Cursor pointing to the different lines of a table gives the developer the option of travelling along the table querying different lines and columns in a quick and simple way.

This own class created to make developers’ work quicker, is useful because the created methods are accessed from inside the application. So methods included in these classes are specially designed for our particular database, they are useless to external systems. This is the difference between this own created class and SQLite helper class or Hibernate.

What the DataDBAdapter class really contains is not a simple method called insertData() but a few of them, each one related to the data type. So if we want to insert the patient’s weight in the database, we should call the insertWeight() method, including the weight as a parameter. The same happens, for example with the hours slept, or other data. These methods will include the parameters into a ContentValues Object, and then call the SQLiteHelper methods to insert them in the final database.

![Layers between MADep application and the database](image)

4.4.2 Creation of a Database

The creation of the database is one of the most important parts of the database section. This is due to the common nature of the final database throughout all the MADep system. It doesn’t
matter who develops the file that creates the database, but it is essential to use the same one all through the application, whenever the database is created. The best way to ensure this is by creating a method in a class which is responsible for creating the database. We can use the same class that helps developers to interact with the database, creating an inner method called on just to create the database.

As was presented on the MADEp flowchart, in figure 4.2, when it is the first time that we access the MADEp application, the system will check if the patient already has an initiated database on this smartphone or, in contrast, whether this is the first time the patient accesses this terminal, and then it will begin the database creation Activity. Here, the doctor will insert all the patient information that will be inserted in the database. What the system really does here is call a helper method responsible for deciding if this database has already been created. If the patient is recognized, the existing database will be opened and the system will begin working on it. In contrast, if the patient is a brand new user, the system won’t find the corresponding database so, a new patient database will be created. From now on, this patient will have their own database on the smartphone.

MADEp is thought to be a personal system controlled by the hospital. The main idea is to assign one non-transferable phone per patient, so, in fact, the idea of choosing between an existing or a new database is just for security and as an academic exercise. Just one database matching the patient is included in the smartphone.

The database creation method must be developed anyway. To create a new database, the SQLite helper will help us to avoid the final contact with the database but, assuming that this is the most detailed part, the developer has to create each table and column one by one, explaining what kind of parameter it will contain. The code to create a table in a database should be something similar to this:

```java
"CREATE TABLE " + PATIENT_TABLE_NAME + " (" +
PATIENT_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
PATIENT_MADEP_ID + " NUMERIC, " +
PATIENT_NAME + " TEXT, " +
PATIENT_SURNME_1 + " TEXT, " +
PATIENT_SURNME_2 + " TEXT, " +
PATIENT_ALIAS + " TEXT, " +
PATIENT_BIRTHDAY + " DATE, " +
PATIENT_MAIN_CAUSE + " TEXT, " +
PATIENT_has_COUPLE + " NUMERIC, " +
PATIENT_has_EMPLOYMENT + " NUMERIC, " +
PATIENT_SENT") ;
```

Every key identifier is detailed on each line, and the corresponding data type next to the identifier. All this SQL code will be saved in a Java final String, being called on later by the creator method and launched directly to the SQL command line. The developer should previously design each table and its content, divide the information in several tables and then types every table code, as seen above.

When a database is created, it is given a name (in MADEp’s case the patient name or ID), and then table creation methods are called on, one for each table that is required in the database.
This is only done the first time the patient accesses the database, but it is essential to ensure that this is done in an accurate way. All the MADep tables included in a patient profile can be seen on figure 4.8 where an Entity-Relation diagram shows all the MADep database structure.

<table>
<thead>
<tr>
<th>PATIENT</th>
<th>PHQ9_QUESTIONNAIRE</th>
<th>WEIGHTS</th>
<th>SLEPT_HOURS</th>
<th>TROUBLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id</td>
<td>_id</td>
<td>_id</td>
<td>_id</td>
<td>_id</td>
</tr>
<tr>
<td>id_madep_user</td>
<td>id_madep_user</td>
<td>id_madep_user</td>
<td>id_madep_user</td>
<td>id_madep_user</td>
</tr>
<tr>
<td>name</td>
<td>name</td>
<td>name</td>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>surname_1</td>
<td>surname_1</td>
<td>surname_1</td>
<td>surname_1</td>
<td>surname_1</td>
</tr>
<tr>
<td>surname_2</td>
<td>surname_2</td>
<td>surname_2</td>
<td>surname_2</td>
<td>surname_2</td>
</tr>
<tr>
<td>alias</td>
<td>alias</td>
<td>alias</td>
<td>alias</td>
<td>alias</td>
</tr>
<tr>
<td>birth_date</td>
<td>birth_date</td>
<td>birth_date</td>
<td>birth_date</td>
<td>birth_date</td>
</tr>
<tr>
<td>main_cause</td>
<td>main_cause</td>
<td>main_cause</td>
<td>main_cause</td>
<td>main_cause</td>
</tr>
<tr>
<td>has_couple</td>
<td>has_couple</td>
<td>has_couple</td>
<td>has_couple</td>
<td>has_couple</td>
</tr>
<tr>
<td>has_employment</td>
<td>has_employment</td>
<td>has_employment</td>
<td>has_employment</td>
<td>has_employment</td>
</tr>
<tr>
<td>patient_sent</td>
<td>patient_sent</td>
<td>patient_sent</td>
<td>patient_sent</td>
<td>patient_sent</td>
</tr>
<tr>
<td>phq9_answer_0</td>
<td>phq9_answer_0</td>
<td>phq9_answer_0</td>
<td>phq9_answer_0</td>
<td>phq9_answer_0</td>
</tr>
<tr>
<td>phq9_answer_1</td>
<td>phq9_answer_1</td>
<td>phq9_answer_1</td>
<td>phq9_answer_1</td>
<td>phq9_answer_1</td>
</tr>
<tr>
<td>phq9_answer_2</td>
<td>phq9_answer_2</td>
<td>phq9_answer_2</td>
<td>phq9_answer_2</td>
<td>phq9_answer_2</td>
</tr>
<tr>
<td>phq9_answer_3</td>
<td>phq9_answer_3</td>
<td>phq9_answer_3</td>
<td>phq9_answer_3</td>
<td>phq9_answer_3</td>
</tr>
<tr>
<td>phq9_answer_4</td>
<td>phq9_answer_4</td>
<td>phq9_answer_4</td>
<td>phq9_answer_4</td>
<td>phq9_answer_4</td>
</tr>
<tr>
<td>phq9_answer_5</td>
<td>phq9_answer_5</td>
<td>phq9_answer_5</td>
<td>phq9_answer_5</td>
<td>phq9_answer_5</td>
</tr>
<tr>
<td>phq9_answer_6</td>
<td>phq9_answer_6</td>
<td>phq9_answer_6</td>
<td>phq9_answer_6</td>
<td>phq9_answer_6</td>
</tr>
<tr>
<td>phq9_answer_7</td>
<td>phq9_answer_7</td>
<td>phq9_answer_7</td>
<td>phq9_answer_7</td>
<td>phq9_answer_7</td>
</tr>
<tr>
<td>phq9_answer_8</td>
<td>phq9_answer_8</td>
<td>phq9_answer_8</td>
<td>phq9_answer_8</td>
<td>phq9_answer_8</td>
</tr>
<tr>
<td>phq9_answer_9</td>
<td>phq9_answer_9</td>
<td>phq9_answer_9</td>
<td>phq9_answer_9</td>
<td>phq9_answer_9</td>
</tr>
<tr>
<td>phq9_total</td>
<td>phq9_total</td>
<td>phq9_total</td>
<td>phq9_total</td>
<td>phq9_total</td>
</tr>
<tr>
<td>phq9_date</td>
<td>phq9_date</td>
<td>phq9_date</td>
<td>phq9_date</td>
<td>phq9_date</td>
</tr>
<tr>
<td>phq9_sent</td>
<td>phq9_sent</td>
<td>phq9_sent</td>
<td>phq9_sent</td>
<td>phq9_sent</td>
</tr>
<tr>
<td>weight_date</td>
<td>weight_date</td>
<td>weight_date</td>
<td>weight_date</td>
<td>weight_date</td>
</tr>
<tr>
<td>weight_value</td>
<td>weight_value</td>
<td>weight_value</td>
<td>weight_value</td>
<td>weight_value</td>
</tr>
<tr>
<td>weight_sent</td>
<td>weight_sent</td>
<td>weight_sent</td>
<td>weight_sent</td>
<td>weight_sent</td>
</tr>
<tr>
<td>slept_hours_date</td>
<td>slept_hours_date</td>
<td>slept_hours_date</td>
<td>slept_hours_date</td>
<td>slept_hours_date</td>
</tr>
<tr>
<td>slept_hours_value</td>
<td>slept_hours_value</td>
<td>slept_hours_value</td>
<td>slept_hours_value</td>
<td>slept_hours_value</td>
</tr>
<tr>
<td>slept_hours_sent</td>
<td>slept_hours_sent</td>
<td>slept_hours_sent</td>
<td>slept_hours_sent</td>
<td>slept_hours_sent</td>
</tr>
<tr>
<td>trouble_type</td>
<td>trouble_type</td>
<td>trouble_type</td>
<td>trouble_type</td>
<td>trouble_type</td>
</tr>
<tr>
<td>trouble_date</td>
<td>trouble_date</td>
<td>trouble_date</td>
<td>trouble_date</td>
<td>trouble_date</td>
</tr>
<tr>
<td>trouble_duration</td>
<td>trouble_duration</td>
<td>trouble_duration</td>
<td>trouble_duration</td>
<td>trouble_duration</td>
</tr>
</tbody>
</table>

Figure 4.8: Entity-Relation Diagram of the SQLite database in the MADep application

4.4.3 Inserting and querying data

Inserting data into an SQLite database is simple as we are using an SQL declarative language. We should call the insert method and fill it with the corresponding identifier keys. Querying data is a little bit complicated. The developer must switch the line that they want to recover and then extract the data needed from this line. The manual method of doing this using direct SQL language forces the developer to know every line of every table structure exactly. This is not an easy and quick way to work, but otherwise, sometimes it could be the only way.

Inserting data in an SQLite DB

When the developer wants to insert a new line in a database or update an existing one it is as simple as writing the insert method from SQL language and then filling it in with the corresponding data. An example could be something like this code:

```
INSERT INTO PATIENT_TABLE_NAME
(PATIENT_ID,
PATIENT_NAME,
PATIENT_SURNAME_1,
PATIENT_SURNAME_2)
VALUES (ID_RECEIVED,
NAME_RECEIVED,
SURNAME_1_RECEIVED,
```
This has no secret to include a few parameters, the problem appears when we are trying to include a whole table. This will be solved using the SQLite helper. In the DataDbAdapter class we have already developed methods in charge of this task. Using methods like insertPatientData(), we only need to include the data that we want to insert into the database as parameters to the method, and this will do the work for us.

**Querying a SQLite DB**

To query an SQL database, the developer should know the whole database structure accurately, to ensure that the query is well positioned to the data that we want to recover. This could be a difficult process if we’re directly using SQL language, a simple query could be as complex as this one:

```sql
SELECT PATIENT_NAME, 
PATIENT_SURNAME_1, 
PATIENT_SURNAME_2, 
PATIENT_ALIAS 
FROM PATIENT_TABLE_NAME 
WHERE PATIENT_ID = ID_RECEIVED;
```

This only shows the data, but does not select any of it and convert it into a useful parameter. The complex syntaxes of these methods bring us to the use of a SQLite helper class, provided by Android SDK itself. This is so useful for developers, even essential. For even an easier and quicker developers’ task, the DataDbAdapter previously developed class helps the developers to interact with the database in a simple and quick way. To query data we use a Cursor that collects all the information of a single row in the database, and then we indicate which column we want to extract and convert to a useful parameter. We will call the query method in the DataDbAdapter class to obtain this Cursor, and then we will collect those columns we really need for our objective.

### 4.5 JSON

We’re inserting data into the database all through the month, everyday an amount of data come into the patient’s info on the smartphone but, what happens then? All the information should be sent in one single packet. The data recovered in 15-30 days must be zipped and then sent, and this must be done in each MADep’s patient’s smartphone. JSON technology, introduced in section 3.3.3, is capable of packing all this data into a single JSON Object.

#### 4.5.1 JSON structure

JSON technology is based on key-value pair mapping systems. JSON creates data structures in order to sort an amount of information into a known structure. This permits a system to transfer data from one place to another just knowing the kind of structure this data has, or not even that.
JSON structures, as introduced in section 3.3.3, consist of series of key-value pair relations, conforming a whole structure being accessible using a simple and quick method. Every key should be related to a single value, but sometimes it doesn’t happen.

In MADep, the JSON Object created for the data transmission is as simple as the database permits. It is important to notice that as an important amount of data will have already been sent in previous data transfers, just the new data should be sent. This brings us to an important point, the importance of creating a method to identify which data has already been sent.

Some ideas have come out, send data only entered from a date on, erase the sent data and just save the new coming, or create a flag to mark that which has already been sent and that which has not. This last idea is the one chosen by this master thesis, every datum will be associated with an extra flag, responsible for indicating if that datum has been sent or not. This is not an optimal solution but a simple and practical one. Maybe, an amount of data could be related to just one flag, like a whole day data collection, or a whole session of the Virtual Assistant.

"PATIENT_MADEP_ID" : "999999"
"PATIENT_DATE" : "21/12/2012"
"WEIGHTLIST":
{
  "WEIGHT_DAY_1": 78
  "WEIGHT_DAY_2": 79
  "WEIGHT_DAY_3": 79
  "WEIGHT_DAY_4": 77
  "WEIGHT_DAY_5": 76
  ...
},
"SLEPT_HOURS_LIST":
{
  "HOURS_DAY_1": 8
  "HOURS_DAY_2": 5
  "HOURS_DAY_3": 12
  "HOURS_DAY_4": 5
  "HOURS_DAY_5": 6
  ...
},
"PHQ9_QUESTIONNAIRE":
{
  "PHQ9_ID": 78
  "PHQ9_ANSWER_1": 4
  "PHQ9_ANSWER_2": 9
  "PHQ9_ANSWER_3": 3
  "PHQ9_ANSWER_4": 7
  "PHQ9_ANSWER_5": 5
  ...
},
The JSON structure will be created in the main MADep Service. When we have already recovered all the information that must be sent, using DataDBAdapter methods, all this data will be inserted into a JSON Object, similar to the one shown above. What we can see in this figure is the mapping of every single datum into the JSON structure. This structure is the one that will be sent using an HTTPS connection. The data will be sorted by tables, similar to the database structure, and inside them every parameter will be identified with a singular key. This map is small enough not to repeat any key-value pair.

4.5.2 Insert data into a JSON Object

The introduction of data into a JSON Object is as simple as knowing the key identifier and switching it to the corresponding value, previously extracted from the database. This action takes part in MADep’s main Service, which is responsible for data transfer. The first step of this Service is the data query from the database, as seen is section 4.4. When this data recovered from the database is in the Java class, referred as a variable, it should then be sent to the network. But an intermediate state appears at this point, the data must be structured is the previously explained JSON map.

Here is where the JSON language appears. JSON consists of several Java libraries containing useful methods that permit the developer to manage JSON data and structures in an easy and simple way. In fact some libraries are not needed indeed, but are so useful in many actions. We will use basically the actions of including and recovering data from JSON Objects and the serializing methods from JSON to String and the opposite one.

Inserting data into a JSON Object consists of referring every value to its key in the JSON mapping. So, the developer must create an empty JSON Object using the given libraries and then, fill this Object with the values. This is done one by one, we include a single value into the JSON Object and then we include the next one, so we create the mapping.

An insert method is already developed in the JSON libraries, so to insert a value we call on this method (named JSONObject.put("key", "value")), passing it the parameter to include and the given key. The method is in charge of the insertion of this parameter.

When all the data has been inserted into the JSON Object, we have the whole mapping of this Object, but we are unable to insert it as a JSON Object into an HTTPS request. The solution comes through another JSON library. There’s a method that permits the developer to serialize the JSON Object into a JSON String. This String has the same structure as a JSON mapping, as shown in figure 4.9, but this String is a serialized Object so, it is able to travel into an HTTPS connection as a normal parameter. So, after inserting all the data into the JSON Object, we have serialized this Object into a single String and we must include it into the HTTPS request. We will see how to do this in the following sections.
4.6 HTTPS

HTTPS, as introduced in section 3.3.2, is a HTTP secure Internet transfer protocol based on TCP/IP. This protocol permits MADep to exchange data between the smartphone and the system at the hospital. By now, it will be explained the data transfer from the smartphone to the system at the hospital, because it is a one-way action. It will be a response packet including the result of the data exchange but it will have no impact on the application database, it is only needed for the exchange checking.

Security is essential in MADep as we are managing private clinical data. Apart from signing this data not using the patient’s whole name, but a MADep ID, this should be protected again using another security system. Here is where HTTPS plays an essential role in the MADep system. The data sent throughout the MADep system should be protected from outside attacks. On the smartphone it is supposed that the security is guaranteed because of the inner work, MADep is a self-contained application, and uses internal databases to store patient’s data. When an external action involves MADep it must be certified by the user, remember the initial signing of some terms and conditions when MADep is installed on the smartphone. The external access to MADep will only be permitted in two cases:

- Bluetooth: When external data coming from the sensors comes into the MADep application, it is inserted into the internal database but no data is extracted from it so, using Bluetooth only permits external parts to insert data, not to query it. Anyway, as introduced in section 4.2.2, MADep uses a Medical Bluetooth technology that assures the privacy of the data exchanged so, this communication protection is already guaranteed.
by this Bluetooth special protocol.

- HTTPS: The time to send data to the system at the hospital is when data collected in MADep and stored in the internal database is sent to the external world. There’s a key point here where security must be ensured to a high level. First of all, data won’t be sent using the patient name but the MADep Identifier for this user. This ensures that individual data interception can’t be related to a particular patient. But this is not enough. HTTPS ensures the certification that data will travel to a known target, previously identified using HTTPS keys and certificates.

### 4.6.1 How to obtain a key?

In contrast to HTTP, HTTPS is a secure protocol that needs a signed certificate to guarantee secure communication. We will see how we create the certificate on the server in section 5.3.1, but now we should ensure that the smartphone accepts the secure connection to the corresponding server. When we have already certified the HTTPS connection on the system at the hospital and we have the certificate, we need to include the public certificate of the server into the Android application, as an external resource.

Using openssl tools, we will obtain the public certificate (PEM) of the server accessing it and storing this public certificate in a file.

```bash
echo | openssl s_client -connect seneca.upc.es:443 2>&1 | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > mycert.pem
```

What we use now is the Java key-tool preinstalled in the JRE. Importing the just obtained public certificate mycert.pem, we will now generate a .bks keystore containing this PEM file as a trusted certificate. We will use BouncyCastle tools to generate this .bks file.

```bash
keytool -import -v -trustcacerts -alias 0 -file <(openssl x509 -in mycert.pem) -keystore mystore.bks -storetype BKS -provider org.bouncycastle.jce.provider.BouncyCastleProvider -providerpath ./bcprov-jdk16-145.jar -storepass ************
```

This will generate a keystore containing just our trusted key, so MADep application doesn’t need extra certificates. This mystore.bks file should be included in our Android Project’s resources directory, including it as a resource, similar to including a new external image on the project. The next step consists of creating an SSLSocketFactory Object in order to include it into a SchemeRegistry, used in the main Service to include security into HTTP communication.

```java
try {
    final KeyStore ks = KeyStore.getInstance("BKS");

    //importing keystore from R.raw.mystore
    final InputStream in = myCtx.getResources().openRawResource( R.raw.mystore);
    openRawResource( R.raw.mystore);
```
try {
    ks.load(in, myCtx.getString(R.string.mystore_password).toCharArray());
} finally {
    in.close();
}

//Create a SSLSocket based on the trusted keystore
SSLSocketFactory httpsSocket =
    AdditionalKeyStoreSSLSocketFactory(ks);

} catch( Exception e ) {
    throw new RuntimeException(e);
}

When we have the SSLSocketFactory built using the keystore previously obtained from
the public certificate of the server, we will use this SSLSocketFactory to insert security into
our HTTP default connection. We will do this by inserting this SSLSocketFactory into a
SchemeRegistry, and then link it to a Connection Manager. When we create the Default HTTP
Client, we must import this Connection Manager containing the SSLSocketFactory.

final SchemeRegistry schemeRegistry = new SchemeRegistry();
schemeRegistry.register(new Scheme("http",
    PlainSocketFactory.getSocketFactory(), 8080));
schemeRegistry.register(new Scheme("https", httpsSocket, 443));

//Initiate Http parameters including the SSLSocketFactory
// for secure connections
final HttpParams httpparams = new BasicHttpParams();
final ThreadSafeClientConnManager cm =
    new ThreadSafeClientConnManager(httpparams,schemeRegistry);

//Create a default HTTP Client importing trusted certificates
DefaultHttpClient client = new DefaultHttpClient();
DefaultHttpClient httpClient =
    new DefaultHttpClient(cm, client.getParams());

//Create the response and execute the POST
HttpPost post = new HttpPost("https://seneca.upc.es:443/
    MADepServer/MADepServlet");

//Include params, an ArrayList containing patient data
UrlEncodedFormEntity ent =
    new UrlEncodedFormEntity(params, HTTP.UTF_8);
post.setEntity(ent);

//execute Post method on secure HTTP communication
httpClient.execute(post);

When we have included the trusted certificate into our default HTTP connection we initiate
the communication. The smartphone will compare its trust certificate included on the imported
keystore with the public certificate offered by the server. If these certificates match, the con-
nection is secure, and we can trust this server. Data is transferred then from the smartphone
to the system at the hospital, concluding the secure connection with a successful data transfer.
If certificates don’t match, the connection is not secure, so the smartphone won’t complete the
transfer, keeping the patient data on the safe local database.

4.6.2 HTTPS POST

The final action done by the MADep main Service is the sending of all the data inserted into the
single JSON Object to the target server using a secure connection. This part will be developed
using HTTPS protocol and including this data into a HTTPS POST method.

A POST method in HTTPS will be received by the server as a message containing data. So,
in the application we should include all the data that we want to send inside this kind of request
message. HTTPS only accepts serialized data, resulting on Strings or simple data format. This
is the reason for the JSON String methods, explained in section 4.5.2.

When all the data is mapped into a JSON Object and then serialized into a simple String, this
can be included easily into an HTTPS request. What we need to do is to create an ArrayList
containing NameValuePair Objects. On this list we associate the JSON String to an identifying
key, know by the system at the hospital. This allows us to include this String into the package
being accessed by its associated key. When we create the Entity, in charge of the HTTPS
parameters, we include this ArrayList which contain all the patient data. This data will be
consequently recovered by the system at the hospital, and then inserted into the final hospital
database.

Now we have a complete HTTPS request. Security has been ensured using the SSLSocket-
Factory, previously certified by the key-store file obtained from the server certificate. Data has
already been inserted as an HTTPS parameter, including all the data that we want to send in
this packet. What we need now is the URL where this data should be sent.

Using the known URL pointing to the MADep server, the application will be able to complete
the HTTPS connection. The port will be previously indicated in the SSLSocketFactory because
different ports will be used in HTTP and HTTPS connections.
With the final HTTPS request, shown in figure 4.10, the MADep’s home system will send all the patient’s newly collected data to the system at the hospital. This action will be done by each patient’s system, and the MADep system at the hospital will collect all the incoming information from every MADep home system. From now on, the server is in charge of the MADep complete system’s work. In the following chapter, the tasks of the system at the hospital will be explained in detail, from the reception of the HTTPS request to the final interface acceded by the medical team.
5 System at the hospital

Here is when MADep jumps to another kind of technology, structures and working methods. HTTPS request sent from the Android application brings us to the MADep System at the hospital. We change smartphone applications for server, Activities and Services for JSP and Servlets, SQLite for MySQL, and obviously, patients’ houses for the hospital.

The structure of this part of the system is different from the already explained system at the patient’s home. This is the reason why this project is structured in two essential subsystems. One of them developed in the patient’s house, working together with the electronic project partners to develop sensors and Bluetooth data exchange. The other part consists of the hospital environment system, containing the data of all the patients in the main database, servers providing information and access to the medical and, all of them, developed together with the software partners.

As was already introduced in section 2.2, this master thesis is responsible for the connection of these two parts. Providing both parts interaction, using access to databases, Android and server technologies and data exchange protocols, the system is able to make these two parts into a single one, generating a self-contained and autonomous system, MADep.

In this chapter will be presented many parts of the system at the hospital. Beginning with an explanation of what a server is, we will shortly see the differences between Databases (Model), JSP (View) and Servlets (Controller), as well as what POST and GET methods are.

After that, a main idea of what the structure of the server is, will be shown in section 5.2. First of all, we will observe the modular idea of the system at the hospital. Having a basic idea of the technologies that the system at the hospital must have, finally, we will see the flowchart of how the system at the hospital activity will flow.

Finally, both methods, POST and GET will be studied in detail. POST actions will be explained in section 5.3, beginning from the HTTPS certificate creation, recover data from a JSON String, and inserting them into a MySQL database using Hibernate. GET method will be explained briefly in section 5.4, presenting HTML language, and presenting a future development of the medical interface.

5.1 Servers

A server is a program running locally in a physical computer, in charge of handling the requests of other programs, the "clients". In our case, the clients are the MADep application and the medical team.

There are several kinds of servers: file server, database server, game server, print server, and so on. The types of server included in the MADep system are web servers. These kinds of servers receive HTTPS requests and generate responses to these calls. Inside the server, this could take a more complex structure and generate other types of messages and actions, even
different servers can be included in the same structure. In fact, MADep server works as a web server, but another server is used. It will be explained in section 5.3.3 that MADep web server will access an internal MySQL database server in order to store all the MADep data into a MYSQL database. In the next section, the server structure and the methods used will be explained in detail.

5.1.1 Server structure

A server must be connected to the external world, this provides the server with the capacity to receive external requests and send them a response. This brings us to the idea of the listening behaviour of a server. In order to listen to the client’s requests, a server must include a channel to be aware of external connections. The TCP/IP stack will provide this channel.

What a server really needs is a listening channel, provided by the URL assigned to the server. When a client wants to access a server, it must know where the information should be sent. If there’s nobody listening in the target direction, the client will receive an error, and the communication will finish right here. So, what we need for our server is to ensure that the URL assigned to our server is unique and that it is operative. The URL contains the public IP and the port assigned to this method. The physical computer will assign this port to the chosen code, and any external connection requesting this URL(IP+Port) will access using this code, and then the request will be attended.

A server can be as complex as the developer wants, and it can contain a user interface or not. The work done by MADep doesn’t need a user interface at all, what it really needs is a final interface shown to the medical team, but this could be placed apart from the MADep server requesting data from the MADep main server. As it is a part of the final MADep system, a simple user interface has been developed too. This server is structured as a model server, including the three main parts a server should have:

- **Model**: A database can be included in the server if it is needed. If there’s an amount of data contained in the server, it can be stored in a database. This database can be placed in the same physical computer or in an external one, accessing it using a data server. In the system at the hospital, a localhost MySQL database is used to store all the patients’ data. This part is transparent, its location and its technology (kind of database) can be changed without involving the other parts.

- **View**: A visual part of the server shows the user the data in a familiar and attractive way, making the interaction easier to the final user. This part tends to be developed by a graphic designer, and should be transparent to the rest of the server. In this project, this is included as part of a simple HTTPS GET method and the visual design is not attractive at all.

- **Controller**: The core of the server contains the actions and tasks. All the work done by the user is developed in the main code, containing the methods, the classes and the external communication with clients. This is really the server itself, the skeleton of the whole system. This part is transparent to the Model and View part, and can be improved depending on server requirements.
The MVC is a development paradigm that is used in most web servers, including MADep. Sometimes they are mixed, and their differences are not at all clear. For example, JSP tends to contain the View with few adaptive actions, and Servlets tend to do the main tasks of the Controller using the developed code, receiving requests and responding to them. In the next section we will see the differences between them, and how they interact inside the system at the hospital.

## 5.1.2 Model-View-Controller

Databases are the tools used to manage data on the servers, and to build the Model of the server. JSP can be used to create the user interface (View), Servlets are responsible for showing results and receiving external activity (Controller). All of them can do inner tasks and be the skeleton of the server, performing the code that provides responses to the requests, and accessing data. So, what is the difference between them?

### Model

Databases work on background, they are structured in tables and contain all the information stored in the server. They provide no activity by themselves, they are like passive structures ready to be modified or queried by the server.

Without Databases, the server will consist only of a simple website providing standard responses to the request methods, or pre-set data, without possibilities of inserting, querying or storing data. The use of databases will be detailed in section 5.3.3.
View

JSP is responsible for the user interface creation. JSP tends to use language such as HTML, used to create an attractive user interface. Methods and resources are used to structure the code in order to obtain a static screen ready to be shown to the external user.

JSP files can contain Java code that permits their static nature to become a little more dynamic. What really happens is that a few Java code lines are inserted between static languages in order to provide actions that permit the inclusion of dynamic data. For example a text box being filled by just a typed name, if we use just static language, we won’t be able to change this name while it’s being typed. The Java code, inserted in the static language permits JSP to contain dynamic parts.

This is used when just a few changes should be included in a static code, but it is not common when many actions are to be done in the server. Here is where Servlets take part; they are able to contain Java code in order to make the server a really dynamic web server.

Controller

Servlets seem to be like normal programs. Written in common developer’s language, Java in our case, they seem like normal actions performed in a simple program. They are structured in classes, methods and they contain variables and constants. The difference between a Java program and a Java server is that some methods are listening to external requests instead of waiting passively for inner calls.

Sometimes, Servlets can contain interface code (like HTML), included as a Java method, written as a string and exported to the interface. This is useful when the interface code included is not so complex, and it is better to include it as a few lines in the Servlet than to create an extra file to contain it. It is useful too if the code includes loops like for(), or filters depending on the code that we want to show.

In MADep, Servlets are used to develop the user interface too. As explained, MADep system provides a final interface for the medical team, but this will be entirely developed by the software team. This master thesis provides the skeleton of the server, the background code, and the database access, but does not provide a user interface. Otherwise, in order to observe what is really happening in the background, a simple user interface is needed, so it will be developed too according to the MADep system.

The MADep Servlet is divided into two essential methods, POST and GET, each of them provides a totally different interaction with the external client, POST interacts with the Android application, while GET shows the user interface as has just been explained. This will be detailed in the following section.

5.1.3 POST and GET methods (Controller)

MADep server contains two essential methods, POST and GET. As has been already explained, they are responsible for two different tasks included on the system at the hospital, but both of them have the same background essence, to access a MySQL database and to receive or send this data to the external client. The difference between them are that while POST receives data through an HTTPS request coming from the MADep application and inserting it into the
database, the GET method recovers data from this database and shows it to the medical team using an HTML web browser.

**POST**

The system at the patient’s home calls POST method through the application. This Android client sends an HTTPS request to the URL related to the server listening port. As the HTTPS connection is set as a POST, this homonymous method is started.

The POST method is responsible for extracting this data from the HTTPS request, converting them into an understandable Object, and then extracting and inserting all this data into a database. Obviously this is not as quick to do as it is to explain. Many technologies and actions are required throughout this process. JSON takes part in this method too, as it does in the Android Service in the system at the patient’s home. MySQL database will be used to store data, and it will be accessed as an external client, using a data server. Hibernate will take part in this process too, allowing data interaction from Servlet to database to be easier, simpler and more flexible.

All these technologies and steps will be explained in more detail in section 5.3.

**GET**

While POST method receives information and inserts it into the database, GET does exactly the opposite. An external client calls the method, as it is in POST, but this client doesn’t bring us data but requires it. The GET method is responsible for creating the response, including all the data requested by the client. This client will be the external medical team, who want to access the patients’ data stored in the database. The GET method will create a user web browser website, presenting the data previously collected from the database. All this process begins by executing a MySQL query to collect the data from a MySQL data server, then, this data is transformed into understandable data, and finally the HTML language creates the user interface, introduced in the HTTPS response in order to show it to the medical team, the client.

This method, and the creation of the user interface will be explained in section 5.4.
5.2 Architecture of the System at the hospital

As introduced in section 5.1.1, server is organized as a MVC paradigm, involving Model (databases), View (JSP) and Controller (Servlets). MADep server includes the View part in the Servlet that carries out most of the server tasks, particularly in the GET method. The real server structure consists of two main parts, the receiver and storage part (POST method) and the consulting part (GET method). The user interface part is a simple window to the database, to check information from patients. What is included in this master thesis is a prototype of the medical interface.

5.2.1 Main structure

Figure 5.2.1 shows a modular architecture of the MADep system at the hospital. As explained in the introduction of this section, both user interface and code task parts are mixed in this server. POST methods receive messages with data entrance, and GET method receives a call to send data in the response. Both POST and GET methods access the database, each one for a different reason.

What figure 5.2 shows is the modular structure of the MADep system at the hospital. The whole server structure as a physical computer where all the tasks are carried out, and how external clients access the server. What this server offers are two main methods.

- The POST method will be used by the system at the patient’s home. Data coming from the smartphone, sent on the HTTPS connection detailed in section 4.6, will request this method transporting all the data coming from the patient’s home system. As a response,
it will simply receive a report of the situation. Inside the POST method, a few tasks will take place. First of all data will be extracted from the HTTPS request. A conversion from the incoming parameters to a JSON Object is the second step and, finally Hibernate will be used to insert the data into the database.

- The GET method on by the medical team. Doctors don’t insert data into the database, they just consult them. So when the medical team requests the GET method only show the intention of consulting the database. As a response, all the data they want to consult from the database will be returned as HTML language in order to show it as an Internet webpage. Inside the GET method many tasks are necessary to consult this data. First, data is extracted from the database using MySQL language. When we have the data outside the database, this will be inserted as part of a HTML code included in the response sent to the medical team.

Both methods use an external MySQL database, in this case placed in the same physical computer, but it is not necessary to be like this. The access to this database will be done using a URL pointing to it. In the case of the POST method, Hibernate tools will be responsible for inserting data into the database, using classes and mapping files. In the GET method, MySQL language will be used in order to simplify the action because this will be just a demonstration extraction. The software partners will develop the final query to the database in the final MADep system.

5.2.2 Server requirements

The server will be developed in a Tomcat running server environment [17]. The structure of Tomcat and its use as a server running environment has already been explained. But to run tomcat we need some computer requirements previously available.

As well as using Tomcat, another key tool is needed for this system. MySQL databases work using MySQL tools; this should be previously installed in the server computer or in the external data server. In this case, as was detailed, the database will be set up in the same physical computer as the MADep server will be placed.

In this master thesis the server computer used is the physical machine placed on the C3-003 lab, called SENECA. Seneca is a Ubuntu UNIX core based machine that allows external public access. This is reserved for this connection, so ports and access permissions can be manipulated to install the MADep server as well as the MySQL database. What should be set up in SENECA in order to run the MADep server are:

- JRE: The Java Runtime Environment is already installed on Seneca as it is a Unix based system. It consists of several Java libraries and the JVM (Java Virtual Machine). This JVM is in charge of the Java code running; used in the Servlets, which are part of the Tomcat package. This Java code will be responsible for the tasks carried out by the server, the POST and GET methods are part of this Java code.

- Tomcat: This server running environment must be installed in SENECA, Tomcat version 6 is the one chosen for the MADep environment. Tomcat provides many libraries and tools to run MADep as a Web services server. What Tomcat really does is create all
the structure needed to contain server classes, files, resources and libraries, as well as certificates and permission files. All the tools needed to start, stop and debug the server are already provided by Tomcat.

- MySQL tools: These tools must be installed in Seneca in order to manage MySQL databases in this server. What these tools provide are MySQL language interface to create and manage MySQL databases, as well as permitting access to external clients who want to manage this database information. In our case, the MADep server will use these tools to insert and query data as if it were from an external user, but placed in the same physical computer.

Having these three main tools already installed in Seneca, what we need now is a public access to the MADep server. Providing a public World Wide Web address, Seneca will be visible to any external doctor who wants to access MADep’s GET and POST methods. This public address consists of a public IP assigned to the Seneca server, and the port used to attend these external codes pointed to the MADep Servlet.

https://seneca.upc.es

Now Seneca is prepared and ready to receive POST and GET requests from both the smartphone and the medical team. It is able to run MADep’s Java Servlets, as well as to access the MySQL database created on the same computer. Seneca will contain all the data from MADep, so the database’s whole structure will be constructed into Seneca too, using MySQL tools.

5.2.3 MADep system at the hospital Diagram

The behaviour of the system at the hospital depends totally on the method called on. If the POST method is requested by the smartphone, an insert data protocol is initiated, independent from the other protocol initiated when the medical team requests the GET method. These two main execution threads work independently one from each other, but they have a common point, the database. It’s important to remember, as introduced in section 3.2.1, that MySQL databases are able to attend several connections simultaneously, so the POST and GET methods are independent in this way too. A doctor can be querying data while an external patient is inserting it. The behaviour of these two main methods can be seen in figure 5.3, which shows the flowchart of the system at the hospital.
Figure 5.3: Flowchart of the behaviour of the system at the hospital behaviour
5.3 POST method

The POST method is one of the two main methods on the MADep server. As was already explained, the POST method will be requested by the MADep application, set on an Android smartphone system and responsible for data collection in the patient’s home.

The MADep application will initiate an HTTPS connection containing all the data collected over a period. This HTTPS connection will request a POST method contained in the MADep server, situated in the physical computer named Seneca. This request will be accessed using a URL pointing to a certain port on Seneca.

The POST method will be listening on that port while the MADep server is running. When any patient initiates this HTTPS request to a POST method, a code will be started, beginning a series of tasks responsible for the data extraction of the data contained in the HTTPS request, the information will be transformed to JSON Objects, and then extracted from JSON and inserted into Hibernate classes.

Finally, Hibernate will call MySQL server and insert this data into it. This action provides the system with the availability of the data inserted on the database to any other external client, such as the medical team.

This method will be executed any time a MADep application, coming from any MADep patient, executes an HTTPS POST request. To create this POST method, many parts have been individually developed, as will be explained in detail in the next chapters.

5.3.1 HTTPS certificate

What we first need to do to ensure a HTTP secure connection is to obtain a self-signed HTTPS certificate. As was explained/set out/shown in section 3.3.2, external certifying entities are responsible for providing this certification. As only internal users will use MADep, this certificate will be self-signed, that is enough for the security required for MADep. Remember that data is confidential, and no user names will be included in data exchange but patient ID’s will occupy their place in the identifying package.

To create a HTTPS certificate we must generate a server key, and then generate the Certificate Signing Request (CSR). Finally this CSR authenticated with the regenerated server key, will create the self-signed certificate, which will be used together with tomcat to ensure secure connections to the MADep server using HTTPS.

Generate server.key

The first step is to create the RSA Private Key, this will be done using OpenSSL toolkit, provided by Seneca as a Unix based machine. This key is a 1024 bit RSA key that is encrypted using AES and stored in a PEM format so that it is readable as an ASCII text. We should type the next line in order to generate this server.key file in the server directory.

```bash
openssl genrsa -aes192 -out server.key 1024
```

We are requested to write a pass phrase, this is like a password that will be used while the final certificate is not yet generated. This is a temporary password, but is essential until the
self-signed certificate is created. We can remove this pass phrase when we have generated the server.csr file. We will do this using the next lines:

```bash
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
```

**Generate server.csr**

The Certificate Signing Request is, as its name shows, a certificate request that should be sent to an external Certifying entity in order to be signed as a secure Internet connection. This has an economic cost and is used when external users wants to access Internet pages in a secure way.

Certification Entities are recognized as objectively reliable. This security granted to certain web connections is ideally certified with a globally recognized security seal because of the known name of the certifying entity. So everyone assumes that this is a secure connection signed by this entity.

To generate this Certificate Signing Request, we will use OpenSSL toolkit too, typing the following line on the server directory, and using the server.key just generated.

```bash
openssl req -new -key server.key -out server.csr
```

This will open many lines and fields to be filled in using the MADep information. These lines are something like this:

```
Country Name (2 letter code) [GB]: SP
State or Province Name (full name) [Berkshire]: Barcelona
Locality Name (eg, city) [Newbury]: Barcelona
Organization Name (eg, company) [My Company Ltd]: UPC
Organizational Unit Name (eg, section) [ ]: Dep. de Telematica
Common Name (eg, your name) [ ]: Jaume Olles
Email Address [ ]: joll3173@alu-etsetb.upc.edu
Please enter the following ‘extra’ attributes to be sent with your certificate request
A challenge password [ ]:
An optional company name [ ]:
```

This server.csr file should ideally be sent to an external Certifying Entity but in the MADep project it will be self signed, as is explained in the next section.

**Generate server.crt**

A Certifying Entity could sign an MADep certificate but this is not necessary. MADep is a self-contained system, only internal users, patients and doctors will use it. So security will be granted using a self-signed certificate, which everyone relies on, because it is signed by the MADep system itself. So, to self-sign our server.csr certificate request, we must type the next line, finally generating our self-signed certificate for MADep.
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

So, now, we have both the server.key and the server.crt files. These two files are not linked to any server particularly so, we should include them in the MADep server.

**Configuring the self-signed certificate in MADep**

By now, the key and the self-signed certificate files are already generated and signed, and they can be used as a secure HTTPS connection, but they are not inserted in our system.

In order to use MADep HTTP secure connection in our Tomcat running environment, we have to create a key-store containing these certificates. Tomcat will use this key-store in order to certify the connection as secure. First of all we must import the self-signed certificate into a key-store, ready to be used by Tomcat.

```bash
keytool -import -file server.crt -keystore madepcert -alias madepcert
```

Now, we must rewrite a few lines on the server.xml file on Tomcat’s conf/ directory. We must modify the lines referred to the secure connection:

```xml
<Connector port="443" SSLEnabled="true"
   maxThreads="200" scheme="https" secure="true"
   keystoreFile="/usr/local/tomcat/ssl/madepcert"
   clientAuth="false" sslProtocol="TLS" />
```

Now, our system at the hospital is ready to work as a secure HTTP connection. What we have to do now is introduce this key-store into the Android system containing the MADep application, in order to ensure that the public certificate of the system at the hospital is accepted by the Android application.

### 5.3.2 JSON

A JSON Object was created before the HTTPS request sending. This JSON was formatted with all the parameters extracted from the patient’s database on the smartphone. In addition, the HTTPS request doesn’t contain a JSON Object but its serialized brother, a JSON String. This String can be read and printed as a simple String. If we print it, we can see that all the structure of the JSON is written in a simple String. In order to extract data from the String, we have to parse it.

Some libraries contain a very useful method called JSONObject.fromObject (String JSON-String). This method is the key to deserialize the JSON String. What this method really does is render a JSON serialized Object from the syntax’s of a JSON String. This is the reverse step from the one previously carried out by the MADep application on the smartphone.

So, now we have recovered the JSON Object exactly as we have built on the system at the patient’s home. We have already finished the transaction, because we have the same Object that we had before on the other side of the communication. What we have to do now is to ensure
that this data coming from the patient’s smartphone is conveniently stored in the database of
the system at the hospital.

The first step, when we have recovered the JSON Object is to extract the data from this
mapped file. In order to do this the developer must know the key-value pairs established
on the system at the patient’s home. Every value, as introduced in section 4.5, has its own
key identifier. So what we must do is recover every single value using its own key identifier.
Value by value, we will create a variable according to the type of parameter we want to extract
from the JSON Object, and then call the method JSONObject.getInt("KEY_ID_1"), JSONObject.getString("KEY_ID_2"), and so on, depending on the type of parameter. What we are
really doing is what we had done in the MADep application, but in the opposite direction, as
shown in figure 5.4.

![Figure 5.4: Evolution of the JSON serialized map to the list of common parameters, same steps
as on the system at the patient’s home, but in the opposite direction](image)

After this action, we will have a collection of parameters, but inserting them into the database
is not as easy as it seems. And even if it was, we have to ensure its compatibility with any
database language, and amount of servers conforming the final database. This will not be pos-
sible if we use directly here the MySQL language to insert this data into the MADep database.
In the next sections we will see how a MySQL database works, and why we need Hibernate
tools in order to ensure some points of communication between the server and the database.
5.3.3 MySQL

As was previously presented on section 5.1.2, the server is built using the MVC paradigm. The Model part will be developed using MySQL in the MADep system. MySQL database stores the data coming from each patient on the MADep network. This MySQL database has been previously created in the system at the hospital, following the structure of the system at the patient’s home database, but with the assumption that the data will be kept here until the treatment finishes or the medical team decides to destroy it in every patient case. Another important point is the storage of several patients in the same database. Patient information will be stored in a Patients’ Table, where all the information from each patient will be stored and linked to the MADep Patient ID, which will identify the rest of the data received through the network.

This MySQL database is placed in the same server as the main MADep server is. Which means that the remote access to the database will be simply done pointing to the localhost URL. What the system really does is to insert data into a MySQL database to open a MySQL environment inside the Java code. When the system wants to access the database, a MySQL Driver is used. We initiate a connection to this database, using its URL (localhost in our case). We must use the user and password to access the database. When this connection is already set, a common environment is running. The developer should use MySQL language to fill Strings with the MySQL actions and the parameters that he wants to insert or query.

The POST method uses Hibernate as a tool to insert data into the central database. Hibernate will translate any database language, transparent to the Servlet code. To make it easier to connect to the database, and to write the code needed to insert data into any database, we will use the same methods independently of the database placed on the other side of the connection.

The creation of the database will be done previously to any action in MADep; it is supposed that this should be constructed before MADep is distributed. In the case of the patient ID’s, we assume that every doctor will insert the patient into the MADep database when this patient joins the MADep system treatment. In this MADep prototype, conforming to this master thesis, the patient information will be inserted through the MADep application and sent to the system at the hospital as a new patient.

5.3.4 Hibernate

Hibernate, as previously detailed in section 3.2.2, is an Object-relational mapping library that helps developers to interact with databases. Hibernate maps an Object-oriented model to a relational database, MySQL in our case.

Hibernate provides the developer classes instead of tables. This is an easier way to work because, in Java language, we can work directly with classes, methods and variables. Instead of using the MySQL language, a complex and strict syntax language, not matching Java methods and complicated to refer to when we are using a Java environment.

Using Hibernate we just instantiate a previously written Hibernate Object. This Hibernate Object has its own Hibernate mapping file, corresponding to a database table. Every variable of this Hibernate class has its own column in the table. This relation is the aforementioned Object-relational mapping.

Files needed to create this Object-relational mapping are simple and similar whatever the
table is. As explained in section 3.2.2, two essential files are used for each table in the database: the Hibernate class and the mapping file.

The Hibernate class tends to be named after its corresponding table into the database. This Object contains as many variables as columns on the table, their names and variable type match the columns of the database. This class has two methods assigned to each variable, setter method to pass the variable from the code to its particular class, and getter method to do just the opposite, extract the variables from the Hibernate class to the main code. These methods will be the interface between the main code and the database from now on. An example of a Hibernate class is:

```java
public class Patient_data {

    private int id;
    private String name;
    private Date date;

    public Patient_data() {
        //Empty constructor
    }

    public void setId(Int id) {
        this.id = id;
    }

    public int getId() {
        return id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public Date getDate() {
        return date;
    }

    public Date getDate() {
        return date;
    }

    ...
}
```
The mapping file has the same name as the Hibernate class, it is written in xml language and is a description file connecting every variable included in the Hibernate class to its linking column in the table. The description consists of the variable name in the Hibernate class, its corresponding name on the database, and the type of variable it is. An example of a mapping file, related to the previous Hibernate class is:

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="org.MADep.main.Patient_data" table="PATIENT_DATA">
    <id name="id" type="int">
      <column name="PATIENT_ID" />
      <generator class="assigned" />
    </id>
    <property name="name" type="java.lang.String">
      <column name="PATIENT_NAME" />
    </property>
    ...
  </class>
</hibernate-mapping>
```

Having all the tables mapped into Hibernate classes and their linked mapping files, the use of Hibernate consists of using simple Hibernate methods. To insert data into the Hibernate Object, we just call the setter method corresponding to this particular attribute and we pass it on as a value. The Hibernate class will include it as an internal variable, linked to its column on the mapping file. If we want to get a variable from the database, we just call on the getter method, and the variable will be returned. These are a few lines of the MADep server code in the POST method. These lines show how to insert data into a database using Hibernate:

```java
int _id = 00000;
String _name = Peter

Patient_data patient = new Patient_data();
patient.setId(_id);
patient.setName(_name);
```

With all the Hibernate classes already filled in with incoming data, what we need to do now is to create the final link between Hibernate and the database we want to use. For this purpose we need an extra file, the Hibernate configuration file. There will be a single file for each database, not for every table but just the database containing all of them. This file identifies how to connect to the database, localhost in our case but it can be placed in an external server or even in several data servers. This file contains the user name and password to access this
particular database, previously inserted by the developer. Finally, what this file indicates is the
type of database language that will be used for this connection. An example of our Hibernate
configuration file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.current_session_context_class">thread</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.password">12345678</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost/MADepDB</property>
    <property name="hibernate.connection.username">MADepClient</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <mapping resource="org/MADep/main/Patient_data.hbm.xml"/>
    ...
  </session-factory>
</hibernate-configuration>
```

This file will be called on in the main code using a SessionFactory class. This is a helper
class responsible for creating the communication channel between Hibernate and the database.
It is a written code, what it really does is build a new Session responsible for initiating the
database using the Hibernate configuration file. So everything typed on the configuration file
will be used by the SessionFactory to create this communication channel. This Session is the
channel used to execute commits, queries, as well as to initiate and close the connection.

When the communication is already established, the rest of the code is transparent to the
configuration file and the database placed on the other side of the channel. This is the main
advantage of Hibernate, it is transparent to the database, and it is only identified in the con-
figuration file. Even many configuration files can be included in the code, and then when we
create the SessionFactory we choose which configuration file we want to use. It means that the
database placement can be changed at any time. MySQL can be converted into another totally
different database language, but Hibernate will keep working as it did. The user can change,
the password can change, the database can change, the dialect can change, but the Hibernate
methods used in the main Java code won’t.
5.4 GET method

The GET method is the second of the two main methods on the MADep server. As well as in the POST method, data is received and stored in the database, the GET method has the opposite task, it is responsible for extracting data from the database and sending it to the client when it is ask for. It’s important to notice that the GET method will be requested by the medical team, and right now it works on an HTML Internet browser.

An external client, the medical team in our case, uses the GET method in the server to query the database and obtain certain patient data to observe and study the patient’s lifestyle, which is the main objective of the MADep system. The use of this method is transparent to the doctors; they only access a given Internet website and interact with it in order to view the patient’s data.

This action is composed of many tasks carried out by the system at the hospital. The GET method is in charge of these little tasks that compose the final user interface which is ready to be used by the medical team. What really happens through this website query is the sending of a HTTPS request from the external client to the server, calling on the GET method.

When a request for the GET method comes into the server, it is empty and it only asks for a response containing the data query. This request is not really empty at all; it contains the patient ID corresponding to the data that the doctor wants to check. The GET method uses this patient ID to select the corresponding data from the database, using MySQL language in order to obtain this data from the MADep database, set in the same physical computer, in our case, Seneca.

When this data is already in the Java Servlet, these are included in an HTML code. This HTML code composes the entire website shown to the medical team, and inside it the data is dynamically placed in a sorted table.

The medical team can access this data while patients insert their own data in the same MySQL database because, as explained in section 3.2.1, MySQL allows multiuser access at the same time. This is one of the main advantages of MySQL compared with other database dialects.

5.4.1 Database query

When the GET method wants to access the database we need to initiate a query in order to extract data and insert them into the HTTPS response. Remember that, for now, the medical team will only check data, and won’t insert any new data. So, in the server, data is inserted in the POST method using Hibernate, and is queried in the GET method.

MySQL environment will be used to query data when it is requested by the medical team. This action could be developed using Hibernate too, or even another interface that makes MySQL queries more comfortable. But we should remember that this part will eventually be developed in another way, maybe using an external program with its own database query methods.

This user interface developed in this master thesis is just a prototype providing us with the possibility of consulting the database in a quick and easy action, just by checking a website. But this will result in a whole complete program with menus, options, and attractive screenshots presented to the medical team. Right now, just a simple MySQL query will be used in this user interface. When a MySQL query is done in a Servlet, what we need to do is use the
MySQL Driver in order to include MySQL language into a Java code. So, first of all, we must initialize the MySQL Driver from the MySQL library that we have previously added to the Tomcat libraries. It is essential to include this library in the MADep server project, because it is responsible not just for this communication but also from any other access to the database.

To use the MySQL Driver we only need to load it with a reference to its location in the MySQL library. After loading the MySQL driver, we create the connection to the database, using the same parameters as Hibernate uses in the POST method. We must include the user name, the password and the URL to the database, localhost in our case.

To use the MySQL Driver we only need to load it with a reference to its location in the MySQL library. After loading the MySQL driver, we create the connection to the database, using the same parameters as Hibernate uses in the POST method. We must include the user name, the password and the URL to the database, localhost in our case.

//Import driver from MySQL, fix the DB name, user and password
DriverManager.registerDriver(new org.gjt.mm.mysql.Driver());

//Initiate the connection to MySQL
Connection conexion = DriverManager.getConnection(
    "jdbc:mysql://localhost/MADepDB",
    "clienteMADep", "12345678");

Statement s=conexion.createStatement();

Now we have the connection already set. What we need now is an environment to write MySQL actions in MySQL language, so we use the Statement to execute a query involving all the data we want to extract from the database, for example a simple "SELECT * FROM PATIENT_DATA". This will give us access to this data in order to extract one by one the selected parameters.

As we did previously in Hibernate, MySQL library gives us the option of using methods in order to extract data from the MySQL database. But this time it is not as flexible as using Hibernate tools, now we must include the MySQL column name directly into the method, and the value of this column is returned as a parameter.

int patient_ID = rs.getInt("PATIENT_ID");
String patient_Name = rs.getString("PATIENT_NAME");

These variables recovered from the database are now in our Java Servlet, we can use them as we wish. What we will do is insert this data in an HTML code, responsible for creating the user interface, as explained in the following section.

5.4.2 HTML

HTML language is a tool to generate websites that we will use to create the user interface of this master thesis. As it will be a simple user interface showing the data requested by the medical team, it will not include complex language, methods or external resources to make the website more attractive or interactive.
In this case, the website will simply show the data throughout tables dynamically created depending on the data stored on the database. When a HTML website is created we should be able to make it flexible to contain an unknown amount of data. As the HTML code will be not written into a JSP but a Java Servlet, this will be as easy as including a loop depending on the amount of data queried.

As explained in section 5.1.1, View and Controller parts are usually two independent parts in an ideal server. In our case, as was already explained, both View and Controller parts are placed in a single Servlet. The website only appears when a doctor requests it using the GET method, so website generation will be included as part of this Java Servlet responsible for handing the GET request. So, the GET method will incorporate a part of the code that will create the website. What we will do exactly is initiate a PrintWriter in the HTTPS response using HTML content type. This way, the creation of the website is as easy as writing HTML language into a simple String and inserting it in this PrintWriter. This will be sent as a response to the requesting doctor, who accessing the GET method using a website browser will obtain a website interface showing the patient data.

```java
response.setContentType("text/html");
PrintWriter out = response.getWriter();

out.println("<body><h3>Patient Information: </h3>");
out.println("<b>Patient MADep ID:</b> " + rs.getInt("PATIENT_MADEP_ID") + "<br/>");
out.println("<b>Patient Complete Name:</b> " + rs.getString("PATIENT_NAME") + rs.getString("PATIENT_SURNAME_1") + rs.getString("PATIENT_SURNAME_2") + "<br/>");
out.println("<b>Patient Alias:</b> " + rs.getString("PATIENT_ALIAS") + "<br/>");
out.println("<b>Patient Birth date:</b> " + rs.getString("PATIENT_BIRTHDAY") + "<br/>");
...
```

This could be an example of an HTML code included in a Java Servlet, creating a simple dynamic table while the database query method rs extracts data from the database. This code can be modified, extended and composed as a more attractive website. But these actions will be developed by the software partners, together with the development of a more interactive program which will be presented to the medical team.

### 5.4.3 Medical team interface

As it was already explained in previous sections, the final medical team interface will not be the response obtained in this GET method. The response of this method will just provide the data queried in the HTTP response, given to the final interface.

When this data is received in the program responsible for the interaction of the medical team with MADep, this data will be organized and shown in a comfortable and attractive screen to
the doctors. This program, developed by the software team, should be as interactive as the medical team needs, and should be able to generate interfaces according to the queries made by the doctors.

This program will be the image of the MADep presented to the medical team, as well as the MADep Android application there is also the image of MADep for patients. So, it means that this program should be developed in detail according to the medical team’s requirements.
6 Demonstrator

The System at the patient’s home and System at the hospital are already detailed in previous chapters, explaining every kind of technology used to build them and their structure and behaviour inside the MADep system. In this chapter, the prototype developed for this master thesis will be shown and explained in detail step by step, joining both system at the patient’s home and system at the hospital and creating a single and complete prototype.

Beginning in the hospital, where the patient initiates the treatment, the doctor will start the MADep system on the smartphone, where the patient will be registered in the system database. After this initial action, the patient will be free to live his own life in his own house, which begins the MADep treatment. Sensors obtain data by monitoring the patient’s behaviour and send them to the smartphone, which will insert them in the local database. On a previously established day, the patient will interact with the Virtual Assistant, and then, data will be sent to the system at the hospital. Doctors will then be authorized to access the MADep database to check data collected by MADep system in order to provide the patient with a better treatment for his particular depressive case.

6.1 MADep system at the patient’s home

MADep system is divided into two main subsystems, the System at the patient’s home and the System at the hospital. As has been already detailed in the System at the patient’s home chapter, the first subsystem is focused on the patient’s house, especially on the smartphone provided by the medical team. This application has been designed to cover all the requirements of MADep in the patient’s house subsystem and offers the patient a comfortable environment to behave in a natural way. It is important to notice that a relaxing background has been chosen instead of the standard black background. This is because of the mood of depressive patients. Doctors prefer to show relaxing images to patients, which is why hospitals walls are painted using relaxing colours.

When a patient is diagnosed as having depression the medical team can decide to treat this patient using MADep. This will be a helpful tool to monitor the patient’s lifestyle in order to provide him or her with better treatment and solution to the illness. When doctors decide to initiate MADep with a particular patient, this will begin with an authorizing protocol signed by the patient, who allows the medical team to install sensors in the patient’s house, to collect information from these sensors and to manage this private information in order to improve the patient’s depression. If the patient authorizes these conditions, the MADep treatment begins.

First of all, many sensors will be installed in the patient’s house, a bed sensor, a special weighing scale, sensors on the fridge, and many more. When these sensors have been installed in the patient’s house and checked to see if they work correctly, the doctor is prepared to introduce this patient into the official MADep database and begin the treatment.
Now is time to insert this patient’s data into the MADep database. In the hospital, both doctor and patient will install the MADep application into the smartphone. The patient must accept the terms and conditions of the MADep application, shown on the smartphone when the application is installed. When the patient authorizes the application conditions, it is time to introduce this patient’s information in this singular application provided to the patient.

The doctor will be responsible for inserting this data into the system. The first screen asks us to decide between registering a new patient, and initiating MADep, as shown in figure 6.1. Now it is time to introduce a new patient into the MADep database, so the doctor should choose "New Patient".

![Figure 6.1: Screenshot of first MADep application screen](image)

When the option of "New Patient" is clicked, a new screen appears. In this screen the doctor fills in the personal data of the patient. First of all, the doctor must insert the Patient MADep ID. Every patient is identified by the system with an ID, this option provides privacy to the patient and makes it possible to mark all the information of this patient as an anonymous number instead of a name. So, this MADep ID will, from now on, identify this patient in front of any MADep activity.
Personal data, such as the patient’s name and surnames, date of birth, causes and characteristics contributing to the depression. Identification will be requested too. When all this information is completely filled in on the New Patient form, as shown in figure 6.2, it will be inserted into the local database placed on the smartphone. At the same time, this information will be automatically sent to the MADep central database placed in the hospital in order to register this patient in the MADep main database. All the information received from now on will be checked to join an active MADep patient, and then inserted in the main MADep database.

![Figure 6.2: Screenshot of New Patient Activity](image)

From now on, the smartphone is an important object for the patient’s treatment. The patient is informed that he should keep this smartphone with him all the time, because the smartphone is responsible for most of the data sent to the medical team, and the information collected by the MADep system needs to be as reliable as possible.

The doctor informs the patient of the importance of having the smartphone with him, explaining that a sensor is located in the smartphone itself. Now it is time to inform the patient that an interaction between the patient and the MADep application is not needed, because the system will work automatically. The only thing the doctor requests is that the patient answer a few questions on the Virtual Assistant every 15 days.

This prototype includes an interface to see how the sensors collect information from the external environment as well as the Virtual Assistant responsible for the interaction between the patient and the MADep. In the main menu, shown in figure 6.3, there are 3 options. The Weight List and the Slept Hours List options are very similar; they show us the data list collected from the external sensors. The difference between them is that Weight List provides us the information collected by the weighing scale’s sensor, and the Slept Hours List contains the list of the hours slept by the patient every day. The third option brings us to the Virtual Assistant, where MADep will show a test to the patient, who will be asked to fill it in.
When we access the Weight List what we obtain is a list of the data collected from the weighing scale. Here we are only able to see the values of the patient’s weight, but in the final MADep database, data and time of the weight are shown too. Even while we are looking at the weight list, the weighing scales can send new data so, if it happens, a new weight value will appear automatically on the list, and inserted in the local database together with the actual date and time. This demonstrates that MADep is a non-interactive system with work done automatically by sensors and the smartphone. The patient just has the responsibility of weighing himself everyday, and MADep will do the rest. The weighing scales will collect the weight value, and then send it to the smartphone using Bluetooth, while the smartphone will receive this value and insert it into the local database. The weight list refresh is shown in figure 6.4.
As shown in figure 6.4, another available interaction is the sleeping hours of the patient. This information will be received in an automatic action, as the weight does. In our prototype, to test different ways of collecting data, this Activity allows us to introduce data using a manual action. When the patient accesses the Slept Hours List, no data appears on it. To introduce the amount of hours slept, the patient can press the menu button in order to access a manual data insert interface, where the patient can insert the amount of hours manually. When we return to the Slept Hours List, it has been refreshed, and the amount of hours has been inserted into the database. The patient does not carry out this action in the final MADep prototype. The patient just lies on the bed. Sensors, Bluetooth and the smartphone do the rest of the work, as they do on the Weight values. Figure 6.5 shows manual data insert used in the Slept Hours List.

Figure 6.5: Screenshots of MADep slept hours list and how we can enter manual data

In a normal procedure, the patient doesn’t interact with the smartphone at all. They are only asked to weigh themselves everyday, and to live their normal daily life. Sensors will collect every movement, habit and lifestyle of the patient and in an automatic process insert all this data into the local database. But now it is time to make the patient interact with MADep. Every 15 days, the Virtual Assistant will ask the patient a few questions about his lifestyle, his habits and his mood. The patient can ask these questions when he wants all through the day. This test, as explained in section 2.3.2, is a standardized test prepared to interact with depressive patients, and tries to be a tool to complete the information collected by MADep all through these 15 days. An example of the Virtual Assistant test is shown in figure 6.6.

Now it is time to send all this information to the medical team. Every 15 days, when the Virtual Assistant test is completed, all data will be sent to the hospital to store it on the main MADep database, containing every patient’s information.

Not all the data in the smartphone will be sent, just that information which has not been sent yet. This is possible due to the checking done by the main MADep Service when it queries the database in order to collect the information ready to be sent.

The user interface will come back to the main menu when the Virtual Assistant test has been completed, but MADep application begins background work responsible for sending all the information to the hospital. Most important on this point is not to disturb the patient, because we should remember that MADep tries to be a non-interactive system working on automatic
actions. So the transaction will take place in background, using a Service, as it has already been detailed in section 4.3.3.

At this moment, the Service begins an HTTPS connection with the MADep system at the hospital in the hospital, sending an HTTPS request containing all the data collected from the patient’s lifestyle during all of these 15 days. So, it’s time to leave the subsystem placed in the patient’s house in order to access the subsystem placed on the hospital.

6.2 MADep system at the hospital

The second subsystem is based on the hospital, as explained in detail in the System at the hospital chapter. This system at the hospital will receive data from every patient included in the MADep system; as well as that, it will provide the doctors with a website to check all this information with the target of give the patient better treatment. This system at the hospital has been designed in a relaxing user interface related to the design of the MADep application on the smartphone. Following the same design of the application, the website accessed by doctors tries to be a reflection of the application used by the patients; presenting to doctors the idea of what their patients will see.

On the system at the hospital two important actions are carried out. The action related to the MADep application and the incoming data, these are done totally in the background, as explained in section 5.3. What the server really does is receive a POST request coming from the MADep application with all the information of the patient. The POST method initiates a sequence of actions responsible for inserting this data into the main database, but this has no effect on the user interface.

What really provides interaction with the user is the website, built as the GET method of the MADep main server. This tool, provided as a website in this MADep prototype, will be used by doctors in order to check patients’ data, study each case and decide which process will be chosen for every case. When the doctor inserts a patient into the database, at the initial phase of the MADep treatment, the application calls the GET method in order to insert this patient’s
information into the main database. From now on, this patient is included in the MADep project and has his own space and MADep identifier into the system.

We should remember that MADep is an automatic system, avoiding any interaction between the patient and system and of course between the doctor and data collection system. The doctor just inserts the patient into the MADep system and then all the information is automatically sent to the system at the hospital, available to be checked by the medical team whenever and wherever they want to.

The website is accessible from any web browser using a HTTPS connection, using the URL of the server dedicated to MADep queries. When a doctor sits on his chair at his desktop, and decides to check how his MADep patient is evolving, he just turns on his computer and opens his favourite web browser. The doctor accesses the previously stored favourite URL that drives takes him directly to the MADep main page, welcomed to an initial page asking him which patient he wants to choose, as seen on figure 6.7.

On this website, the doctor has to choose a patient or directly insert the MADep ID of his patient. If he clicks on a name, the MADep ID matching this patient name will automatically be inserted into the MADep ID text box. Now, with the patient already chosen, he proceeds to check the data collected from this patient since the day of his insertion into the MADep system. If the doctor decides to insert the MADep ID directly into the ID text box, it is possible that he will insert a wrong MADep ID, or maybe this patient has left the MADep program. When this happens, a warning page appears in order to notify him that this patient is not included in the MADep system, as is shown in figure 6.8.

No problem! Doctors have a list of the current patients included in the MADep system in the initial page. They only need to choose one of them and it will appear automatically in the ID text box. Now we ensure that this patient exists on the MADep system. The doctor will be redirected to a page showing all the patient information and data collected, as shown in figure 6.9.

Distributed in comfortable tables, the doctor can evaluate the evolution of the patient, his lifestyle, his alimentary habits, how many hours he sleeps and if his weight is set on the correct
limits. Apart from this, the result of the Virtual Assistant test are shown in a table easy to observe, which will provide the result of every test, answer by answer and the total of them. In addition, we have every answer distributed by columns, so it is quick to evaluate the improvement of the patient in each topic presented as a simple question.

Doctors are informed of their patients’ lifestyle, habits and evolution every 15 days, without the necessity of leaving his desktop. This is much better than a single session of 30 minutes duration once every 1 to 3 months, depending on the patient treatment. MADep will alert the doctor if a limit is exceeded by the patient, and will provide a useful tool to detect a patient’s relapse in long-term treatment.

MADep system will be a useful system to monitor depressive patients in their own houses and report this information to the doctors in their workplaces. This system will make the treatment of depression easier, quicker and, especially, closer and more reliable.
Figure 6.9: Screenshot of Website showing all the patient collected data
7 Conclusions and Future Lines

7.1 Conclusions

MADep has been developed as a monitoring system to assist depressive patients in order to improve their treatment using information about their daily lives. Multiple users will work simultaneously with the same system due to the multiuser architecture, while a central server collects all the incoming information. Doctors can monitor patient’s behaviour, and patients don’t even have to modify their lifestyle.

The main target has been successfully achieved, providing the whole system an autonomous and robust structure. Patients and doctors will be connected in a way never seen before, closer than ever. The doctor will know how the patient behaves and his lifestyle in his own house, not only in the hospital.

System at the patient’s home and System at the hospital have been developed in two different and individual systems working as a single one. The fact of developing two different systems has become an advantage. HTTPS connection permits both subsystems to work independently. This permits future local improvements in the system without the need to change the rest of the system.

The main advantage of defining two completely independent systems is the use of the most suitable technology for each part. Apart from this, the use of widely different technologies in both subsystems have been a personal challenge during the development of the project.

During the development of the patient’s subsystem, I acquired new knowledge about the structure of applications and systems on small devices. Mobile technologies and smartphone applications are the future in communication technologies, so being trained in these technologies and getting to know their possibilities is a key aspect to work on the field of communication engineering.

On the other hand, the technologies used at the hospital subsystem of MADep are completely different from smartphone applications. The Client/Server architecture is in the base of the whole internet, and this master thesis gave me the opportunity to acquire new knowledge about server technologies that is going to be essential for future projects. A telecommunication engineer must know how the network works and should have the knowledge of what a server is and how it can be developed.

The data exchange part of MADep system uses leading technologies in communication services. HTTPS, Hibernate, server tools and Android are used in many current communication systems and projects, and it is expected that they are going to be used in future applications too. Learning about these technologies has provided me with knowledge about present-day tools used in telecommunications engineering, providing me skills to develop and to keep on learning.

This master thesis gave me the opportunity to apply knowledge acquired during the university
course and to develop new products using new and current technologies. As a master thesis involved in a real project with several partners and dozens of participants, it has given me the opportunity to get involved in teamwork on a project of this size for the first time.

7.2 Future Lines

MADep has several potential improvements. Many of them could be developed in a future prototype to improve the whole system quality and some of them should be included in the prototype used on real patients.

MADep is a clinical system responsible for clinical data exchange, patient’s privacy is essential and data confidentiality must be guaranteed in the final prototype. This is the main target to be tested on a final prototype. Security on the final system at the hospital must be accurately studied, especially where the database will be placed. MADep main database will contain private clinical data of all MADep patients. Privacy is essential for this database, which should be accessed only by patients and doctors. Even an extra security filter could be inserted; permitting the access only to the doctor assigned to each patient. Data encryption must be present on communications between MADep main database and final medical user interface, as well as between user application and MADep system at the hospital. Bluetooth communication used in the patient’s house subsystem must be encrypted too, and verified on every data exchange between sensors and smartphone. This Bluetooth communication must assure that information will only be sent to the selected device, avoiding any possibility of connection to external devices. Even the smartphones must be protected against external attacks trying to obtain data from the smartphone database. The idea of erasing data already sent to the system at the hospital can avoid future attacks trying to obtain data from the smartphone database.

Sensors and smartphones need to be recharged or work with autonomous batteries. The necessity of avoiding cables in the patient’s house has been shown, and wireless communications in all the system allow us to work with these requirements. But energy supplied to these devices could be a problem to guarantee this characteristic. Another limitation could be the loss of battery on the smartphone. If the main device loses its battery, MADep will stop working. We need to ensure that the battery is enough to satisfy the requirements of the system, but this can only be studied on a real smartphone model chosen to be the MADep main device. This must be checked on the final prototype.

If the smartphone loses Internet connection, data exchange communication won’t be completed. This is partially solved using the flags inserted on the smartphone database indicating if data transferring has been properly completed or not. The solution to this problem comes when we try to resend the information when the Internet connection is re-established. This could be solved using a Broadcast Receiver to tell the system that the Internet connection is lost and re-established again. When the system receives this broadcast, it initiates a checking of every data item, starting a new communication to resend this information. These limitations could be solved in a future prototype, when every part is completely developed and the final system is tested. The relation between the sensors systems, the smartphone and the data exchange protocol will be finally solved in this final prototype.

The amount of patients that the system at the hospital can manage working simultaneously could be a problem if the system grows to include a great number of users. The data exchange
between system at the patient’s home and system at the hospital is completed in a short period of time, but if a large number of patients try to send information exactly at the same time to a single server, the system could become saturated. The use of several servers working together using a single database would solve this problem. This array of servers can provide more capacity to simultaneous requests than a single server. A real study can be done in a real scenario when a complete prototype is finished, providing a QoS of the system. Assuming that a single communication is sent by each periodically, the system seems to be prepared to work with a large number of patients.

A better user interface for doctors should be designed. The new interface could provide an option to classify information in a more comfortable structure. The final user interface should be distributed in menus and lists to offer doctors an easier way to check a patient’s data. The final prototype will include a program containing all these specifications. An alarm system could be included in the system to warn doctors of a dangerous situation in a patient’s case. This alert could be sent to the doctor using e-mail, SMS or any kind of instant communication. This will provide the system with not only a monitoring nature but also a safety aspect working in real time. A feedback system could be included in the communication between doctor and patient to allow the medical team to send messages to the patient. These messages can contain advice, questions or even the intention to arrange a meeting.

MADep can be improved in many ways to create a better system in each prototype version, including extra tools and system improvements to make the whole system more reliable, secure and flexible.
Bibliography


