SYSTEM TO PREDICT BIPOLAR DISORDER CRISES ANALYSING MASSIVE DATA

Ana María Martínez Gómez

DOUBLE DEGREE IN COMPUTER SCIENCE AND MATHEMATICS
COMPUTER SCIENCE FACULTY
COMPLUTENSE UNIVERSITY OF MADRID

BACHELOR THESIS
5º academic year

16th June 2016

Director: GUADALUPE MIÑANA ROPERO
Codirector: MARÍA VICTORIA LÓPEZ LÓPEZ

Contents

List of Figures

List of Tables

Abstract

English

During the development of this project I learnt about *Big Data*, *Android* and *MongoDB* while helping to develop a bipolar crises-prediction system by analysing massive quantity of data from diverse sources. Concretely, I did a theoretical part about NoSQL database, Streaming Spark and Neural Networks and then I designed and configured a MongoDB database to be used in the bipolar disorder project. I also learnt about Android and designed and developed an Android mobile application to collect data to be used as an input in the crises-prediction system. I did usability testing upon completion of the project too.

Spanish

Durante el desarrollo del proyecto he aprendido sobre *Big Data*, *Android* y *MongoDB* mientras que ayudaba a desarrollar un sistema para la predicción de las crisis del transtorno bipolar mediante el análisis masivo de información de diversas fuentes. En concreto hice una parte teórica sobre bases de datos NoSQL, Streaming Spark y Redes Neuronales y después diseñé y configuré una base de datos MongoDB para el proyecto del trastorno bipolar. También aprendí sobre Android y diseñé y desarrollé una aplicación de móvil en Android para recoger datos para usarlos como entrada en el sistema de predicción de crisis. Una vez terminado el desarrollo de la aplicación también llevé a cabo una evaluación con usuarios.

Keywords

English

Android, mobile applications, databases, NoSQL, MongoDB, Big Data, bipolar disorder, Spark, neural networks.

Spanish

Android, aplicaciones móviles, bases de datos, NoSQL, MongoDB, Big Data, trastorno bipolar, Spark, redes neuronales.

1 Introduction

1.1 Objectives

Bipolar disorder often leads to periods of sick leave and close attention, causing economic and social problems in work and family environments. Most patients suffer crises that can be avoided through early prediction. The objective of my project is to learn about *Big Data*, *Android* and *MongoDB* while helping to develop a bipolar crises-prediction system by analysing massive quantity of data from diverse sources such as mobile phone sensors, voice monitoring and accelerometers. This project is an small part of a bigger project (see [13]) supported by the UCM groups G-TeC and GRASIA (figure 1.1). There were similar studies made, the main difference is that in this case we are going to pay close attention to high sleep quality as we consider it to be a determinant factor which will improve predictions significantly. As it is a really ambitious project I have developed a part of it, as it is detailed in the Work plan section.

Figure 1.1: UCM groups: Grasia and G-TeC

1.2 Antecedents

I had not had any prior experience with *Big Data*, *Android* nor *MongoDB* before starting this project. However, some of the subjects I took at the University played a pivotal role while learning to work with these technologies. Specially, the subjects of *Tecnología de la programación*, where I learned important concepts of OO programming and I programmed in Java, and the subject of *Bases de datos*, where I acquired sound knowledge of relational databases and SQL, and *Diseño de sistemas interactivos*, which helps me in the designing and usability testing of the mobile application.

1.3 Work plan

As I had not had any previous contact with *Big Data*, the first step was to investigate the technologies that are currently being used in this field. As it is a very wide topic, I was recommended to concentrate on *NoSQL databases*, *Streaming processing with Spark* and *Neural networks*. After that, as the project has just started, I focused on collecting data to be analysed and on the database design and management and its final integration. To collect data I developed an *Android application* which users will use to fill in a daily questionnaire. It allows users to register and to sign in and it also collects other information measuring the interaction of the user with the device. Regarding the database, I decided to use a *MongoDB database*. I installed a MongoDB server, designed the collections and users and illustrated how the data can be queried. I also commented on the *MongoDB* Drivers that could be used to integrate the database with the different clients that need to access to the database server and I explained in detail the *Java Driver* which I used in the mobile application.

1.4 State of the art

1.4.1 NoSQL databases

The environment for data has significantly changed since the introduction of relational databases in 1970:

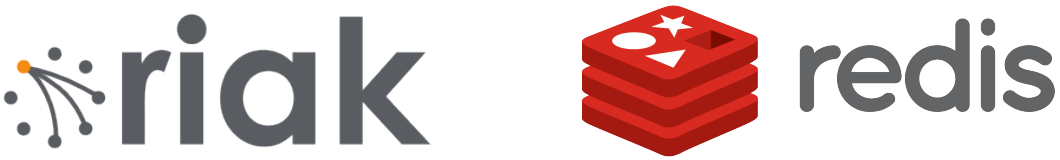
- **The emergence of cloud computing** where data is stored in multiple servers. When performing queries in a complex relational database, several large tables are joined and executing distributed joins is a very complicated problem.
- **The need to store unstructured data**, such as social media posts and multimedia. This is not possible in relational database tables where only structured data can be saved.
- **Database schema needs to change rapidly**. In relational databases it is needed to define the schema in advance and alter it is really costly.

As a result, it does not still made sense to define objects as sets of tables (or relations). Non SQL or not only SQL (NoSQL) databases, which are more flexible, distributed, horizontally scalable and in many cases provide better performance, have gained traction, specially in *Big Data* and real-time web applications. However, as they use an alternative model to the one used in relational databases, they usually lack of other desirable properties such as security in the transactions, integrity and consistency.

It is clear that NoSQL will be useful in some parts of our project, although that doesn't mean that SQL is not going to be used. What is not so clear is what is the best type of NoSQL data store as there are many different types: document, key-value, column, graph, multi-model, etc. Following the advice of my directors, I am going to analyse key-value and document-based, explaining how they work, their advantages and disadvantages and their possible application to the project.

The main references that has been used for this section are [34], [30] and [23].

1.4.1.1 Key-value databases



(a) *Riak* (b) *Redis*

Figure 1.2: Key-value databases most popular implementations

Key-value databases or key-value stores are the simplest model. A key-value database is a hash table which stores pairs *(key, value)*, which only makes sense to use when all accesses to the database use a primary key. We could already have a pair *(key, value)* using a two-columns relational database, although in that case the *value* column was restricted to have a single basic type of data (for example *Integer* or *String*), while in a key-value database any kind of data can be stored. The database does not care of what is inside, as it is the application who must understand what it is stored. Because of the use of primary-key access they have great performance and can be easily scaled.

There are many kinds of key-value stores. Some of them allow storing complex value types and provide additional operations over these structures. Some others provide iterators over the keys. There are several private options, such as *DynamoDB*, which is provided by *Amazon*, but as we prefer free of charge options, we are now going to concentrate on two of the most popular open source key-value databases: *Riak* (figure 1.2a) and *Redis* (figure 1.2b).

Riak is an implementation of *DynamoDB* with advantages features. It is a distributed key-value database where values can be anything, from plain text to images, and relationship between keys are handled by links. All data can be accessed using a simple HTTP interface. *Riak* offers high availability, fault-tolerant, flexibility, scalability and it supports advanced querying via map-reduce. However, it lacks robust support for custom queries and do not have foreign keys.

Redis supports lists, sets and hashes and can do range, difference, union and intersection operations. It has one of the most robust query mechanisms for a key-value database and it is perfect when working with risk data.

As this model is so simple it can perform well in many scenarios but it will generally be unsuitable in the following cases:

- If having relationships between different sets of data is needed, even though some key-value stores provides traversing links.
- When having multioperation transactions and when an operation fails you want to revert the rest of the operations.
- If you need to search the keys using data from the value part.
- To operate over multiple keys at a time.

1.4.1.2 Document-based databases

(a) *MongoDB* (b) *CouchDB*

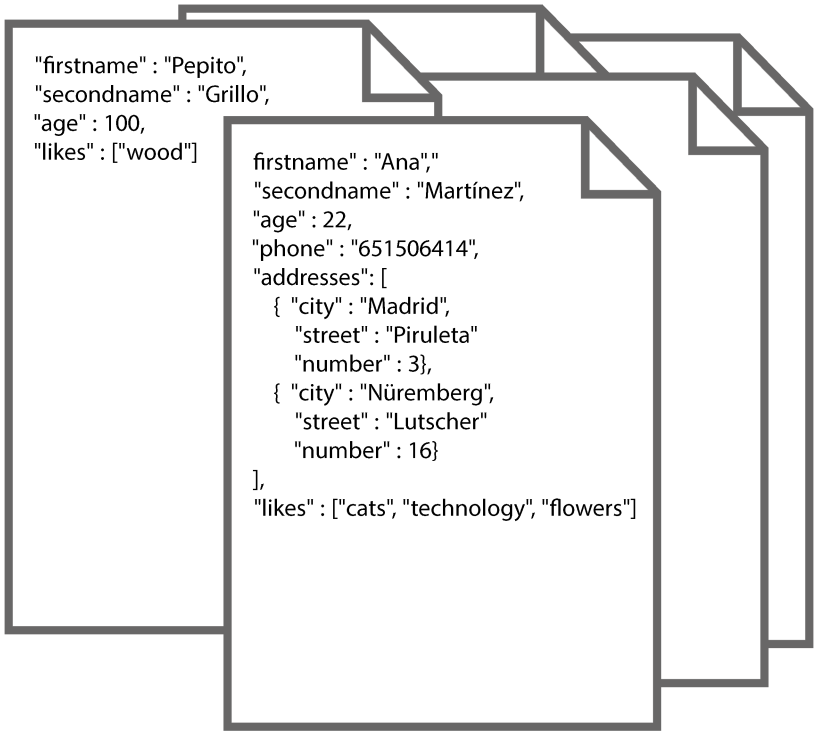
Figure 1.3: Document databases most popular implementations

Document-oriented databases are a subclass of the key-value databases, which implies that key-value databases are more general. In document-based databases each record is a document, typically XML, JSON or BSON. These documents are self-describing, hierarchical tree data structures. Moreover, documents are independent units which improve performance, as related data is read contiguously and allow us to distribute data across multiple servers while keeping related data together.

Documents have differences in their attributes and any data can be stored in them (there are specific restrictions depending on the type of document database), so that unstructured data can be stored easily. Similar documents belong to the same collection. A collection is analogous to an SQL table, but in relational databases each columns stores the same type of values or `null`, wasting memory and allowing less flexibility. The figure 1.4 shows an example of the differences between a table with persons in a typical relational database and the collection persons in a document database. Documents are closer to what we have in our programs than relational schemas. Note that documents can be as complex as you choose: nested data can be

firstname	secondname	age	phone	city	address	like1	like2	like3
"Ana"	"Martínez"	22	"651506414"	"Madrid"	"C/ Piruleta, 3"	"cats"	"technology"	"flowers"
"Pepito"	"Grillo"	100	null	null	null	"wood"	null	null

(a) *Relational database - Persons table*



(b) *Document database - Persons collection*

Figure 1.4: Person schema

used to provide additional sub-categories of information about an object. You can also use one or more document to represent a real-world object.

The strength of a document database's query language is an important differentiator of these databases. We can query the data inside the document, while in key-value databases we would have to seach the document by its key, get the whole document and then look into the document. Because of that, document stores provides as a balance between flexibility and similarity to relational databases, which I am used to work with.

We are going to analyse *MongoDB* (figure 1.3a), as it is a representative implementation of document databases and *CouchDB* (figure 1.3b) which is used in some medical projects. Both of them are free and open-source, there are some private options such as *Azure DocumentDB*, *Lotus Notes* or *SimpleDB*) but expense is not worthwhile and we can not afford it.

MongoDB uses JSON documents to manage the data, although internally it saves BSON documents, which is a JSON-like format more compact than JSON that improves efficiency. Documents are limited to 16MB. *MongoDB* configures consistency and improves availability by using replica sets. It allows atomic transactions at the single-document level and it can also be scaled easily. All this features of *MongoDB*are detailed in section 2.1.

As we have said above, *CouchDB* has been used in some medical projects, although things like x-ray images or electroencephalography wave recordings are saved and it is quite doubtful that those kind of projects are similar to ours. Let's mention two interesting examples: [33] discuss the use of *CouchDB* for Document-based Storage of DICOM Objects and [31] explains why using Document-Based Databases for Medical Information Systems in Unreliable Environments is well-suited.

1.4.1.3 Conclusion

Both key-value and document-oriented databases are appropriate for web and real-time analytics and provides us with schema flexibility, although document stores query engine is a determinant factor. Moreover, the fact that they are closer to relational databases will makes easier the transition from SQL and understanding the database and its design. Also, the popularity of *MongoDB* provides us with a large amount of documentation and manuals, which make my learning much easier. All in all, I consider *MongoDB* to be the best option to this project.

1.4.2 Streaming processing with Spark

Volume, Velocity and Variety are the three properties which define *Big data*, and that is why we usually talk about the 3Vs (figure 1.5). Streaming processing is the solution when we want not only to process big volumes of varied data but to do it fast too. It processes data in real time and focuses on unlimited and continuous data flow. Although data revolution has just started, stream processing has already demonstrated to have a great future and it will surely become very important for most companies in the close future.

We are going to focus interest on streaming analytics, although some of the technologies mentioned can be used for other purposes. There are many different frameworks (such as *Apache Storm*, *Apache Spark* or *Apache Samza*) and products (such as *IBM InfoSphere Streams* or *TIBCO StreamBase*) available on the market. As all of them are very similar we are going to concentrate on *Apache Spark*, because it is one of the most popular solutions and as it is free software we can use it with no cost.

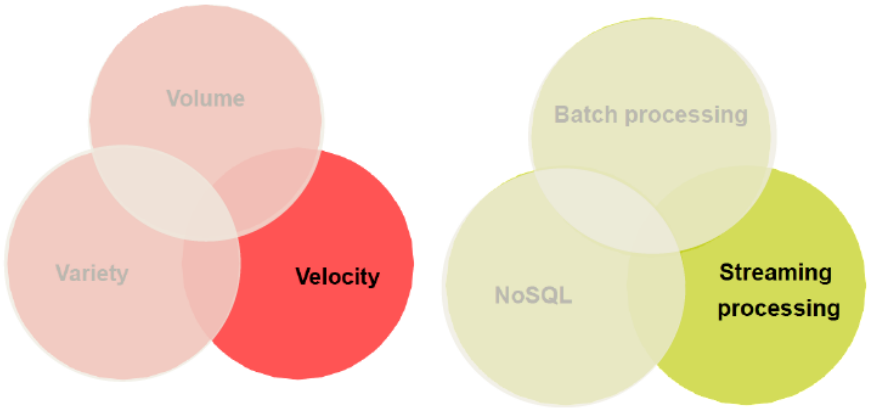


Figure 1.5: 3Vs

Apache Spark is a fast and general cluster computing framework originally started by Matei Zaharia as part of his thesis at *UC Berkeley AMPLab* in 2009 and open sourced in 2010. In 2003, it was donated to the Apache Software Foundation where it remains today. *Databricks*, which is a startup offering commercial support for Spark, was created in 2004.

Spark was developed as an alternative to *Hadoop*'s two-stage disk-based *MapReduce* paradigm shown in figure 1.6. *Hadoop* is slow because of replication, serialization and disk I/O, so that it is inefficient for iterative algorithms (most of Maching Learning Algorithms are iterative) and interactive data mining.

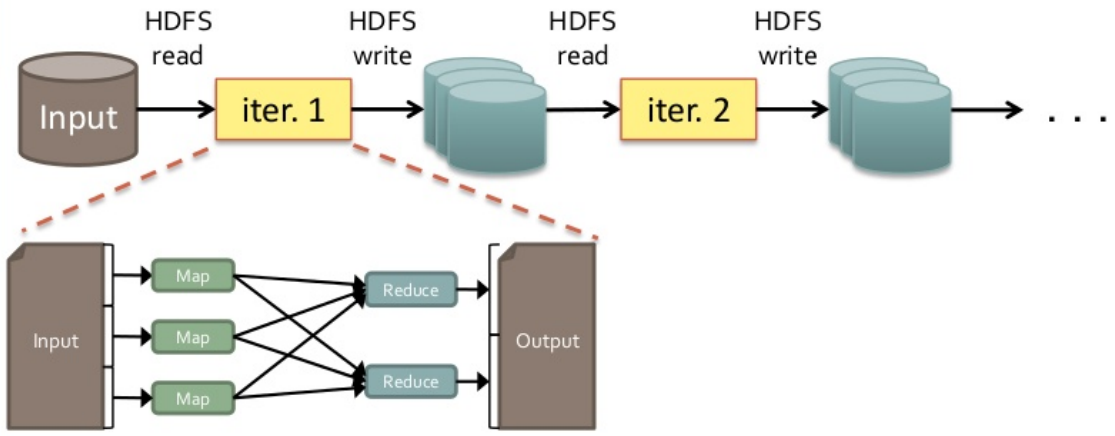


Figure 1.6: Data sharing in Hadhoop

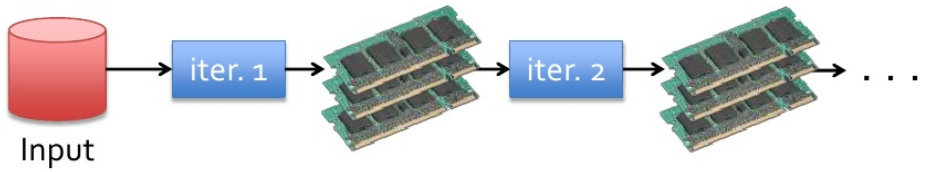


Figure 1.7: Data sharing in Spark

Spark's multi-stage in-memory paradigm provides performance up to 100 times faster than *Hadoop* MapReduce in memory (see figure 1.8), or 10 time faster on disk. That is achieved by allowing to access to the same dataset repeatedly without going to disk. 1.7.

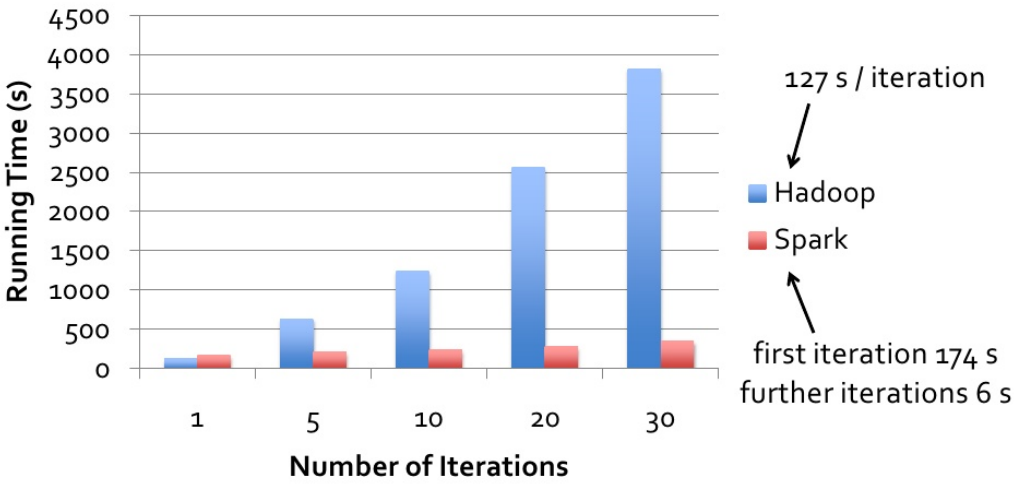


Figure 1.8: Hadhoop vs Spark

Spark revolves around the concept of Resilient Distributed Datasets (RDD), which is a collection of elements that can be operated on in parallel. They can work with any kind of data, which is another plus point. All necessary transformations are done over the RDDs, which contain not only the data but also the transformation operations that must be done with them. Each RDD is divided in multiple partitions which can be executed in different nodes of the cluster (it also stores information about partitions and the node that it should go to). Moreover, they are immutable: once data have been read it is assumed that it is not going to change, so it is not read again. When transformations are applied another RDD is created with the result of these transformations. As a result of being immutable and containing information about transformation flow, partitions and nodes, RDDs are fault-tolerant.

RDDs are also efficient as they use lazy evaluation: Transformations are defined but they are not executed until an action runs it. That allows to plan and optimize when the operations are done and avoids unnecessary exchange of data between machines. *Spark* allows a RDD to remain in memory to avoid recalculating it every time it is needed again.

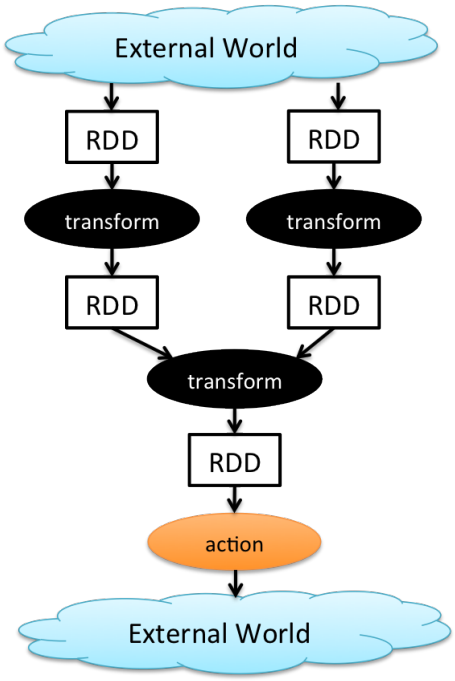


Figure 1.9: Spark process

To sum up, the process is illustrated in figure 1.9. First a RDD is created, transformations are applied (when necessary according to lazy evaluation) and then actions, obtaining a value. During all the process everything is in RAM memory and at the end data is saved.

Another important advantage of *Spark* is that it offers more operations apart from Map and Reduce and allows more flexibility when defining operations. That is why parallel applications can be built using *Spark* quickly and easily, allowing more complex *Big Data* problems being solved. Some of the most common transformations and actions can be seen in table 1.1. It also support writting applications in Java, Scala, Python and R.

Spark applications run as independent sets of processes on a cluster. There is a driver program, which is a process running the main function of the application and creating the SparkContext. It also declares how RDDs are created and what transformations and actions are done over them. The SparkContext connects to a cluster manager and then Spark acquires executors on nodes. After that, the

TRANSFORMATIONS
map(func)
filter(func)
mapPartitions(func)
union(otherDataset)
intersection(otherDataset)
distinct([numTasks]))
groupByKey([numTasks])
reduceByKey(func, [numTasks])
sortByKey([ascending], [numTasks])
join(otherDataset, [numTasks])
cogroup(otherDataset, [numTasks])
repartition(numPartitions)
...

ACTIONS
reduce(func)
collect()
count()
first()
take(n)
takeSample(withReplacement, num, [seed])
takeOrdered(n, [ordering])
saveAsTextFile(path)
saveAsSequenceFile(path)
saveAsObjectFile(path)
countByKey()
foreach(func)
...

Table 1.1: RDDs transformations and actions

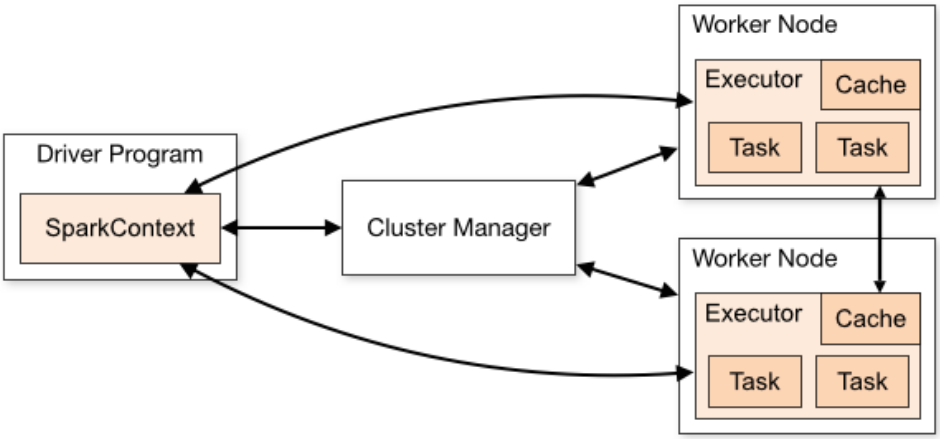


Figure 1.10: Spark Engine

application code is sent to the executors and the SparkContext sends tasks to them. Figure 1.10 is diagram of Spark Engine which shows its components and the way they interact.

Spark ecosystem is another important advantage. As it can be seen in figure 1.11, it includes Spark SQL, Spark Streaming, Mllib and GraphX. Spark SQL lets you query structured data inside RDD. Spark streaming is the stream processing engine. MLlib is Apache Spark's scalable machine learning library. Lastly, GraphX is Apache Spark's API for graphs and graph-parallel computation.

We have already analysed in detail how Spark works and all the advantages that it provides: it is incredibly fast (specially for iterative algorithms), fault-tolerant and

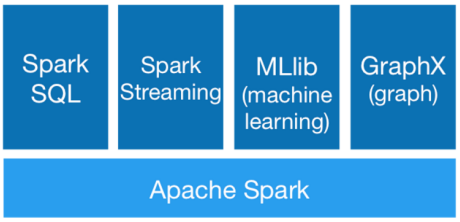


Figure 1.11: Spark ecosystem

efficient and it provides ease of use (offering a wide range of operations and allowing to write in several programming languages) and generality (with its ecosystem). Its only disadvantage is that it a quite new technology and it has not been tested in clusters with a lot of nodes. We are now going to concentrate on how Spark Streaming works.

Figure 1.12: Spark streaming flow

Spark Streaming discretizes the streaming data into tiny, sub-second micro-batches and with each batch of data is built a RDD. This allows the streaming data to be processed with the Spark engine explined above. This process can be seen in figure 1.12. Dividing the data into small micro-batches allow us to schedule the tasks more efficiently, while in the traditional record-at-a-time approach the static scheduling may cause that some nodes become a bottleneck and slow down the pipeline. In Spark Streaming, the tasks are balanced across the nodes, some of them run a few longer tasks while others run more of the shorter tasks. The difference is illustrated in figure 1.13

Spark Streaming provides a high-level abstraction called DStream, which represents a buffer with a sequence of RDDs. Tranformation can also be used with DStream, provinding ease of use. The most common transformations can be seen in table 1.2. The transformation *transform* is particularly interesting as it allow applying a RDD-to-RDD function to every RDD of the source DStream.

Last but not least, we are going to explain why Spark Streaming is fault-tolerant. As all data transformation are based on RDDs operations and we have already explained that they are fault-tolerant, if the input dataset is present, all data can be recomputed. It tolerates the failure of the driver node by saving the state of the DStreams periodically to a HDFS file, which can be used to restart the streaming computation. It also tolerates the failure of a node. When the input source is a HDFS

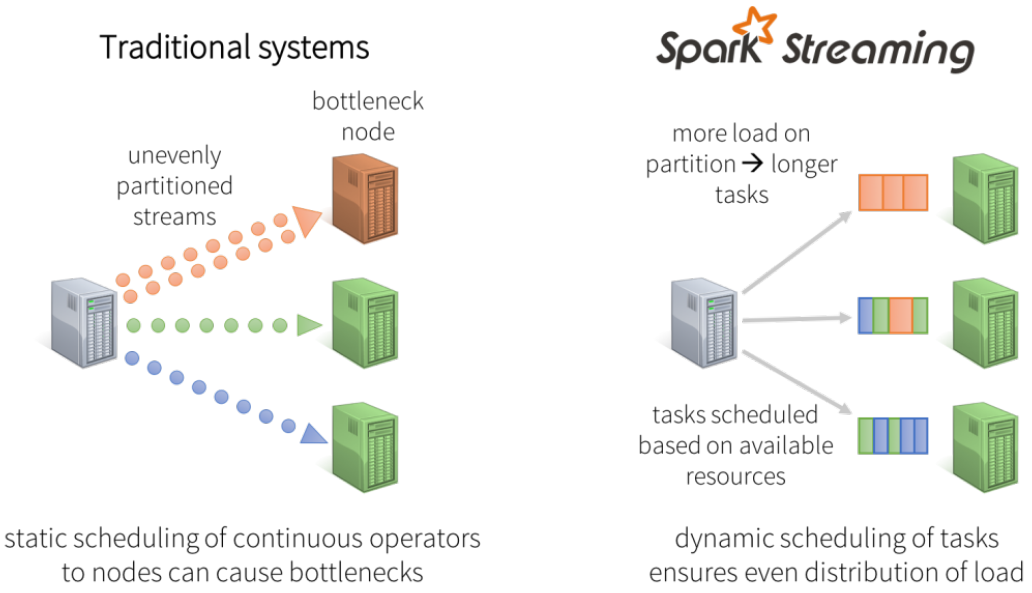


Figure 1.13: Spark dynamic load balancing

TRANSFORMATIONS
map(func)
flatMap(func)
filter(func)
repartition(numPartitions)
union(otherStream)
count()
reduce(func)
countByValue()
reduceByKey(func, [numTasks])
join(otherStream, [numTasks])
cogroup(otherStream, [numTasks])
transform(func)
updateStateByKey(func)
...

Table 1.2: DStreams transformations

data can be recomputed. On the other hand, when the input source receives data through a network, data is replicated in memory between nodes of the cluster. The only problem is when the node which fails is the one where the the network receiver was running, as the data that has not been replicated will be lost.

Not only does Spark allows to recover from failure, but it also do it faster that the traditional approach. In Spark, there are small tasks that can be run anywhere. That allows that failed tasks can be rerun in parallel on any other nodes, as it is illustrated

in figure 1.14.

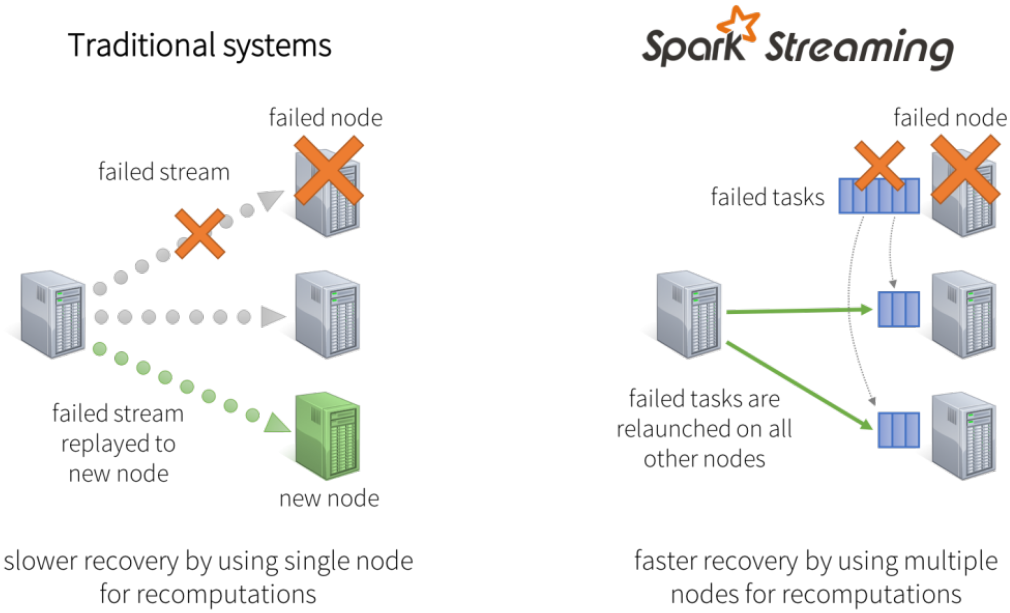


Figure 1.14: Spark process

The main bibliographic references used for this section are [8], [1] and [3]. The images are also taken from these sources, except the figures 1.8 and 1.6, which I took from [18] and [2] respectively.

1.4.3 Neural networks

The aim of this section is to follow part of the chapter 5 of [21], commenting on some details. I have also added information from the the videos I saw at [36]. At the end I will also explain the importance of *Neural Networks* on *Big Data*.

Artificial neural networks are a machine learning technique based on how your neurons in your brain work and they have been particularly successful recently in big and hard problems. It is a supervised method, as we give it inputs and outputs to train it, and it can be used for regression if the outputs are continuous and classification otherwise. They model complex and nonlinear patterns that simpler models may miss. For example, we could predict if a person has a bipolar crisis base on how many hours the person slept the day before and how concentrate he feel he is, by using a classification neural network.

Let's start by defining the basic neural network model. Given the input variables $x_1, \ldots, x_D$, we construct M linear combinations:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{1.1}$$

With $j = 1, \ldots, M$. The superscript (1) indicates that we are in the fist *layer* of the network. The parameters $w_{ji}^{(1)}$ and $w_{j0}^{(1)}$ are called *weights* and *biases* respectively and the quantities a_j are called *activations*. Using a differentiable nonlinar function h_1, which is called *activation function*, we get what is called *hidden units*:

$$z_j = h_1(a_j) \tag{1.2}$$

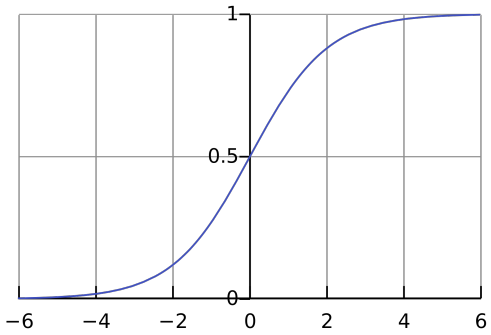


Figure 1.15: Logistic sigmoid

The activation function is generally a sigmoid function, which is a function which has a S shaped curve. The most famous example is the *logistic sigmoid*, which you can see in figure 1.15 and whose equation is:

$$h_2(a) = \frac{1}{1 + e^{-a}} \tag{1.3}$$

Linearly combining the hidden units, we get *output unit activations*, which corresponds to the second layer of the network:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{1.4}$$

With $k = 1, \ldots, K$ being K the number of outputs. As in the first layer, w_{k0}^2 are *bias parameters*. Again, using another *activation function* we get a set of network

outputs y_k.

We can now combine all the stages to give the overall network function:

$$y_k = h_2 \left(\sum_{j=1}^{M} w_{kj}^{(2)} h_1 \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{1.5}$$

Where w is the vector of *weight* and *bias* parameters. Also, the process of evaluating this formula can be seen as a *forward propagation* of information through the network, as it can be seen in the illustrative figure 1.16.

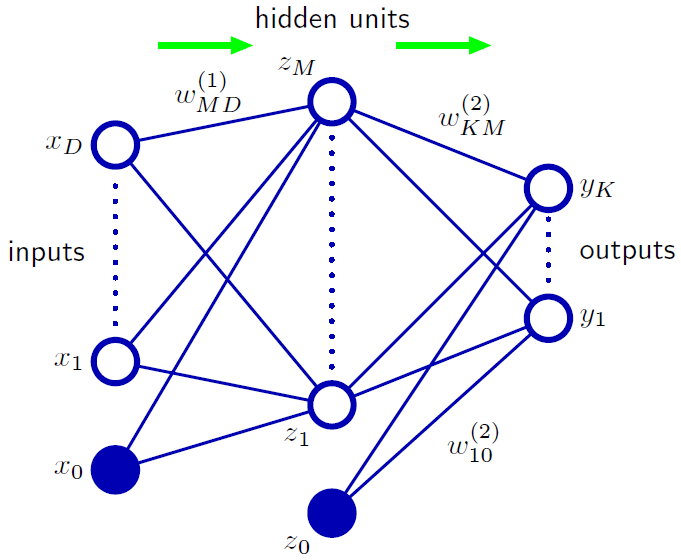


Figure 1.16: Neural networks

To simplified the formula 1.5 we can add an additional input variable x_0 whose value is 1. That way, the formula 1.1 becomes:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{1.6}$$

And the formula 1.5 turns into:

$$y_k = h_2 \left(\sum_{j=0}^{M} w_{kj}^{(2)} h_1 \left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right) \tag{1.7}$$

Note that the neural network in figure 1.16 can be easily generalized to an arbitrary number of layers. Those networks with many layers are known as deep neural networks and are used in deep learning. Although the one in 1.16, which is known as two-layer-network or single-hidden-layer network, is the most widely used in practice.

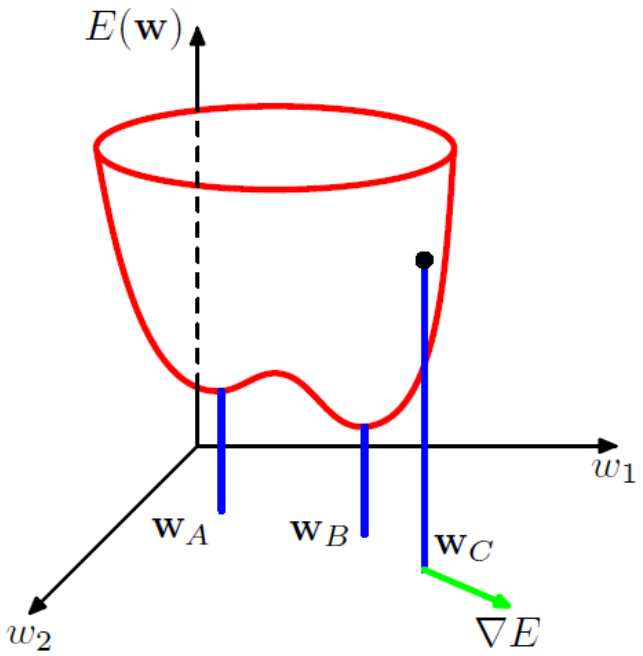


Figure 1.17: Error function as a surface in the weight space

As the layers are fixed, the structure and behaviour is fixed too, but the result is being updated as we train the neural network. Training a network means minimizing the error function and as we do not have much control over data, we will minimize the cost by changing the weights. Because of that the error function is of the form $E(w)$ and it must be differentiable. The error function can be viewed as a surface in the weight space, as in figure 1.17. If we make a small step in the weight space from w to $w + \delta w$, then the change in the error function is $\delta E \simeq \delta w^T \nabla E(w)$. We can see that doing the Taylor expansion of $E(w)$ and $E(w + \delta w)$ centred in 0:

$$E(w) \simeq E(0) + w^T * \nabla E(w) \tag{1.8}$$

$$E(w + \delta w) \simeq E(0) + (w^T + \delta w^T) * \nabla E(w) \tag{1.9}$$

And subtracting 1.8 to 1.9 we get $\delta E = E(w + \delta w) - E(w) \simeq \delta w^T \nabla E(w)$.

As E is differentiable in w, the derivative with respect to the vector v, $D_vE(w)$, is the component of the gradient in the direction of v:

$$D_vE(w) = \|\nabla E(w)\| * \|v\| * cos\theta \tag{1.10}$$

where θ is the angle formed by $\nabla E(w)$ and v. If we take v with $\|v\| = 1$ then:

$$D_vE(w) = \|\nabla E(w)\| * cos\theta \tag{1.11}$$

Consequently, $D_vE(w)$ is maximum when $cos\theta = 1$, or equivalent, when v has the direction of the gradient and this maximum value is $\|\nabla E(w)\|$. So, $\nabla E(w)$ points in the direction of greatest rate of increase of the error function. Because of being differentiable, $E(w)$ is a smooth continuous function of $E(w)$ and then its smallest value will occur when $\nabla E(w) = 0$. Otherwise we could make a small step in the direction if $-\nabla E(w)$, reducing the error value.

If E is a non-convex function (as in figure 1.17), that is that the function goes up and then down, our neural network may get stuck in a local minimum. For a successful application of neural networks we do not necessarily have to to find the global minimum, but then we need to compare several local minimums to find a sufficiently good solution.

We do not expect to find an analytical solution to the equation $\nabla E(w) = 0$, instead we use iterative numerical procedures. There are many techniques to optimize continuous nonlinear functions, most of them choose an initial value w^0 for the weight vector and then move through weight space in every step. So, in the r step:

$$w^{(r+1)} = w^{(r)} + \Delta w^{(r)} \tag{1.12}$$

We can use the gradient of the error function to achieve a reduction in the number of evaluations, which turn into an improvement in speed, specially in higher dimensions. For example when using the *local quadratic approximation* the order of finding the minimum is $O(W^3)$, where W is the total number of weights and bias in the network, and by using the gradient information it can be reduced to be $O(W^2)$. The simplest way to use the gradient to find the minimum value of the error function is to take steps downhill (in the direction of negative gradient) and stopping when the

cost stop getting smaller. Then, the formula 1.12 takes the form:

$$w^{(r+1)} = w^{(r)} - \eta \nabla E(w^{(r)}) \tag{1.13}$$

Where $\eta > 0$ is called *learning rate*. In every step the gradient is re-evaluated an the processed repeated. This method is called *gradient descent*. Then, to find a sufficiently good minimum, we should run the gradient descent algorithm several times with different starting points chosen randomly. After that we will need to compare the results using and independent validation set.

Methods like gradient descent, where the whole data set is used at once, are called *batch* methods. There are some variations of gradient descent, such as *conjugate gradients* and *quasi-Newton* methods, which are more efficient for batch optimization than simple gradient descent. On the other hand, the method called *on-line gradient descent* or *stochastic gradient descent* updates the weight vector based on one data point at a time. The point can be selected in sequence or randomly. There are also intermediate scenarios where weight vector is updated based on batches of data points.

Stochastic gradient descent has been proved to be useful in practice for training neural networks on large datasets. In that case the error function is:

$$E(w) = \sum_{n=1}^{N} E_n(w) \tag{1.14}$$

Where N is the number of inputs. And the formula 1.12 takes the form:

$$w^{(r+1)} = w^{(r)} - \eta \nabla E_n(w^{(r)}) \tag{1.15}$$

Stochastic gradient descent method has several advantages. The most important is that it allows to space from local minimums, as ∇E_n will not normally be 0 for every n. It also handle redundancy in the data more efficiently. In batch methods if the input set is repeated n times, also is the error factor multiplied by a factor of n, so we get an equivalent error function which is less efficient to calculate. However, the stochastic gradient descent method is unaffected.

As we have previously commented, artificial neural networks have recently become very popular because of their success in a lot of big and hard problems, and *Big Data*

field is not an exception, as it is explained in [12]. Most machine learning algorithms has problems to scale up to big data. In addition, high dimensionality, velocity and variety are a challenge for this algorithms. Using neural networks with methods such as stochastic gradient descent which learn in an on-line, incremental mode without requiring in-memory access to huge amounts of data, are less vulnerable to the size of the data. They also can take advantage of massively parallel computations (as the train of every input set can be done independently), which use very simple processors and that can not be used in other machine learning technologies.

2 MongoDB database

2.1 Features

In this section we are going to comment on the features of *MongoDB*. I have followed the *Document Databases* chapter in [34], as it is really complete and the concepts are really well explained.

2.1.1 Queries

We have already mentioned that the strength of document databases' query language is an important differentiator of these databases and *MongoDB* is not an exception. In *MongoDB* documents are indexed using a BTree and queried using a *JavaScript* query engine. It has a query language which is expressed via JSON. All queries address a single collection and indexes allows efficient queries (see 2.3.2.3).

Let's give an example of how queries are made in *MongoDB* in comparison with SQL. Suppose that we want to query for all users whose house is in a Spanish city. In relational databases we would probably have three collections, one for users, another one with houses and the last one with information about cities. Then, the SQL would be:

```
SELECT * FROM users, houses, cities
WHERE
houses.ownerId = users.id
AND houses.citiesId = cities.id
AND cities.contryName = "Spain"
```

And the equivalent *Mongo* query would be:

```
db.orders.find({"users.house.contryName" : "Spain"})
```

The query for *MongoDB* is simpler because the objects are embedded inside a single document and the information is usually repeated in several document (the country name for every house in the same city). There are more examples of *MongoDB* queries in the section 2.3.3.

2.1.2 Transactions

Transactions in relational databases represent any change in the database. So when a transactions begins, the database can be modified using commands over different tables such as *insert*, *update* or *delete* and then it can be decided if the changes are kept (*commit*) or not (*rollback*). This is not generally available in NoSQL databases, a write can only succeed or fail. Transactions in *MongoDB* are at the single-document level (*atomic transactions*): one or more fields of the document (including sub-documents and elements of an array) are updated and then the document is updated in the database. Transactions involving more than one operation are not possible, although there are other document databases, like *RavenDB*, that support them. In fact, those multiple operation transaction are not needed in our project. For example, they would be needed if we had duplicated data in several collections, but as it is explained in 2.3.2 it is not the case in the database designed for the project.

2.1.3 Availability

MongoDB uses the master-slave set-up and maintains several copies of data called *replica sets* using native replication. The *replica set* is used to improve data availability, by increasing the number of nodes and allowing reading from slaves. The nodes elect the master (priorities can be given to favour some nodes) by themselves. The master is who has all the data updated and receive all requests, and then data is replicated to the slave nodes. If the master node goes down, the remaining nodes elect a new master and it will serve all future requests. When the node that failed gets on-line again, it joins in as a slave and get the updated data from the current master. In our case we currently have one server so we do not have the chance to use *replica set*. Although, that is not an availability problem at the moment as the traffic in our server is really low and in the future we could add more nodes easily as it is explained in section scaling.

2.1.4 Consistency

Consistency in *MongoDB* database is also configured by using *replica sets* (multiple copies of data) and waiting for the writes to be propagated to a given number of slaves before they success. This number can be increased for stronger consistency but it will affect write performance, as the writes will have to be propagated to more nodes.

The number of slaves the writes have to be propagated to and if reading from slaves is allowed can be configured in the connection, database, collection or individually for each operation and the specific characteristics of the project need to be taking into account to choose those settings. In our project we have set them when creating users (see Appendix: Creation of users and roles) by setting the `w` attribute to `majority`. Currently there is only one node, the server I configured in section 2.3.1.2, so writes

will return immediately. If we add more servers in the future, it will wait for the write operation to have propagated to the majority of voting nodes before it returns as successful. Until we add more servers there is no option to read from slaves.

2.1.5 Scaling

Scaling is a very important issue in our project as we have a single node at the moment and we plan to add more nodes in the future. *MongoDB* allows horizontal scale easily with no downtime and without changing your application. To allow heavy-read we can add more read slaves and read directly from them. When a new node is added, the process will be the same as the one explained in the availability section when the master node gets on-line after having gone down: it get the data from the existing nodes, join the *replica set* as secondary node and it is ready to serve read requests.

When we want to horizontally scale for writes, we use *sharding*. It is a technique similar to partitions in relational databases as the data is also split by a concrete field, but then it is moved to a different node. *MongoDB* automatically balances the data by dynamically moving it between nodes. So, to scale for writes we just have to add more nodes to the cluster and increase the number of writeable nodes. Moreover, each shard can be a *replica set*, ensuring better read performance within the shard.

2.2 Learning MongoDB

It was the first time I worked with NoSQL databases and with *MongoDB* in particular. First of all, I took almost the whole *MongoDB for Java Developers* course ([10]), although the course didn't start until 24 May 2016, so I saw the videos of previous editions of the course at *Youtube*. There is also a subject of Systems for management of data and information (*Sistemas de Gestión de Datos y de la Información*) in the Master's of the Computer Science Faculty where *MongoDB* is studied. I asked some students in this class to send me the slides they were using ([22]), which I found really useful as they have a lot in-depth information and illustrative examples. I also used the complete official guide of *MongoDB*, see [11]. For more specific topics or doubts I used other guides, blogs and forums such as [14].

2.3 Database for the project

2.3.1 Installation and configuration

2.3.1.1 Development

I used *Windows 7* for testing before moving to the production environment. The last version I used was the 3.2.6, which is the last stable release. But I also worked with other versions as the 3.2.6 was launch on Apr 28, 2016. I download *MongoDB 2.3.6 MongoDB Community Edition* from http://www.mongodb.org and I followed the installation guide in the section *Install MongoDB Community Edition on Windows* of [11].

I created the configuration file in my computer in the following route `C:\Program Files\MongoDB\mongod.cfg`. The configuration file can be seen in the Appendix: MongoDB configuration file. The data is stored in `C:\data\db` and the log file can be found in `C:\data\log\mongod.log` in my computer.

After the installation I configured a *Windows Service* for *MongoDB Community Edition*. I ran this command in the *cmd* window with admin permissions to install the service:

```
"C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe"
   --config "C:\Program Files\MongoDB\mongod.cfg" --install
```

The following command can be ran to uninstall the service:

```
"C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe" --remove
```

To start, restart and stop the server I used these commands:

```
net start MongoDB
net restart MongoDB
net stop MongoDB
```

The *Mongo shell JavaScript* client can be ran from the *cmd* window:

```
"C:\Program Files\MongoDB\Server\3.2\bin\mongo.exe"
```

Or, after enabling authentication with the admin user (see 2.3.2.2):

```
"C:\Program Files\MongoDB\Server\3.2\bin\mongo.exe" -u "admin"
   -p "password5" --authenticationDatabase "bipolarDatabase"
```

2.3.1.2 Production

Debian 7.1 is one of the OS recommended in the *Production Notes* of [11]. In addition, for *MongoDB 3.2*, 32-bit binaries are deprecated and will be unavailable in future releases. So *Debian 7.1* 64-bit was installed in the Server which is at the Department *Seminario ACyA* of the Computer Science Faculty, whose static global IP address is `147.96.25.33`.

As in development, I installed the version 3.2.6 of *MongoDB Community Edition*. I followed the installation guide in the section *Install MongoDB on Debian* of [11] but I had a problem during the installation:

```
E: Unable to locate package mongodb-org-shell
```

I managed to solved it by installing every `.deb` from [5].
The configuration file is in `/etc/mongod.conf`. I eliminated the line where the bind IP is declared as it is the loopback address by default and by eliminating it I made the server listen in all IPv4 addresses on the local machine. This include the global IPv4 which is the one the external clients are going to use to communicate with the server. I also modified the configuration file to configure authentication, as it is explained in 2.3.1.3.

The log file is save by default in `/var/log/mongodb/mongod.log` and the database data is storaged in `/var/lib/mongodb`.

To start, restart and stop the server the following commands can be used:

```
sudo service mongod start
sudo service mongod restart
sudo service mongod stop
```

After installing and configuring the server, it was needed to open the port 27017 (*MongoDB* port by default). I used `iptables` command to do so:

```
sudo iptables -A INPUT -p tcp --dport 27017 -j ACCEPT
sudo iptables -A OUTPUT -p tcp --sport 27017 -j ACCEPT
sudo iptables-save
```

To check that everything is working properly:

```
sudo netstat -lnp  grep mongo
```

But surprisingly, after all that, connecting to the server from outside was not possible because the UCM net does not allow connections to some ports and the

27017 is one of them. It would have worked in other net but not at UCM one. So, I tried with other ports until I found a port which works, the 8080. So I changed the port in the configuration file (the configuration file can be seen in the Appendix: MongoDB configuration file) and from that moment on I used the 8080 port.

In the server, the *Mongo shell JavaScript* client can be ran from the terminal window:

```
mongo
```

Or, after enabling authentication with the admin user (see 2.3.2.2):

```
mongo -u "admin" -p "password5"
   --authenticationDatabase "bipolarDatabase"
```

I also managed the server database from my house using Windows *cmd*, connecting with the admin user to the server global IP and the port 8080. I used the following command (it is necessary to include the port, it won't work otherwise):

```
"C:\Program Files\MongoDB\Server\3.2\bin\mongo.exe" --host
   147.96.25.33 --port 8080 -u "admin" -p "password5"
   --authenticationDatabase "bipolarDatabase"
```

2.3.1.3 Authentication

In both development and production environment I configured authentication that, as we will explain in 2.3.2.2, it is crucial for the veracity and validity of the data stored in our database. Before creating any role I had to change the authentication mechanism to the previous one (*MONGODB-CR*) instead the default new one in releases 3.* (*SCRAM-SHA-1*). I had to do that because we planned to access the *MongoDB* database from the Android application and when using *SCRAM-SHA-1* from the *Java Driver* `java.nio.channels.AsynchronousSocketChannel` is called and it does not exist in Android, so it causes an exception and as a result it does not work. So after installing and starting the server I connected with a client without authentication and I changed the authentication mechanism in *MongoDB* following the following steps:

```
use admin

var schema = db.system.version.findOne({"_id" : "authSchema"})

schema.currentVersion = 3

db.system.version.save(schema)
```

After that we create the roles specified in 2.3.2.2 as it is described in Appendix: Creation of users and roles. Then I modified the configuration file to add authetication (`authorization: enabled`). The final configuration file can be seen in the apendice Appendix: MongoDB configuration file.

Then I restarted the *MongoDB* server to enable authentication and from that moment on I connected to the database with a specific user.

2.3.2 Design

The design of the database includes the design of collections, the creation of roles and users and the creation of indixes.

2.3.2.1 Colletions

In document databases there are not rows as there are in relational databases, so we can not restrict what it is saved in the database. However, it is necessary to know what other applications are saving in the database to be able to read the information we need and if some applications are saving the same data all of them should do it in the same way.

I have called our database **bipolarDatabase** and the collections that I have defined for our projects are the followings. The name of the collection and attributes are in bold and the BSON data type of each attribute is also specified. Information about BSON types can be found in the section *BSON Types* of [11].

collection **users**: Stores the users data.

_id	**ObjectId** *unique, compulsory*
	Identifier.
email	**String** *unique, compulsory*
	Email address.
name	**String** *compulsory*
	Full name.
birthDate	**Date** *compulsory*
	Birth date.
gender	**Boolean**
	Gender. Values: true (woman) and false (man).
pin	**32-bit integer**
	6-digits password to be use in the mobile app.

cohabitation **32-bit integer**

Who the user lives with. Values: 0 (alone), 1 (with his/her couple), 2 (with his/her couple and children), 3 (with his/her parents) and 4 (others).

diagnosis **String**

ICD-10 diagnosis.

diagnosisAge **32-bit integer**

Age at which the disorder was diagnosed.

senLit **Boolean**

Whether the patient is sensible to Lithium or not.

senVal **Boolean**

Whether the patient is sensible to Valproate or not.

senCar **Boolean**

Whether the patient is sensible to Carbamazepine or not.

seasonality **Boolean**

Whether the patient presents seasonality or not.

maniaCrises **32-bit integer**

Frecuency of mania crises.

mixedCrises **32-bit integer**

Frecuency of mixed crises.

freePeriod **32-bit integer**

Number of months where the patient presents no symptoms.

psycSymp **Boolean**

Whether the patient presents psychotic symptoms or no.

others **String**

Other diagnosis.

collection **comments**: Stores the prescriptions and the notifications to be send to the mobile app.

_id **ObjectId** *unique, compulsory*

Identifier.

user_id **ObjectId** *unique, compulsory*

User identifier.

prescription **Boolean** *compulsory*

Whether the document represents a prescription or a message.

dateStart **Date** *compulsory*

Start date and time.

dateEnd **Date**

End date and time. If it is empty the end date is indefinite.

time **32-bit integer** *compulsory*

Time at which the alarm has to be repeated or the medicine taken from the start to the end date. Minutes from 00:00.

name **String**

Active ingredient or commercial name.

type **32-bit integer**

Medicine type. Values: 0 (Lithium), 1 (anticonvulsant), 2 (antipsychotic), 3 (anxiolytic/hypnotic), 4 (antidepressant) and 5 (others).

title **String**

Official text.

text **String**

Alarm text.

dose **32-bit integer**

Dose.

collection **records**: Stores the medical records.

_id **ObjectId** *unique, compulsory*

Identifier.

user_id **ObjectId** *unique, compulsory*

User identifier.

date **Date** *compulsory*

Date and time at which the record was saved.

eeag **32-bit integer**

Rating in the GAF scale (see [7]).

hdrs **Document**

The document contains the following keys, which represents the HDRS questions (see [9]), each with a 32-bit integer value associated: **deprMood**, **feelGuilty**, **suic**, **insoEarly**, **insoMiddle**, **insoLate**, **workActiv**, **retard**, **agitat**, **anxiPsych**, **anxiSomat**,

somSymptGastr, **somSymptGener**, **geniSympt**, **hypochon**, **lossWeight** and **insight**.

ymrs **Document**

The document contains the following keys, which represents the YMRS questions (see [15]), each with a 32-bit integer value associated: **elevMood**, **incrActEner**, **sexuInters**, **sleep**, **irritab**, **speech**, **lanThoughDis**, **content**, **disAggrBehav**, **appearan**, **insight**.

panss **Document**

Document which can contains three documents with every part of the PANSS interview (see [20]). Their keys are: **panssPos**, **panssNeg** and **panssPg**.

panssPos contains the following keys, which represents the items of the Positive scale part of the PANSS interview, each with a 32-bit integer value associated: **delsus**, **concepDisor**, **halluBehav**, **excitem**, **grandios**, **suspc** and **hostil**.

panssNeg contains the following keys, which represents the items of the Negative scale part of the PANSS interview, each with a 32-bit integer value associated: **blunAffec**, **emotWithd**, **pootRapor**, **passSocWithd**, **diffAbstThin**, **lspontConvflow** and **steoThink**.

panssPg contains the following keys, which represents the items of the General Psychopathology scale part of the PANSS interview, each with a 32-bit integer value associated: **somaConcer**, **anxiet**, **guiltFeels**, **tension**, **mannPost**, **depress**, **motoRetar**, **uncoop**, **unuThough**, **disorient**, **poorAtten**, **ljudInsight**, **distVolit**, **pinpContr**, **preoc** and **asocAvoid**.

collection **analysis**: Stores the result of the analysis phase.

_id **ObjectId** *unique, compulsory*

Identifier.

user_id **ObjectId** *unique, compulsory*

User identifier.

date **Date** *compulsory*

Date and time at which the prediction was saved.

eeag **32-bit integer**

Prediction of EEAG (see [7]).

hdrs **32-bit integer**

Prediction of the sum of the answer's value in the HDRS test (see [9]).

ymrs **32-bit integer**

Prediction of the sum of the answer's value in the YMRS test (see [15]).

panss **32-bit integer**

Prediction of the sum of the answer's value in the PANSS interview (see [20]).

collection **mobileTests**: Stores the daily test that are done in the mobile app.

_id **ObjectId** *unique, compulsory*

Identifier.

user_id **ObjectId** *unique, compulsory*

User identifier.

date **Date** *compulsory*

Date and time at which the test is sent. There is only one test per day.

speedReaction **Document**

Document which can contains three 32-bit integer with which we measure the motor speed. Their keys are: **last** (the time to introduce the pin the last time - the one that was correct), **total** (total time since the users introduces the first number until he introduces the correct pin and clicks *start*) and **panssPg** (the number of tries, minimum one).

affectiveState **32-bit integer**

Affective state. Values: -3,-2,-1,0,1,2,3

motivation **32-bit integer**

Motivation. Values: -3, -2, -1, 0, 1, 2, 3

concentration **32-bit integer**

Concentration. Values: 1, 2, 3, 4, 5

anxiety **32-bit integer**

Anxiety. Values: 1, 2, 3, 4, 5

irritability **32-bit integer**

Irritability. Values: 1, 2, 3, 4, 5

fatigue **32-bit integer**

Fatigue. Values: 1, 2, 3, 4, 5

caffeine **32-bit integer**

Amount of caffeine.

alcohol **32-bit integer**

Amount of alcohol.

tobacco **32-bit integer**

Amount of tobacco.

drugs **32-bit integer**

Whether the user toke any drug different from alcohol and tobacco. Values: true (yes) and false (no).

timeBed **32-bit integer**

Time at which the user went to bed. Minutes from 00:00.

timeSleep **32-bit integer**

Time at which the user fell asleep. Minutes from 00:00.

timeWakeUp **32-bit integer**

Time at which the user woke up. Minutes from 00:00.

menstruation **32-bit integer**

Whether the user is menstruating. This question only makes sense when the user is a woman. Values: true (yes) and false (no).

collection **actigraphData**: Stores the data sent by the actigraph, specified in [16]. There is some information that do no appear in [16] and I obtained it from the files generated by the actigraph which you can see in Appendix: Actigraph data. The data that is is calculated by the actigraph based on other data that we are saving is not stored.

_id **ObjectId** *unique, compulsory*

Identifier.

user_id **ObjectId** *unique, compulsory*

User identifier.

date **Date** *compulsory*

Date and time at which the information was measured.

position **Array**

Array which contains three Doubles for x, y and z axis(in that order) measured in g-force. Range of x, y, z: -8 to 8. Resolution of x, y, z: 0.0039.

lux **32-bit integer**

Light level in lux. Range: 0 to 5000. Resolution: 5.

button **Boolean**

Whether the actigraph button is pressed. Values: true (pressed) and

false (not pressed).

temperature **Double**

Temperature in centigrades. Range: 0 to 70. Resolution: 0.1.

To design the database I have tried to follow the following principles:

- **Store together (in the same document) what it is going to be queried at the same time.** That is why for example the phenotype is store together with the user personal information in the collection `users`.

- **Avoid storing big documents.** Big documents make performance worst as they take up a lot and they also waste the bandwidth. In fact, *MongoDB* limits to 16MB the size of files. Taking that into account, we are storing mobile tests, comments, actigraph data and the analysis results, which are associated to a users, in separate document as we will have many of them and consequently the document which stores them will be really heavy. In this case, referencing is better than nesting.

- **Prioritize the storing phase over the analysis.** For example, we could have stored all mobile tests grouped them by `user_id` but then every time a test is introduced we should have found the `user_id` and, when having millions of users, this could be slow and the mobile apps should keep waiting until it finishes. On the other hand, storing data this way will make the analysis phase, which probably will need all the test of the same user, slower. In bipolar disorder, doctors usually detect a crisis one month later, so taking a little by longer in the analysis phase is not a problem. In addition, although we think that the analysis should be individual for every patient, as the illness varies a lot from one patient to another, we do not discard that analysing the data of different users we could find a relation unknown until now. Because of the difficulty for doctors to analyse hundred of patients at the same time and to compare their patients with healthy users this idea is not hare-brained.

- **Repetitions are not bad if we avoid making several queries.** This is an important principle as joins are really time-consuming in *MongoDB* and we usually need to make several queries. Although I had it in mind while I was designing the database, I did not find any use in this project.

As it is explained in the section *Database References* of [11], there are two ways of relating documents: *manual references* and *DBRefs*. I have opted to use *manual references* because they are simple and sufficient for this case. If I had chosen to use *DBRefs*, the referenced collection (`users`) should be specified every time a new document is introduced and as there is only one collection referenced this does not

make much sense. It is more logical to use *DBRefs* when it is necessary to specified the collection and database referenced.

Another thing worthwhile mentioned is that both prescription and comments are very similar they are stored in the same collection. In fact, in the mobile app they will be processed in the same way and storing them separately would imply querying two collections. Also, there is information that the actigraph application calculates from other data. This calculated data is not stored because, as it has already been mentioned, it is more important to be fast in the storing phase than in the analysis. Moreover, analysing the data really fast is not a priority, so it is not worthwhile saving data that we can calculate when we need it.

Although the design of the database was my responsibility and I took the last decision, I defined the variables of the `users`, `comments` and `records` collections with my classmate David Peñas Gómez, as he needed to know them for his project as soon as possible.

2.3.2.2 Users and roles

Authentication is a very important part of the database as it is going to be accessed from multiples devices an some of them may expose the IP and port of our server and we do not want anyone else to have access to it. Because of the same reason I had placed great importance on creating users with an specific role for every part of the project. I had given each users only the indispensable permissions and above everything I had avoided removing information to the extent possible, so that, in case that we detected suspicious connections, we would not lose information. We are now going to explain in detail what permissions every user needs, making reference to the collections of 2.3.2.1.

The mobile is used to send daily tests and it also manages users registration and log in. That means that the mobile application needs to read the users from the `users` collection (for the sign in), to insert them (for the sign up) and to update them (when updating the profile). It also needs to insert tests in the `mobileTests` collection (when they are filled in), to read them (in case that it is needed to read some tests that are not stored in the *SQLite Android* database) and to update them (when the user changes the answers of a test that has already been stored in the *MongoDB* database). It is planned that the mobile app receives reminders about the medication and comments, although this feature is not already implemented, so the mobile application will also need to read the `comments` collection.

The actigraph only sends the data collected once and never updates or reads it. So, the actigraph only needs to insert data in the `actigrpahData` collection.

The web application is what the psychologist is going to use to interact with the database. He adds new patients to the database and he also consults and edits their profiles, which means that he can see, edit and insert user in the collection `users`. The doctor can also see and update the patients' medical history, therefore he needs to find, update and insert records in the `records` collection. The results of the analysis phase are displayed as the doctor could want to see them, so the web has permission to find in the `analysis` collection. And eventually, in the web the prescription and the comments to be received in the mobile application are also introduced. The doctor can see, update, instead and remove comments or prescriptions in the `comments` collection. This is the only case when removing data is allowed, as the doctor could introduced a prescription or comment by mistake and, in that case, it should be deleted. We could consider having an attribute to invalid comments instead of permanently remove then and this way the web would not need remove permissions. But at this first stage we have decided to keep it this way to simplify.

The analyst has to analyse the information collected by all the sources already mentioned, so it has permissions to read all collection. It also can perform find, update and insert actions over the `analysis` which the conclusions of the analysis are saved. Depending on the technique used in the analysis phase, updating the data stored may not be needed, but we have added as the technique is not already clear.

Last but not least, I created an admin user who has the *dbOwner* role to manage the database. The commands needed to create, find and delete the users with the roles described in this section are in Appendix: Creation of users and roles.

2.3.2.3 Indices

By default *MongoDB* creates indexes only on the `_id` field. Indexes are very necessary for good performance as we can make searches faster by adding them. As we are going to query the `email` attribute repeatedly we have added an index over it. As we want the email to be unique in our database, we have added the `unique` attribute to the index to guarantee its uniqueness.

```
db.users.createIndex({"email": 1},
                     {"unique":true})
```

To check if the index is working as we expect, we can use the operator `explain`, which returns a JSON document with a lot of useful information, including the number of documents examined, the execution time, the indexes it have used, etc. For example:

```
db.users.find({"username":"ana@email.com"}).explain(true)
```

To check the indexes we have defined we can use:

```
db.users.getIndexes()
```

And to delete a index:

```
db.users.dropIndex( "email_1" )
```

We also have to add an index to the `user_id` attribute of `comments`, `records`, `analysis`, `mobileTests` and `actigraphData` collections as it is acting as a *foreign key*. The *shell* commands to add them are:

```
db.comments.createIndex({"users_id": 1})
db.records.createIndex({"users_id": 1})
db.analysis.createIndex({"users_id": 1})
db.mobileTests.createIndex({"users_id": 1})
db.actigraphData.createIndex({"users_id": 1})
```

Text indexes allow us to perform advanced searches over text fields of a document. As we are storing the user's first and family names in the key `name` in the web application the doctor may want to search a patient by its family name and a text index would be very useful for that. Also, the doctor may want to search a patient by its email but he does not remember it exactly, so a text index would be useful here to. Consequently I have added a text index over the document users for this two keys (we can only add one text index but it can have multiple keys). As I think that searching by name will be more common than doing it by email, I have given the name a weight of 3 while email has the default weight of 1. I have also given the index a custom name. As, at least at the beginning the project is going to be in Spain we have also specified Spanish as the default language. Although the language should not be really important, as we are only querying names and emails.

```
db.users.createIndex(
   {
     name: "text",
     email: "text"
   },
   {
```

```
      weights: {
        name: 3
      },
      name: "TextIndex",
      default_language: "spanish"
    }
 )
```

2.3.3 Queries

This section illustrates how to query the data in the designed database. The queries are going to be done for the *Mongo Shell* as the languages to be used in some part of the project are not yet clear. And even if they were clear we may want access the database with other languages in the future.

First of all we have to change to the `bipolarDatabase` database:

```
use bipolarDatabase
```

Let's start querying the `users` collection, as it is used for most clients and has indexes whose use is worthwhile showing. To insert a new user with all the attributes in the `users` collection:

```
db.users.insert(
  {
    "email" : "lolita@email.com",
    "name" : "Lolita Martínez López",
    "birthDate" : ISODate("1980-02-24"),
    "gender" : true,
    "pin" : NumberInt(123456),
    "cohabitation" : NumberInt(0),
    "diagnosis" : "F00.0",
    "diagnosisAge" : NumberInt(21),
    "senLit" : true,
    "senVal" : false,
    "senCar" : false,
    "seasonality" : true,
    "maniaCrises" : NumberInt(3),
    "mixedCrises" : NumberInt(0),
    "freePeriod" : NumberInt(6),
    "psycSymp" : false,
    "others" : "The patient suffers from migraines"
```

```
  }
)
```

Note that we do not introduce the *_id* as it is automatically generated. If there is an attribute that we do not want to insert we can just not include it. By default, the *Mongo shell* treats all numbers as floating-point values, so we use *NumberInt()* constructor to explicitly specify 32-bit integers. *ISODate()* constructor returns a Date object using the *ISODate()* wrapper. If we want to know all the users that are in the database we can execute:

```
db.users.find().pretty()
```

The `pretty` function is to display the results in an easy-to-read format. If we want to find all users that satisfy some conditions, for example all women that are not sensible to Valproate:

```
db.users.find(
  { "gender" : true , "senVal" : false }
).pretty()
```

If we want to find an specific user, for example using the already mentioned typical search by email:

```
db.users.find( { "email" : "lolita@email.com" } ).pretty()
```

We can also find a user by his id:

```
db.users.find(
  { "_id" : ObjectId("573ee9d4ae07fbe5b79bd0d0") }
).pretty()
```

For the last two queries we can also use `findOne()`, as we want only one record. We should take into account that `find()` returns a cursor and `findOne()` returns document. I used the first alternative as the `pretty()` function can only be used over a cursor, to be able to read it easily, but probably in an application using `findOne()` would be better. Also, most drivers include a function to search by id which is worthwhile using, but there is not this function in the *Mongo shell*.

The next query illustrate how to work with dates and with comparison operators.

It finds all users who were born from 1990 on:

```
db.users.find(
  { "birthDate" : { $gte :ISODate("1990-01-01T") }
  }
).pretty()
```

To remove all users:

```
db.users.remove({})
```

And if we do not want to remove all users we can add conditions. For example, all users whose id is *573ee9d4ae07fbe5b79bd0d0*:

```
db.users.remove(
  { "_id" : ObjectId("573ee9d4ae07fbe5b79bd0d0") }
)
```

We can also look for users using the text index created in section 2.3.3. For example:

```
db.users.find( { $text: { $search: "Gómez" } } ).pretty()
```

It will find all users that has *Gómez* (or *gomez* as it is case insensitive by default) in their name or email. We can go a little bit further and print the score and order the results by the score given by the weights of the index:

```
db.users.find(
  {$text: {$search: "Gómez"}},
  {score: {$meta: "textScore"}}
).sort({score:{$meta:"textScore"}}).pretty()
```

We could also want to search the users whose last names are *Martínez Gómez*:

```
db.users.find(
  { $text: { $search: "\"Martínez Gómez\"" } }
).pretty()
```

Querying others collections it is done in the same way, as the most common functions are *insert*, *update* and *find*. So we are just going to show some examples that

could not be illustrated over the `users` collection. Let's start by the `actigraphData` collection. First, we are going to introduce a record in this collection, as it has Doubles, Arrays and Dates with times that we do not have in the `users` collection:

```
db.actigraphData.insert(
  {
    "user_id" : ObjectId("57416ee9494026efedaa7735"),
    "date" : ISODate("2016-05-22T09:43:18Z"),
    "position" : [NumberInt(1),NumberInt(2),NumberInt(3)],
    "lux" : NumberInt(27),
    "button" : false,
    "temperature" : 23.2
  }
)
```

It also important to illustrate how arrays are queried. For example, to find all actigraph records in which the y coordinate is 2 we do:

```
db.actigraphData.find(
  { "position.1" : 2}
)
```

It is important that array indexes start at 0. There are more complex operations over arrays but we do not need them as the client will work with the whole array as the position is not useful without the three coordinates and it is very unlikely that users are found by their position.

We are now going to use the `records` collection to illustrate how to query nested documents. For example, to find the users whose hostility score is less than 20 in the Positive scale part of the PANSS interview:

```
db.records.find(
  { "panss.panssPos.hostil" : { $lt : 20 } }
)
```

Lastly, we are going to use the `mobileTests` and `users` collection to illustrate how joins can be done. For example, to query all mobile test of an specific users by his/her email:

```
id = db.users.findOne( { "email" : "ana@email.com" } )._id
```

```
db.mobileTests.find( { user_id: id} ).pretty()
```

2.4 Integration

Lastly, we are going to comment on how to integrate *MongoDB* with other languages. The available drivers are detailed in the section *MongoDB Drivers* of [11].

Let's start with the mobile phone application. It is being developed for Android, so the official *MongoDB Java Driver* is being used. You can see the code of the mobile application, which you can find in Github (see 3.2), to know the details of how this integration is done.

For the web application we have thought to use *Node.js*. If that is the case the official *MongoDB Node.js Driver* could be used. Also, using *Mongoose*, an ODM for *Node.js* which allows us to communicate with a *MongoDB* database simply and easily, would make the integration much more comfortable. We could also consider using *Ruby on Rails* for the development of the web application. In that case we should use *MongoDB Ruby Driver* or if we want to include validations, associations, and other high-level data modelling functions, the ODM officially supported by *MongoDB*: *Mongoid*.

The actigraph libraries are written in *C#*, so it makes sense to write the application that sends the actigraph data to the database in the same language. Therefore the official *MongoDB C# Driver* can be used for the communication with the database.

Lastly, for the analysis phase *R* is going to be used. As it is explained in [35], to use *R* together with *MongoDB* there are mainly two packages: *rmongodb* and *RMongo*. I recommend using *rmongodb* as it appears to be more popular and it does not require *Java*, while *RMongo* uses the *MongoDB Java Driver*.

3 Android

The objective of this chapter is to comment on the process followed to learn *Android* and to design, develop and test an *Android* mobile app to collect data to be analysed for medical purposes.

3.1 Learning Android

I had not have any contact with *Android* before this project so I spent a considerable amount of time learning about it, although my previous experience with Java made it much easier. Some of my classmates took the iOS and Android programming course in the Complutense Summer School and they send me the projects they did and the slides they used at the course ([27]). I also asked some students that took or are currently taking the subject of development of mobile app (*Programación de aplicaciones para dispositivos móviles*) at the faculty to send me the material they used or are using, that is mainly the slides [26]. So I started with the material of these two courses and then I used the complete official guide of *Android*, see [6]. For more specific topics or doubts I used other guides, blogs and forums, primarily [14].

3.2 Application to collect medical data

The *Android* mobile application which collects data to be analysed for medical purposes plays a really important role in our project, as it would allow us to know how the bipolar patients feel they are. When working with mental illness, this kind of subjective information is really valuable, not because of its veracity but because of what their changeability can reveal. The code of the app is free software and can be found here, together with the .apk, the final protoype, screenshots and other relevant information: https://github.com/Ana06/medical-data-android

3.2.1 Specification

Since I started working in the project the application specification has changed several times because of different reasons:

- **The necessity to have a first version of the application than can collect data as soon as possible.** Some functionality has been simplified or discarded

as it would have delay the development of the app in excess, altough some improvements are planned to be introduced in future versions (see 3.2.5). As an example, at the beginning some games wanted to be implemented as part of the daily test to collect objective information about concentration, reflexes, etc. Instead, the time to introduce the pin is measured as a concentration indicator.

- **Something we had planned to do is impossible, too difficult or not recommended.** For example we wanted to register some information that are not accessible in *Android* for security and privacy reasons such as the use of other apss and the hours the screen is on. We had also thought about using the phone as an accelerometer, but was quite unlikely that the battery can support it.

- **Issues that during the design or implementation of the application we realised that had not been considered.** For example, if we wanted to register the number of tries and timing all of them or only one/some of them when introducing the pin.

- **Bad communication.** In the mobile application that uses the doctor we where told to ask the birth date and in the mobile application to ask the age. When considering the integration of the two applications, we had to change the birth date by the age in the mobile app.

- **The necessities of the project changed.** As we depended a lot on the client, the hospital and specially the doctor working with us, changes had to be made almost every time I had a meeting with him. As an example, the decision to collect caffeine was made towards the end of the development of this project. The way the register is wanted to be done was also changed several times, as it can be seen in the prototyping section.

3.2.1.1 Final specification

The *Android* mobile application is in English and Spanish. The first time the application is used the user should sign in or sign up. If the user has already been registered by his doctor, he had already received an email with the 6 digits password that, together with his/her email, can be used to sign in, and all the information needed by the app will be get from the server. He would do the same if he had already registered. Otherwise the following information is asked: Name, email, age, gender and a 6 digits pin (it should be introduced twice). The chance to read the terms and conditions, which must be agreed, is given. Then, the user will be inform that the register has been correctly completed. The correctness of the information introduced in both the sign in and the sign up processes will be checked and, if there

is any error, it will be indicate appropriately.

It will be allow to modify the daily test answers. The reason why we allow that is because if after completing the test something that change significantly the patient's mood happens, the users should be allow to change his answers. The main page is the one that allow start the daily test or modify it if it has already been filled today. To start or modify the daily test it will be needed to introduce the pin correctly. If an incorrect pin is introduced the pin is cleaned. The number of tries and the time since the first number is introduced until the start button is clicked having provided the corrected pin is saved. The last try (since the first number is introduced after cleaning the pin space until the start button is clicked having provided the corrected PIN) time is also saved.

The daily test asks the following data, which is what is stored in the `mobileTests` collection defined in 2.3.2.1:

1. **Affective state:** 7 points with a center one which is represented accordingly.
2. **Motivation:** 7 points with a center one which is represented accordingly.
3. **Concentration:** 5 points.
4. **Anxiety:** 5 points.
5. **Irritability:** 5 points.
6. **Sleep quality:** 5 points.
7. **Menstruation:** Yes or no.
8. **Caffeine:** Integer
9. **Alcohol:** Integer
10. **Cigarettes:** Integer.
11. **Other drugs:** Yes or no.
12. **Time to go to bed:** Time.
13. **Time to sleep:** Time.
14. **Time to wake up:** Time.

For most information about this data, you can see the help texts in the application. Question 1 to 6 use *Likert scale*, the number of points in these question was greatly discussed and there is information about it at [25]. For these questions the answer of the previous day the test was filled is indicated. We do that to allow users to be more accurate as they can compare with the answer they gave yesterday and value how their mood have change since them.

When all the questions has been answered it will be indicate to the user and show the main page again given the possibility to change the answers, as it has been already mentioned. If the send button is clicked and there is any question that has not filled correctly it will be indicated appropriately. The main page has a menu with the following items: profile, configuration and information. The profile page allows to modify the personal information asked in the registration. Introducing the pin correctly is needed to change it. In the configuration page the option to disable or enable using the mobile data to send information is given. The information page shows the name of the app, its icon, the version, the name of the developer and allows to show the terms and conditions.

3.2.2 Design

In the design of the application I followed the *Material design* suggestions and recommendations in [24], [6] an [32]. *Material design* is a new visual language proposed by *Google*, which represents a rational space in movement. Lights, shadows, typographies, spaces and colours help us focusing our attention and tell us where objects are in relation to other objects and how they move. Movement is meaningful and continuous, feedback is subtle but clear and transitions are coherent and efficient.

Regarding to the colour used in the application, for the primary colour I used the pink from the material design primary color palette ($\#E91E63$) and for the primary dark color a variation of this pink also in the primary palette ($\#C51162$). Consequently, as the accent colour must contrast with the primary one, I used one of the blues from the secondary palette ($\#00B0FF$) for the accent colour. The primary colour is the most widely used across all screens and components. The accent colour should be used for the floating action button and interactive elements, such as selection controls, links, text fields and sliders. However, as the main buttons of the app are to big I decided to use the primary colour for that, to avoid the accent colour to turn the most used. In fact, there are some *Google*'s applications, such as *Gmail*, that use the primary colour for buttons.

Apart from the colours, I did not have to worry about following *Material design* principles as *Justinmind* uses it and I chose a theme in the application project that uses it too.

3.2.2.1 Prototyping

The aim of this section is to illustrate the process followed to prototype the mobile application. First, I did some sketches of lower fidelity and then I did eleven computer prototypes before starting the development or at the beginning of it. I used

Justinmind, which is an prototyping tool with advantages features, for the computer prototypes and, as it is not simple to use, I need to use the *Justinmind* complete guide with explicative videos in [28]. Every prototype consisted on images (*Visual Prototypes*) and from the third prototype also videos (*Functional Prototypes*) to illustrate how the application is used.

In the Appendix: Prototypes images you can see images of the different prototypes and they may make easy to understand the prototyping process. In addition, the videos of the last prototype can be seen in the following links. I do not include the rest of videos here as this one is the one with highest fidelity. Both the images and videos are in Spanish.

Daily use: https://youtu.be/rmyIEO8Utz8

Registration: https://youtu.be/yqyoxoveptU

The first prototype (figure 1) is the one of less fidelity and was done before the questions that we want to ask were clear, as we had to find a balance between collecting as much information as possible and not getting the user tired or bored by asking him too many question. In the second (figure 2), the questions were much more clear and I decided to introduce help texts for every question. In the third (figure 3), we eliminated the question about if the patient has taken his medicines (as it has no sense for most users and we decided that it is better to introduce the notifications feature in the future). We also introduced registration screens and change the way some questions were rated, using a center point. Unfortunately, a misunderstanding with doctor obliged me to change the rating again in the fourth prototype (figure 4). Apart from the center point I also coloured the answer given the previous day for a more accurate answer. In this prototype, I also changed the main screen, as I had been reading about *Android* issues and the grille used before could be difficult to implement for different screen sizes. Some questions were also modified. In the fifth prototype (figure 5) I basically added the terms and conditions in the registration screen. In the sixth prototype (figure 6) I added a settings screen and a confirmation screen to be shown after sending the daily test.

After a meeting with the doctor I realised that the rating method for the questions with center point was not clear and change the values range, -3 to 3 instead of 1 to 7. So I did the seventh prototype (figure 7), in which I also changed the way number were introduced as it could be really unconformable to fill for big numbers. The register was modified too, as we decided to repeat the PIN to avoid introducing a wrong pin without noticing it.

Then, I did some profs in *Android* to decide the menu that would be better for the project. I implemented a *Navigation Drawer* and a *Main menu* and I decided that

the second option fits better our necessities as we will have very few options in the menu and it will not be worthwhile using a *Navigation Drawer*. So, that is what was introduced in the eight prototype (figure 8), together with the screens of the profile, settings and information options.

In the ninth prototype (figure 9) the way the register is done was changed and a screen to indicate that the register has been successfully completed added. Also, the menu was introduced also for the register, but only with the information option, as the rest do not make sense in this part. I added the navigation *Android* bar too, which I had not added before because it is not easy to use it in *Justinmind* in screens where scroll can be done. Lastly, in the final prototype (figure 11) the colours were changed to use the colours recommended by *Material design* guides, as it has been previously explained. This final prototype is a high fidellity prototype. That can be checked by comparing it with the application developed (see Appendix: Screenshots of the final application).

3.2.2.2 Design principles

For doing the prototypes in the previous section and during the development I followed the following principles: **proximity**, **closure**, **consistency**, **feedback and visibility**, **management of the visible state** and **freedom and control**. All this principles are explained in detail in [32] and I am going to mention examples in the mobile application for all of them.

Proximity principle

Related elements must be close or even grouped together, as that favours learning and memorability. The clearest example in the app is the help icon next to the title, indicating that the icon opens the help text for that specific question.

Closure principle

This principle states that a closure of a process has to be natural, a continuation of the process itself. Two good example in my application are the daily test and the register, as to fill in the information you do scroll vertically and the finish button is at the of the screen. If we I had put this button at the beginning of the screen I would not have followed this principle.

Consistency principle

Users learn easily the concepts that are consistent with their previous knowledge. There are two types of consistency: interior and external. The first one consists on ensure that all controls with similar functionality have a similar appearance and vice versa. Also, controls which have a different appearance may have a different functionality. In the app, the questions of the daily tests that are similar have a rating

similar (or exactly the same one) and the ones that are different have a different rating. External consistency consists on use the same expression that the system and other application use. In the app, the menu works exactly as it does in many other *Android* applications and the option and help icons are the ones an *Android* user is used to seeing.

Feedback and visibility principle

The interfaces must change to indicate their state. Apart from the typography, lights, shadows and colours used in Material design to focus the attention, there is also visual feedback, as I have used *Android Toast* messages to indicate errors. Also, there are screen to inform that everything went right after important processes, such as registration and sending the daily test. I have also keep the red color to indicate errors as it is a cultural restriction. Other applications uses other kinds of feedback too, such as auditive and haptical feedback. But using auditive feedback is not a good a idea for our app, as bipolar patient may feel uncomfortable if the application sounds. Using haptical feedback could have been useful in the application, but I didn't have time for it.

Management of the visible state principle

This principle focuses on give the user information about the system while he is doing any process. As there are no long process in the application it does inform of the navigation state. It would become really important if we complicate the daily test in the future and split it in several screens. However, the application informs about the model state, by changing the name of the button by *CHANGE ANSWERS* instead of *START*. It also allow to know the interface state, for example by changing the button of colour when being clicked, informing that this operation is not longer possible and avoiding this way that the user click it several times during long operations (the ones involve a communication with the database)

Freedom and control principle

We have to allow the user to feel that he has the control of the processes. Because of that, I had avoided to use confirmations in the daily test and instead allowing him to change the answers afterwards. Similarly, instead using confirmations in the registration there is a profile screen were the personal data can be modified afterwards.

3.2.3 Implementation

The mobile application designed has been developed using *Android Studio*, the Official IDE for *Android*. Some manuals and guides recommend to use the *Eclipse* ADT plugin, but it is no longer supported so I decided not to use it. During the development I used

the guides and documentation at [6] and I tried to follow the official *Android Code Style Guidelines* at [4].

3.2.3.1 Android versions

When choosing and Android version we have to can into account that when more versions are supported, there are more things to take into account during the development. Some methods are deprecated in some versions and does not exist in some others, and the same happens with some functionalities. Taken that into account, the application can be used in Android 4.1 Jelly Bean (API level 16) or higher, which according to Android Studio includes the 86% of the Android devices.

A good example of the works that involves supporting several versions is the way I managed the `Timepickers` in the application:

```
if (Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP_MR1) {
  questions[11] = tp12.getHour() * 60 + tp12.getMinute();
} else {
  questions[11] = tp12.getCurrentHour() * 60 +
    tp12.getCurrentMinute();
}
```

3.2.3.2 Material design

Some characteristic of Material design, such the material theme, are only available in Android 5.0 (API level 21) or higher. Applications can be designed to use it in devices that support it while being compatible with older Android versions. I managed to do that by using the *v7 Support Library*, which can be used with Android 2.1 (API level 7) and include some features for material design when available. I also used the `Theme.AppCompat.Light.DarkActionBar` theme, as `Theme.AppCompat` themes allows us to set the material design color palette theme attributes.

3.2.3.3 Documentation

Taking into account that the development of the application is going to continue after I finish my project by other persons, I was concern about the importance of document it. I used the Javadoc Standard Comments as it is recommended in the *Android Code Style Guidelines*. Although, it is only compulsory to write Javadoc comments in classes and nontrivial public method, I wrote them in all methods, including private ones, to make easy to understand the code. In some overriding method I just wrote `@override`, as that keeps the Javadoc comment of the class it overrides.

3.2.3.4 Classes

An activity is a concrete thing that the user can do in an application. All activities of the application developed interact with the user, so the Activity class creates the window to do it and because of that they have at least one view associated.

Apart from the activity classes, I created two classes to manage the *SQLite* database: `FeedTestContract` and `FeedTestDbHelper`. *SQLite* is the default database in *Android* and there are classes to manage it easily. Specifically, I used the `SQLiteOpenHelper`, which is a helper class to manage database creation and version management with a useful set of Application Programming Interfaces (APIs). When using this class to obtain references to the database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup. To use it, I created a subclass called `FeedTestDbHelper` which overrides the `onCreate()` and `onUpgrades()` methods. I used a contract class called `FeedTestContract` to define the database schema, which is used in the `FeedTestDbHelper` class to create the database. The database is used to store the daily test locally, in case that there is no internet connection in the moment they are filled in. Also, the last two tests are used to indicate the user the answers he gave the previous and in case he wants to change the days of the current day.

Sending the daily tests to the server database has been more difficult that it may seem, as there are many factors to take into account. We need to keep the last two test but we send them as soon as possible to the database. Then, we need to know what tests have been sent and what of them not, as the user could send several test without connection. Also, we need to check if there is already a test with the same date, to update it instead of insert it again. So, I have created two classes to manage all that. The first one is `SendTest` and is the one which connect with the database to send the daily tests. It inherits from the class `AsyncTask`, which allows to perform background operations ans publish results on the main thread without having to manipulate threads and/or handlers, as the connection with the database is a long operation that has to be run in the background.

I have created the `TestsService` class too, which inherits from the `Service` class. A service is an application component that can perform long-running operations in the background and does not provide a user interface. Although a service runs in the same process as the application in which it is declared and in the main thread of that application, by default. The service will use the `SendTest` to send the daily tests and as it is a intensive operations which is done while the user interacts with an activity from the same application, the service will slow down activity performance. To avoid that, it is run in a different thread with background priority. If the service can not send all tests because of connection problems, a *BroadcastReceiver* which notifies connection changes is set. When the app is closed, the service is closed too, but it is restarted to avoid losing data. There is a bug in Android 4.4.x that prevent the server

to restart when the app is closed. This bug also kills *BroadcastReceivers*, but do not kill alarms, so adding an *Alarm* we could restart our service to avoid losing data in those versions.

For the sign in, sign up and the profile I created the nested classes called `DownloadRegistration`, `SendRegistration` and `UpdateRegistration` respectivily, which are similar to the `SendTest` class. For helping me to manage the registration I also created the `User` class, which has methods to create a user with all the information needed and to save him in the `SharedSettings`.

I also needed a the nested class to allow the terms and conditions in the register to be clickable: `MyClickableSpan`, which inherits from `ClickableSpan`. I created three classes to manage the custom ratings too. As I wanted to use two different custom ratings with a lot of similarities I took advantage of inheritance. So, I created a class called `RatingStars` with the similar functionality, and two classes (`RatingStars5` and `RatingStars7`) which inherit from the first class and contains the specific functionality.

Lastly, I have a class called `Variables` which contains static final variables and methods used around the program.

3.2.3.5 Manifest

Every Android application must have a manifest. It contains important information such as the name of the Java package (unique in the device), the components of the app (activities, services, intentions it can manage, etc.) and the permissions.

In the application manifest you can find the name of the Java package, the name, icon and theme of the app, all the activities, the test service explained in the previous section and the following permissions needed to check the connectivity state: `INTERNET`, `ACCESS_WIFI_STATE` and `ACCESS_NETWORK_STATE`.

3.2.3.6 Main challenges encountered during the development

Apart from the compatibility in some Android versions already commented, I encountered several challenges. The ones I considered to be more difficult to solve are in the following list:

- **Finding out why the service was not restarted when the application was closed.** It is a bug in Android 4.4.x that by coincidence was the version I

was using to test the application. The reason why it was so difficult to discover is that there is not almost information about that. I finally found a detailed article about the problem at [17].

- **Finding out why the authentication with MongoDB does not work in Android.** When using *SCRAM-SHA-1* authentication method from the *Java Driver* `java.nio.channels.AsynchronousSocketChannel` is called and it does not exist in Android, so it causes an exception and as a result it does not work. I solved it by changing the authentication mechanism to the previous one (*MONGODB-CR*) instead the default new one in releases 3.* (*SCRAM-SHA-1*). It was difficult to solve because this problem is only for Android and the Java driver works fine in general, so there is no a lot of information about this error neither.

- **Managing times.** Managing times was a complicated issue during the development of the project because of time changes as UTC is used by default and I needed to manage it in some places in the device local time. The problem more difficult to solved was to manage the day light savings, as it is not something common in most countries an it is not taking into account in most of the solutions I found on the Internet.

- **Making the title to be present when get a question focused because there is an error.** Both in the register and the daily text, we want not only to get focus on the answer when there is an error (what is done by default when calling `requestFocus()`) but also to be see its title. In addition, most of the daily text answers can not request focus, as they are not `FieldTexts`. I did that by using the `Rect` class and the `requestRectangleOnScreen()` method. It was difficult to me to understand who these class and method worked at the beginning.

- **Making the terms and conditions clickable in the register.** The terms and conditions are a subtext in a sentence and we do not want the whole sentence to be clickable. I tried to managed it by defining two `TextViews` in the view, but it did not work. I managed to do it by using the `SpannableString` and `MyClickableSpan` classes in the activity associated.

- **Customizing the `NumberPickers` in the daily test to add an empty option by default**. That is important because we want to force the users to fill in all the question and because of that that the 0 value, which is an acceptable value, can no be the default one. I managed to do that by using the `Formatter` class and the `setMinValue()`, `setMaxValue()`, `setWrapSelectorWheel()` and `setFormatter()` methods.

3.2.3.7 Final application and testing

Screenshots of the final application in two different devices with different Android versions can be seen in the Appendix: Screenshots of the final application. Also, a video of how the application looks and how to use it in a Nexus 4 with Android 6.0.1 can be watched in the following link:

https://youtu.be/InZCw9WAZzI

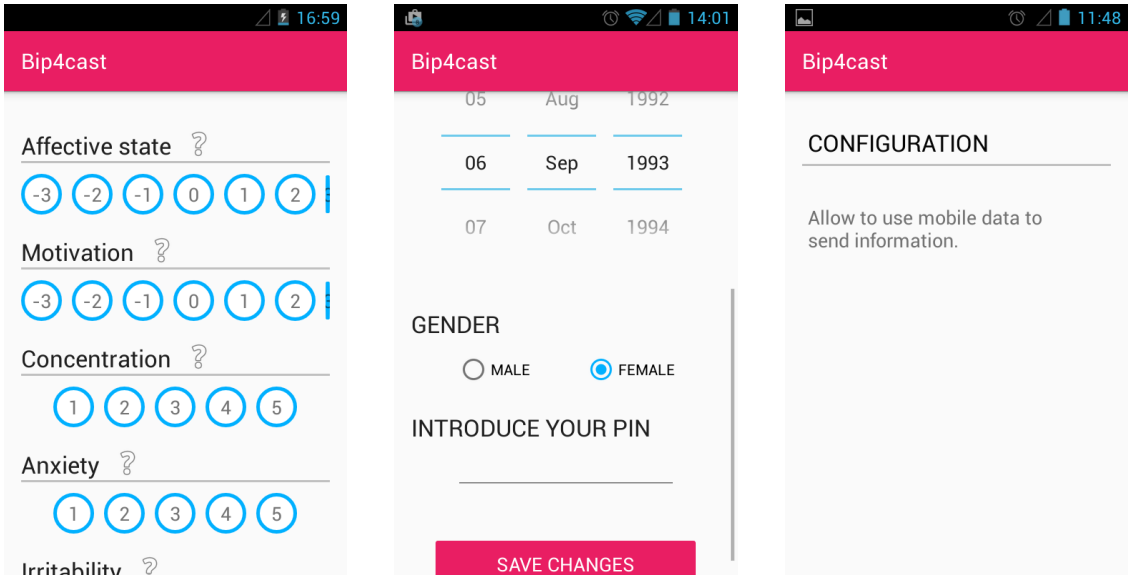
Regarding the testing, during the development of the project I mainly used my own mobile phone, an **Elephone P3000S with Android 4.4.2** to test the application. But I also tested the application in a wide range of phone and tablets. While testing the app I discovered that it was not been displayed properly in some devices with small screen (specially those with older *Android* versions), as it can be seen in picture 3.1. The problem of the ratings was caused because of being using text inside the circles of the ratings. The solution was to manually set the text size for different screen sizes. Also, the scroll was not working properly, although it worked fine in other versions. I solved it by adding a `FrameLayout` between the `RelativeLayout` and the `ScrollView`. The last error I found was that the checkbox did not appear in small devices because of the way padding was set, so it was easy to solve it by changing the way of doing it the `RelativeLayout`. At the last minute I also discover that there is not space between the last line of the help text and the pink box in old version of Android (figure 3.2). Although it does not seem difficult to solve, I did not have time to do it and in fact it is not a hight priority issue.

I could also have used *Android Studio* emulator, but it is really slow, specially for an old laptop like mine and as it was not difficult to have access to several devices to test the app I did not use it.

3.2.4 Usability testing

Usability testing is a technique which consists on allowing users to work with our application directly and observing if they can use it for its intended purpose. The information that we obtain from usability testing is more valuable than using any other technique which does not involve users and it is the only way to know how real users will use our system.

The usability testing can be helpful in any part of the project, but it has a high cost, as we need to recruit users, prepare it, doing it and analyse it. Because of the time needed, I only did usability testing at the end, but taken into account that the development of the application is going to continue and some new features are planned to be introduced, the objectives of this testing were the ones of an evaluation



(a) Ratings did not fix the screen

(b) The last part of scroll views was not seen

(c) The checkbox did not appear

Figure 3.1: Application problems in some devices

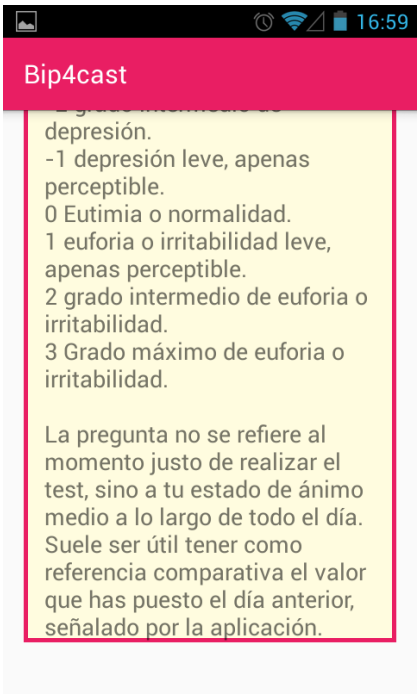


Figure 3.2: No space after the help text in some old devices

test and not the ones of a verification test. In other parts of the project, I evaluated the application by showing it to my directors and other people working in the project and taken their feedback into account.

The aim of a evaluation testing is to check if the users are able to do complex tasks without any help. Although quantitative information is written down the most important information is qualitative. The interaction between the interviewer and the user is minimum and the user has not only to indicate how he would do a task, but to execute it too. The result of this type of test is a list of parts of the application where the users get stuck, make errors or find inconsistencies. Afterwards, improvements to solve the problems found are proposed.

I used the *Chapter 5* of [32] for writing this chapter and to help me to do a good and useful usability testing.

3.2.4.1 Preparation of the usability testing

First of all, I had to decided what parts of the application I wanted to evaluate. Taken into account the objective of the testing, I decided that the best was asking the users to register in the application and fill in a daily test, as this cover the main purpose of the app (filling daily tests) and the register is indispensable to start using it. I summarized my objectives in the following questions, which I expected to be answered by the usability testing:

- ¿Is it easy to complete the register?
- ¿Do the users understand the questions of the daily test?
- ¿Is it easy to fill in the daily test?
- ¿Do the users understand the icons of the application?
- ¿Do the users know what elements are touchable?

I was especially interested in knowing how many questions of the daily test the user does not understand, how many errors does the user make and the time that takes him to complete the daily test. I was also interested in finding out how comfortable and difficult is using the application.

I needed to choose the users too, as they had to be representative of the users of the application. As the app will be used by bipolar and healthy adults, I choose three users of different ages and genders, although they medical information can not be revealed because of legacy and privacy reasons.

After that I made a description of the environment, the tools that would be used and my tasks as a moderator: The sessions will take place in my living room. The users will use their own Android mobile phone to ensure that the errors they make are not because of being using an unknown device. There will be a camera recording the

screen,the interaction with it and the audio. The camera will be on the left side if the user is right-handed or in the right side if the user is left-handed. Ideally, there should be several people participating in the usability testing, but in that case I will be alone. This also has advantages, as the user will fell more comfortable with only one person. First, I will sit down in front the user, give him a short introduction about the application and explain him the task. After that I will move to be able to see the user using the app and ask him to start. I will take notes while he is using the application. At the end, I will allow the user to make comments or explain anything he wants and I will ask him to fill in a questionnaire.

I planned to use the *think-aloud* technique, which consists on asking the users to speak during the testing explaining what they are thinking. We may have to remember the users to keep talking and we can use silences and the phatic function of language in our favour. That technique is really useful as it allows us to understand the user mental model and if what the users expects is what it is happening in the application. It also helps the user to concentrate in the task and reflect on the actions he will do.

When using this technique it is really important to explain it properly to the users, so I will give them an introduction, which will be something like that (it is in Spanish as it is the language in which will be done the testing):

La aplicación que vas a usar ha sido diseñada para recoger datos diariamente sobre el estado de ánimo de los usarios para enviarlos a un servidor y analizados con el objetivo de poder entender mejor enfermedades relacionadas con trastornos afectivos y mejorar su tratamiento. Lo que debes hacer es registrarte en la aplicación y después completar el test diario de hoy. Todo esto debes hacerlo tú solo, así que yo no te puedo ayudar. Durante la prueba es muy importante que expliques en voz alta todo lo que haces, como porque pulsas un botón o las razones por las pones una respuesta en una determinada pregunta.

At the end of any usability testing it is recommended to do an interview, a questionnaire or a combination of them. I will only ask them a general question, as I do not want that they get tired and after that I will ask them to fill in a questionnaire. This way I will have quantitative information. I will use a version of the IBM questionnaire, the PSSUQ, with which I will obtain four variables: the general satisfaction (OVERALL), system usefulness (INFOQUAL and the quality of the interface (INTERQUAL). I used this questionnaire because it is simpler than others, such as UTAUT, and it is quite popular. The questionnaire that I prepared for the interview can be seen in the Appendix: Usability testing questionnaire.

3.2.4.2 Testing sessions

In this section I am going to provide links to the records of the usability testing. Remember that the test was done in Spanish.

As I have already mentioned, first I gave a short introduction. The introduction of the usability testing with the third user can be seen in the following link. The introductions with the two other users are very similar, I choose that one because the audio quality is better:

https://youtu.be/wuXlG2m4rOg

The testing itself with every users can be seen in the three following links. Remember that every user used his own mobile phone.

User 1 (LG-440 - Android 4.1.2): https://youtu.be/xSxgU9x7sP4

User 2 (ZTE blade Apex - Android 4.1.2): https://youtu.be/fmGYmMUiBxw

User 3 (Aquaris E4 - Android 4.4.2): https://youtu.be/POcuAoa6ni4

The short interview after the testing with the first user can be seen in the following link. I chose that one because she was the most communicative user of the three users in this part:

https://youtu.be/GzMy9bZuF3k

The questionnaires the users filled in can be seen in Appendix: Usability testing questionnaire.

3.2.4.3 Analysis and recommendations

The first users was the younger of the three. She did the testing really well and fast with no errors. She was shy and not very talkative, but when she spoke she gave me really valuable information. The second and the third user get stuck trying to sign up, as they were signing in instead. And as the form it is cleared with clicking sing in that takes them too long and they were much slower than the first user.

The three users found filling the birth date very uncomfortable. Also, the second and the third users did not find the help buttons, event though they said not to understand how to answer the questions. The first user found it easily, but she only used for the two first questions. After that, she assumed how the questions works.

That made her to misunderstand the question about concentration, where the question is asking for concentration problems, so the 1 is the best value, in contrast to the other questions.

The second user also get confused when the application took a little bit to sign up, although there is visual feedback, as the button change its colour. The three users, but specially the second one, said that it was difficult to know the time they get asleep and fell uncomfortable filling this question. Any of the users opened the terms and conditions, although probably that is not an usability problem, as they knew how to open them but they just did not do it. Also the second and the third users did not understand that the main screen after filling the test was to change the answers and that it was not compulsory.

So, if we go back to the questions we wanted to answer with the testing:

- **¿Is it easy to complete the register?** No. It is quite easy to fill the information, but not to find where to register.
- **¿Do the users understand the questions of the daily test?** Most of them, except the one of the concentration, the caffeine and the alcohol. It is explained in the help texts, but they did not open them.
- **¿Is it easy to fill in the daily test?** Yes, it is quite easy.
- **¿Do the users understand the icons of the application?** Most of them, but no the help icon.
- **¿Do the users know what elements are touchable?** In general they do, but the terms and conditions and the help icon are exceptions.

All this is qualitative information, but I also get the following quantitative information:

Questions of the daily test the users do not understand: 3 (Concentration, alcohol and caffeine).

Number of errors does the user make (without counting not understanding the questions): 0, 3 and 2 respectively.

Time that takes the user to complete the daily test: 4, 7 and 9 minutes respectively.

We can also get quantitative information from the questionnaires (which can be seen in the Appendix: Usability questions questionnaire) too:

OVERALL (average of all questions): 5.8

SYSUSE (average of questions 1 to 8): 5.3

INFOQUAL (average of questions 9 to 15): 6

INTERQUAL (average of questions 16 to 18): 5.3

The usability problems found can be summarized in the following list. The priority is indicated with a number between 1 and 5, being 5 the the highest priority.

- **The register is not clear**. We should change the register. Adding a new screen to make it more clear if you are signing up or signing in. Priority: 4
- **The sign in is cleared when there are errors**. We should should not clear it to avoid the users filling the data several times. Priority: 2
- **Changing the initial date in the birth date question**. We could change the initial date in the birth date question. Priority: 1
- **Help buttons are not recognizable**. We could change their color and add them relief. Priority: 5
- **Concentration question is easily misunderstanding**. We should rewrite the help text to make the 5 to be the best value, as in the rest of the questions. Priority: 5
- **Visual feedback when clicking a button is not clear**. For operations that can take a while (the ones that needs to connect with the server), we could add a Toast message to make it clear that the button has been clicked. Priority: 3
- **The question about getting asleep is uncomfortable and get users tired**. As the doctor was also thinking that this question may be less useful as we expected, we should eliminate it. Priority: 2
- **It is not clear that the main screen after filling in the test is to change the answer**. We should add a message to make it more clear. Priority: 3

3.2.5 Future changes

Obviously, the development of a large application like that has not yet finished and there are some future changes planned. I have included the changes suggested in section 3.2.4.3 after the usability testing. The changes are ordered by priority, being first the ones with highest priority. This changes have also been added to the *Github* repository by creating issues. In *Github* they have tags to classify them, and screenshots and other useful information has been included.

- **Change the text help of the question about concentration.** We should rewrite the help text to make the 5 to be the best value, as in the rest of the questions, to avoid misunderstandings.

- **Change help buttons colour and add them relief** to make them more recognizable.

- **Change the register.** Add an extra screen to make clear if we are signing up or signing in.

- **Eliminate the email from `SharePreferences`.** The email and the PIN are both stored in the `SharePreferences`. As these data is the one used to log in that could be a security problem. If we eliminate the email, internet will be needed to see the profile, but the application will be more secure.

- **Show a Toast message when clicking a button which activate an action that could take few seconds in finishing**. Improving visual feedback this way.

- **Add a message in the main screen after the daily test has been filled in.** This will make clear that now we can change the answers.

- **Eliminate the question about getting asleep.** As it is a new recommendation of the doctor and because of what I observed in the usability testing.

- **Translate help messages**, which are only in Spanish.

- **Stop clearing the sign in when there are errors** to avoid the users filling the data several times.

- **Add link to www.bip4cast.org** in the terms and conditions.

- **Store and send information about location** to the database as we thing that this data can improve predictions significantly.

- **Change the initial date in the birth date question** For example to be 10/4/1985, as user's birth date would be closer to that date than today's date.

- **Download the last two tests and store them in the database.** This way, when a user that has already been using the app sign in he will have the information of the test of the previous day and the test of the current day, if any.

- **Add Touch Gestures** to custom ratings to make them easy to use.

- **Add notifications** to allow the doctor to send the users remainders about their medication and other messages.

- **Solve bug in Android 4.4.x** There is a bug in Android 4.4.x and Services and BroadcastReceivers are stopped when the app is closed (closed not pause). So, after creating the Service for sending the tests to the database, an alarm should be programmed to restart the service.

- **Solve visual bug in some Android versions** In some Android version there is not space between the last line of the help text and the pink box.

4 Contributions

Not only have I learned a wide range of theoretical concepts and gained a lot of practical experience while doing this project, but I have also contributed to *Github* (4.1a) and *Stack Overflow* (4.1b).

(a) *GitHub* (b) *Stack Overflow*

Figure 4.1: Logos of the communities I contributed to

4.1 Github

GitHub is a web-based *Git* repository hosting service. It has a community of more than 14 million people and over 35 million projects. As it offers free plans for open source projects, it host the world's largest collection of open source software and, by storing past versions of source code, it allows to follow the development of projects.

My *Github* account is Ana06. The mobile app code, together with the .apk, the final protoype, screenshots and other relevant information, can be found here:

https://github.com/Ana06/medical-data-android

The code has been pusblished under GNU GENERAL PUBLIC LICENSE v3. It is a copyleft license that requires anyone who distributes the code or a derivative work to make the source available under the same terms, so everbody will be able to benefit from the current project and from future improvements and changes. More information about this license can be found at [19].

Contributors are accepted, so a list of issues has been created on *Github* with the future changes proposed in section 3.2.5. Anyone can contribute by submiting a pull request or creating new issues with new ideas or bugs. I will revise the correction of the code submited and its adherence to the official *Android Code Style Guidelines*

and, if everything is correct, I will incorporate the changes to the project.

4.2 Stack Overflow

Stack Overflow is the largest online community for programmers, where users ask and answer programming questions. Questions and answers can be edited and voted up and down.

I have previously commented that during the development of the project I used Stack Overflow to solve some of my doubts. So I decided to create an account (ana06) and contribute to the *Stack Overflow* community with the things I was learning about *Android* and *MongoDB*. Apart from voting up (when I earn enough reputation to do it) the answers that worked for me, I answered questions and edited answers with mistakes. Some of my best contributions were:

- **Button style not working for some versions of *Android*** I found a different solution for this problem. I posted in two different pages and some users found it useful.

  http://stackoverflow.com/questions/29860906/widget-appcompat-button-colorbuttonnormal-shows-gray

  http://stackoverflow.com/questions/29882292/buttonstyle-not-working-for-22-1-1

- **Authentication on *MongoDB* 3.0.5 with *Java Driver 3.x*** Because of the problem to authenticate on MongoDB 3.2, I need to use authentication mechanism to the previous one (*MONGODB-CR*) instead the default new one in releases 3.* (*SCRAM-SHA-1*). I found an almost perfect soluction to do it, so I edited it to correct the mistake it had.

  http://stackoverflow.com/questions/32019778/authentication-on-mongodb-3-0-5-with-java-driver-3-0-3-and-gridfs

5 Conclusion

I did not know almost anything about Big Data before starting this project and I have to say that learning about it has been truly rewarding. Although I have only learnt about Streaming Spark and Neural Networks in a theoretical way, I have found those two topics really appealing and I will surely delve into them in the future.

Learning about NoSQL databases and using MongoDB in a real project has been really interesting as I have discovered a new paradigm quite difference to the one in SQL databases which I was used to working with. In fact, when starting reading about documental databases I though that storing data that way was confusing and inhabitual, but after designing and working with a database of this kind during several months I have realised that it is more natural and closer to the information and data structures we have in our programs. I have also noticed that SQL and NoSQL databases can be combined to make the most of the advantages of both technologies, as I used both satisfactory in the mobile application.

Designing, developing and testing a whole mobile application on my own has been a enriching experience too. I love Android and learning how Android applications are developed is something I wanted to learn since a lot of time ago. Having the opportunity to do it while developing an application that not only is going to be used in real life, but also may help to improve the treatment of the bipolar disorder which affects an important percentage of people and their families, is amazing.

Apart from everything I have learnt, I feel proud of having contributed to the Github and Stackoverflow communities and I hope that this will make other people learning a little bit easier.

Acronyms

.apk Android Application Package. 42, 63

ADT Android Development Tools. 49

API Application Programming Interface. 11, 49, 50

BSON Binary JSON. 4, 6, 26

DICOM Digital Imaging and Communication in Medicine. 6

EEAG Escala de Evaluación de la Actividad Global. 30

GAF Global Assessment of Functioning. 29

HDFS Hadoop Distributed File System. 12

HDRS Hamilton Depression Rating Scale. 29, 30

I/O input/output. 7

IBM International Business Machines Corporation. 7, 57

ICD-10 10th revision of the International Statistical Classification of Diseases and Related Health Problems. 27

IDE Integrated development environment. 49

IP Internet Protocol. 24, 25, 33

JSON JavaScript Object Notation. 4, 6, 20, 35, 66

NoSQL Non SQL or not only SQL. vi, 2, 3, 21, 22, 65

ODM Object Document Mapper. 40

OO Object oriented. 1

OS Operating System. 24

PANSS Positive and Negative Syndrome Scale. 29, 30, 40

PSSUQ Post-Study System Usability Questionnaire. 57, 94

RAM Random Access Memory. 9

RDD Resilient Distributed Datasets. v, 7, 9–12

SQL Structured Query Language. 1–4, 6, 11, 20, 65, 66

TIBCO The Information Bus Company. 7

UC University of California. 7

UCM Universidad Complutense de Madrid. 1, 25

UTAUT Unified theory of acceptance and use of technology. 57

UTC Coordinated Universal Time. 52

XML eXtensible Markup Language. 4

YMRS Young Mania Rating Scale. 29, 30

Glossary

cluster set of connected computers (nodes) that work together so that, in many respects, they can be viewed as a single system. 7, 9–12, 22

g-force a measurement of the type of acceleration that causes weight. 32

horizontally scalable vertical scalability is when only one node is improved. In some cases when this is not enough, horizontal scalability is used. It consists on adding more nodes. When adding more nodes is feasible, it is said that it is horizontally scalable. 2

lux (symbol: lx) is the SI unit of illuminance and luminous emittance, measuring luminous flux per unit area. 32

Bibliography

[1] Apache Spark official web page. URL http://spark.apache.org/.

[2] Lightening Fast Big Data Analytics using Apache Spark. URL http://www.slideshare.net/manishgforce/lightening-fast-big-data-analytics-using-apache-spark.

[3] *Streaming Big Data: Storm, Spark and Samza - DZone Big Data*. URL https://dzone.com/articles/streaming-big-data-storm-spark.

[4] Code style for contributors. URL http://source.android.com/source/code-style.html.

[5] Index of /apt/debian/dists/wheezy/mongodb-org/3.2/main/binary-amd64. URL http://repo.mongodb.org/apt/debian/dists/wheezy/mongodb-org/3.2/main/binary-amd64/.

[6] Android developers guide. URL http://developer.android.com/intl/es/guide/index.html.

[7] Escala de Evaluación de la Actividad Global - EEAG. URL http://salpub.uv.es/SALPUB/practicum12/docs/visidom/Escalas+Instrum_valoracion_atencion_domiciliaria/129_ESCALA_EVALUACION_ACTIVIDAD_GLOBAL_EEAG.pdf.

[8] *Slides: Paradigmas del procesamiento en Big-Data*. URL https://sites.google.com/site/tecnologiaucmgtec/downloads.

[9] *The ham-d scale*. URL http://serene.me.uk/tests/ham-d.pdf.

[10] *M101J: MongoDB for Java Developers course*. . URL https://university.mongodb.com/courses/M101J/about.

[11] Mongodb documentation, . URL https://docs.mongodb.com.

[12] *Neural Network Learning in Big Data*. URL http://www.journals.elsevier.com/neural-networks/call-for-papers/special-issue-on-neural-network-learning-in-big-data.

[13] Contrato lou artículo 83 siguientes: Ref: 4155904 clínica nuestra señora de la paz, título: Predicción de crisis en el trastorno de bipolaridad, dirigido por victoria lópez lópez (ucm) y por diego urgelés (clínica nuestra señora de la paz).

[14] Stackoverflow. URL http://stackoverflow.com.

[15] Young Mania Rating Scale. URL psychology-tools.com/young-mania-rating-scale/.

[16] GENEActiv instruction manual v1.2, March 2012. URL http://1yn2me1hmr8p1dh4kt426lhe.wpengine.netdna-cdn.com/wp-content/uploads/2014/03/geneactiv_instruction_manual_v1.2.pdf.

[17] [Bug Watch] Stopping Apps On Android 4.4.2 Can Silently Kill Related Background Services, A Fix Is On The Way, March 2014. URL http://www.androidpolice.com/2014/03/07/bug-watch-stopping-apps-on-android-4-4-2-can-silently-kill-related-background-se

[18] Apache Spark vs Hadoop MapReduce, December 2015. URL http://www.edureka.co/blog/apache-spark-vs-hadoop-mapreduce.

[19] GNU General Public License v3.0, May 2016. URL http://choosealicense.com/licenses/gpl-3.0/.

[20] Positive and Negative Syndrome Scale, January 2016. URL https://en.wikipedia.org/w/index.php?title=Positive_and_Negative_Syndrome_Scale&oldid=699130814. Page Version ID: 699130814.

[21] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2012. ISBN 978-0387-31073-2.

[22] Enrique Martín. *Slides: Sistemas de Gestión de Datos y de la Información*. 2015-2016.

[23] Fabio Fumarola. Document Oriented Databases. URL http://es.slideshare.net/fabiofumarola1/9-document-oriented-databases.

[24] Google. Material design. URL https://www.google.com/design/spec/material-design.

[25] Jan Losby, Anne Wetmore. CDC Coffee Break: Using Likert Scales in Evaluation Survey Work. URL http://www.cdc.gov/dhdsp/pubs/docs/cb_february_14_2012.pdf.

[26] José Luis Sierra Rodríguez y Rubén Fuentes Fernández. *Slides: Programación de aplicaciones para dispositivos móviles*. 2015-2016.

[27] José María Sordo Juanena. *Slides: Programación iOS Y Android*. 2014.

[28] Justinmind. Justinmind support. URL http://www.justinmind.com/support.

[29] James R. Lewis. IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, January 1995. ISSN 1044-7318. doi: 10.1080/10447319509526110. URL http://dx.doi.org/10.1080/10447319509526110.

[30] MongoDB. Document Databases. URL https://www.mongodb.com/document-databases.

[31] Tim A. Majchrzak Oliver Schmitt. Using Document-Based Databases for Medical Information Systems in Unreliable Environments. 2012.

[32] Pablo Moreno Ger, Guillermo Jiménez Díaz, Antonio Sánchez Ruiz-Granados. *Apuntes de la asignatura Desarrollo de Sistemas Interactivos*. 2015/2016.

[33] Simón J. Rascovsky, Jorge A. Delgado, Alexander Sanz, Víctor D. Calvo, and Gabriel Castrillón. Informatics in Radiology: Use of CouchDB for Document-based Storage of DICOM Objects. *RadioGraphics*, 32(3):913–927, May 2012. ISSN 0271-5333. doi: 10.1148/rg.323115049. URL http://pubs.rsna.org/doi/abs/10.1148/rg.323115049.

[34] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley, August 2012. ISBN 978-0-13-303612-1.

[35] Raffael Vogler. MongoDB – State of the R, November 2014. URL http://www.joyofdata.de/blog/mongodb-state-of-the-r-rmongodb/.

[36] Welch Labs. Neural Networks Demystified. URL https://www.youtube.com/watch?v=bxe2T-V8XRs.

Appendix: MongoDB configuration file

Those are the file *mongod.cfg* used for *MongoDB* configuration in both Windows and Debian. Be careful to use spaces and not tabs or it will not work.

Windows

```
systemLog:
    destination: file
    path: c:\data\log\mongod.log
storage:
    dbPath: c:\data\db
security:
        authorization: enabled
```

Debian

```
# mongod.conf

# for documentation of all options, see:
# http://docs.mongodb.org/manual/reference/configuration-options/

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# Where and how to store data.
storage:
  dbPath: /var/lib/mongo
  journal:
    enabled: true
```

```
#processManagement:

# network interfaces
net:
  port: 8080

#security:

#operationProfiling:

#replication:

#sharding:

## Enterprise-Only Options

#auditLog:

#snmp:
```

Appendix: Actigraph data

This appendix contain the first 99 lines of the *.bin* generated by the *GENEActive* actigraph which is wearing one of the patients without any processing. Some lines are too long and do not include usefull information for my project and I have shorted them indicating it with [...].

```
Device Identity
Device Unique Serial Code:029874
Device Type:GENEActiv
Device Model:1.1
Device Firmware Version:Ver4.08a date14Jul14
Calibration Date:2015-05-27 10:19:14:000

Device Capabilities
Accelerometer Range:-8 to 8
Accelerometer Resolution:0.0039
Accelerometer Units:g
Light Meter Range:0 to 5000
Light Meter Resolution:5
Light Meter Units:lux
Temperature Sensor Range:0 to 70
Temperature Sensor Resolution:0.1
Temperature Sensor Units:deg. C

Configuration Info
Measurement Frequency:10 Hz
Measurement Period:1440 Hours
Start Time:2016-03-04 17:14:33:000
Time Zone:GMT +01:00

Trial Info
Study Centre:CNSDLP
Study Code:
Investigator ID:
Exercise Type:
Config Operator ID:DUP
Config Time:2016-03-04 15:55:35:410
Config Notes:Notes
```

```
Extract Operator ID:
Extract Time:2016-04-01 16:13:26:657
Extract Notes:(device clock drift -3.553,657s)

Subject Info
Device Location Code:left wrist
Subject Code:ALPA
Date of Birth:1900-1-1
Sex:female
Height:
Weight:
Handedness Code:
Subject Notes:

Calibration Data
x gain:25304
x offset:-168
y gain:25343
y offset:-663
z gain:25114
z offset:-313
Volts:54
Lux:989

Memory Status
Number of Pages:535

Recorded Data
Device Unique Serial Code:029874
Sequence Number:0
Page Time:2016-03-04 17:14:38:000
Unassigned:
Temperature:36.7
Battery voltage:4.1493
Device Status:Recording
Measurement Frequency:10.0
06000BF2F040073009F2A040072001F25040077006F2904407F00BF[...]
Recorded Data
Device Unique Serial Code:029874
Sequence Number:1
Page Time:2016-03-04 17:15:08:000
Unassigned:
Temperature:35.9
Battery voltage:4.1493
Device Status:Recording
```

```
Measurement Frequency:10.0
EE6FB9FCE000F24FABFCF000EF9FD2FDA000F1D01EFFB000F120140[...]
Recorded Data
Device Unique Serial Code:029874
Sequence Number:2
Page Time:2016-03-04 17:15:38:000
Unassigned:
Temperature:35.4
Battery voltage:4.1493
Device Status:Recording
Measurement Frequency:10.0
FB0F00FEA000FABF07FE4000FA6EF9031000FACF0AFE8000FAAF17F[...]
Recorded Data
Device Unique Serial Code:029874
Sequence Number:3
Page Time:2016-03-04 17:16:08:000
Unassigned:
Temperature:35.1
Battery voltage:4.1493
Device Status:Recording
Measurement Frequency:10.0
F87EDEFC7000F5CEB3FED000F41F2AFE9000F43F54FE3000F2EF33F[...]
```

Appendix: Creation of users and roles

This appendix includes the commands needed to create, find and delete the users with the roles explained in 2.3.2.2.

To create the roles:

```
use bipolarDatabase

db.createRole(
  {
    role: "mobile",
    privileges:
      [
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "users"
            },
          actions: [ "find", "update", "insert" ]
        },
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "mobileTests"
            },
          actions: [ "find", "update", "insert" ]
        },
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "comments"
            },
```

```
          actions: [ "find" ]
        }
      ],
    roles: [ ]
  },
  { w: "majority" , wtimeout: 3000 }
)

db.createRole(
  {
    role: "actigraph",
    privileges:
      [
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "actigraphData"
            },
          actions: [ "insert" ]
        }
      ],
    roles: [ ]
  },
  { w: "majority" , wtimeout: 3000 }
)

db.createRole(
  {
    role: "web",
    privileges:
      [
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "users"
            },
          actions: [ "find", "update", "insert" ] },
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "comments"
            },
```

```
          actions: [ "find", "update", "insert", "remove" ]
        },
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "records"
            },
          actions: [ "find", "update", "insert" ]
        },
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "analysis"
            },
          actions: [ "find" ]
        }
      ],
      roles: [ ]
   },
   { w: "majority" , wtimeout: 3000 }
)

db.createRole(
  {
    role: "analysis",
    privileges:
      [
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "users"
            },
          actions: [ "find" ]
        },
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "mobileTests"
            },
          actions: [ "find" ]
        },
```

```
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "comments"
            },
          actions: [ "find" ]
        },
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "records"
            },
          actions: [ "find" ]
        },
        {
          resource:
            {
              db: "bipolarDatabase",
              collection: "analysis"
            },
          actions: [ "find", "update", "insert" ]
        }
      ],
      roles: [ ]
    },
    { w: "majority" , wtimeout: 3000 }
)
```

To see the roles:

```
db.getRoles()
```

To remove all roles from the database:

```
db.runCommand(
  { dropAllRolesFromDatabase: 1,
    writeConcern: { w: "majority" }
  }
)
```

To create the users:

```
db.createUser( { "user" : "androidUser",
                 "pwd": "password",
                 "roles" : [ "mobile" ] }
)

db.createUser( { "user" : "actigraphUser",
                 "pwd": "password2",
                 "roles" : [ "mobile" ] }
)


db.createUser( { "user" : "webUser",
                 "pwd": "password3",
                 "roles" : [ "mobile" ] }
)

db.createUser( { "user" : "analyst",
                 "pwd": "password4",
                 "roles" : [ "analysis" ] }
)


db.createUser( { "user" : "admin",
                 "pwd": "password5",
                 "roles" : [ "dbOwner" ] }
)
```

To see the users:

```
db.getUsers()
```

To eliminate a user (for example `androidUser`):

```
db.dropUser( "androidUser" )
```

To remove all user from the database:

```
db.runCommand(
  { dropAllUsersFromDatabase: 1,
```

```
    writeConcern: { w: "majority" }
  }
)
```

Appendix: Prototypes images

The following images of the different prototypes (2 or 3 captures of every prototype, except for the last one, for which 6 captures are shown) pretend to underline the main difference between every prototype and to make easy to understand the process followed to prototype. You can read more about the prototyping in section 3.2.2.1.

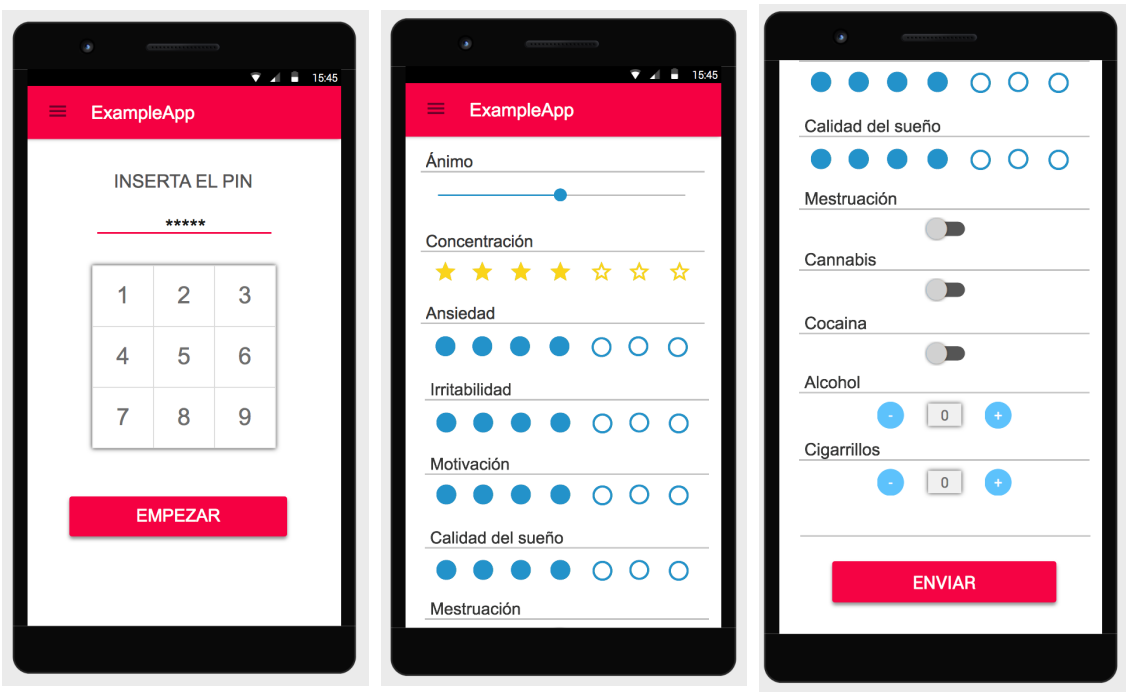


Figure 1: Fist prototype

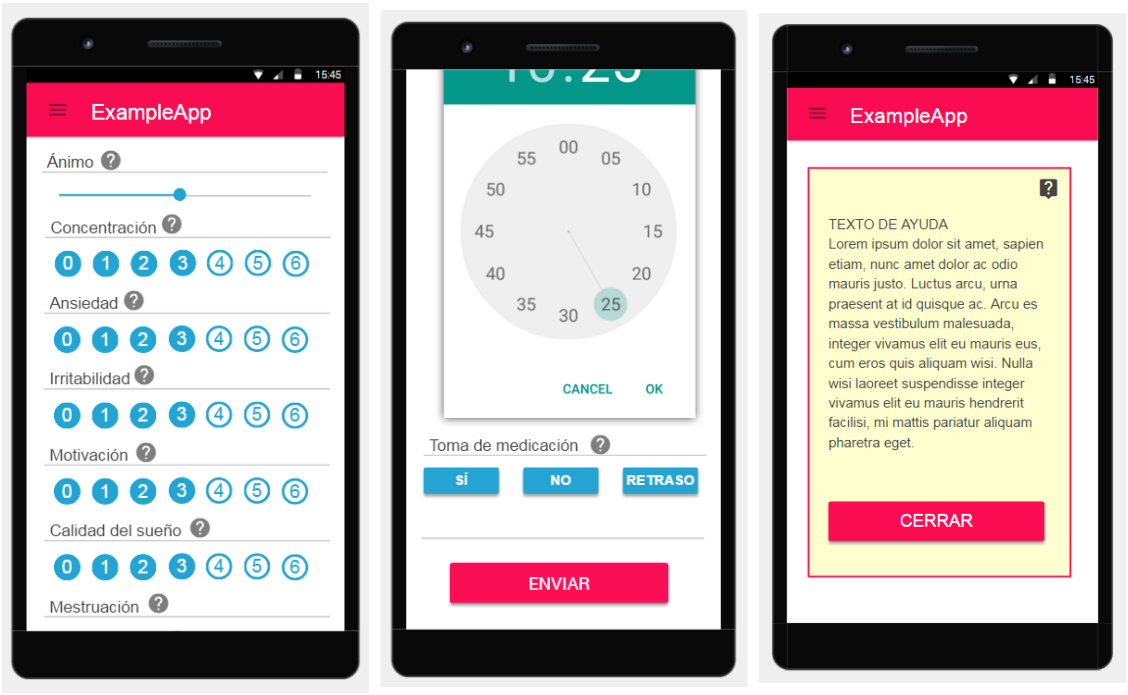


Figure 2: Second prototype

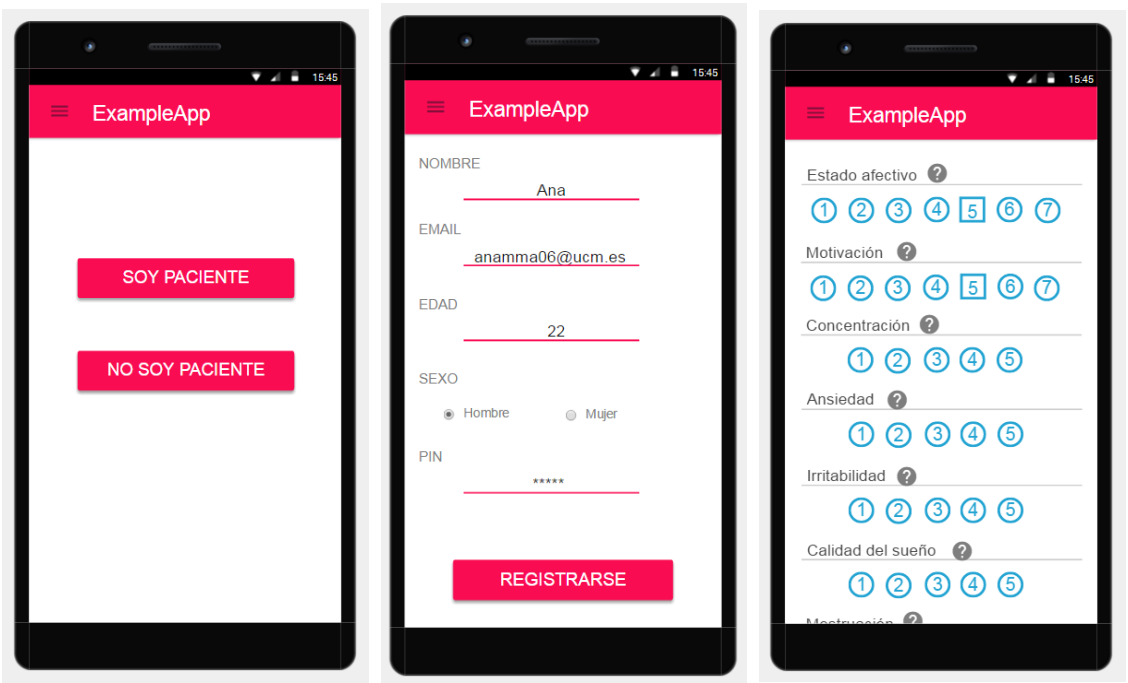


Figure 3: Third prototype

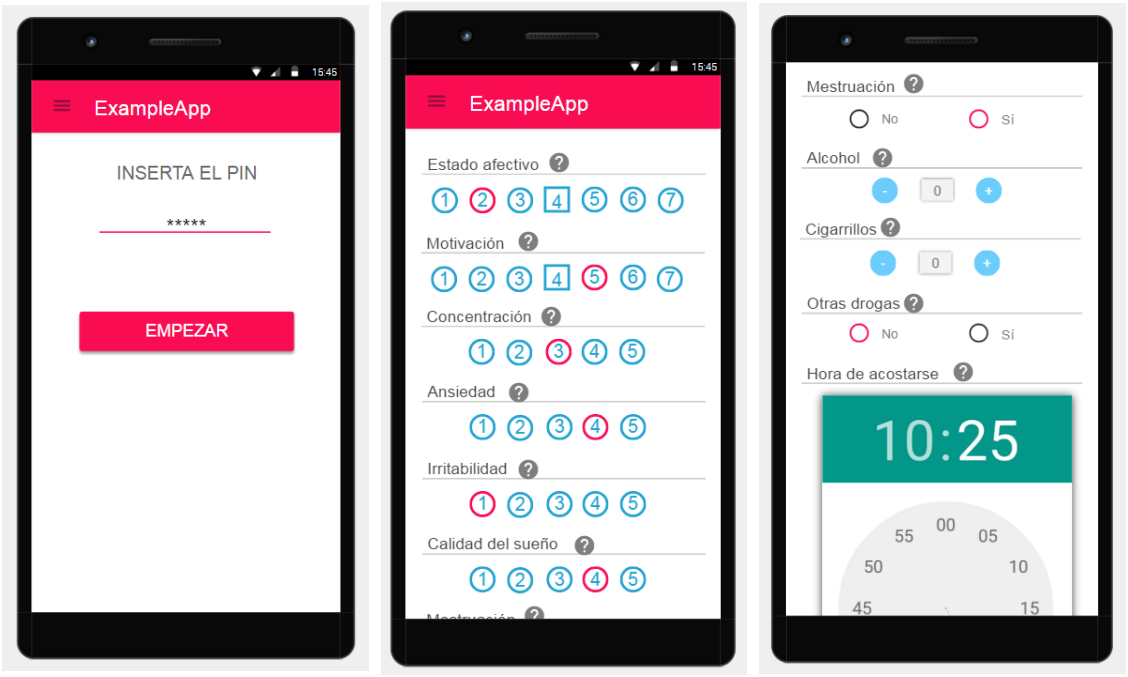


Figure 4: Fourth prototype

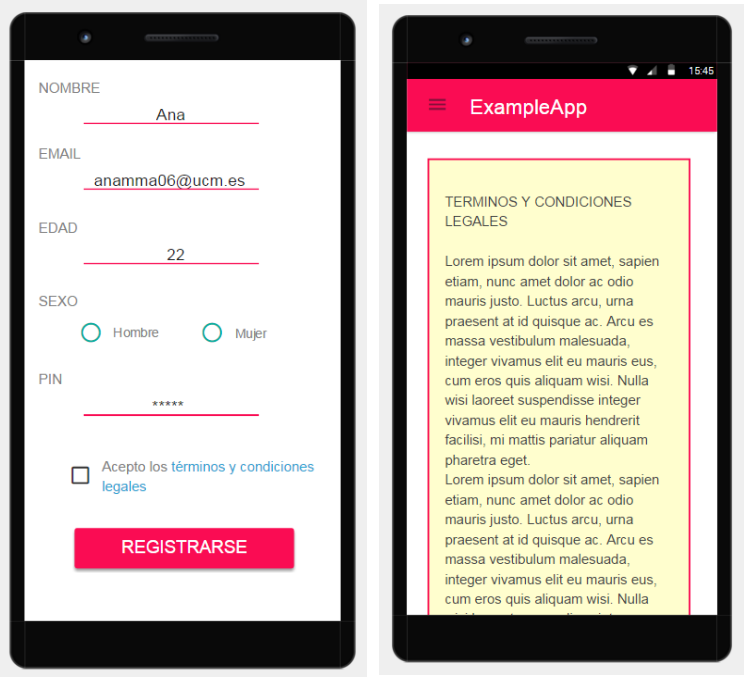


Figure 5: Fifth prototype

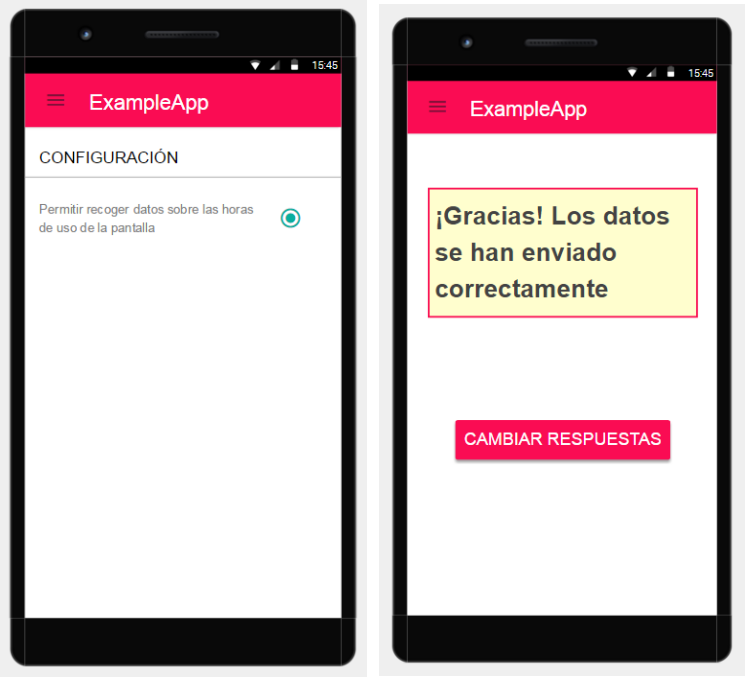


Figure 6: Sixth prototype

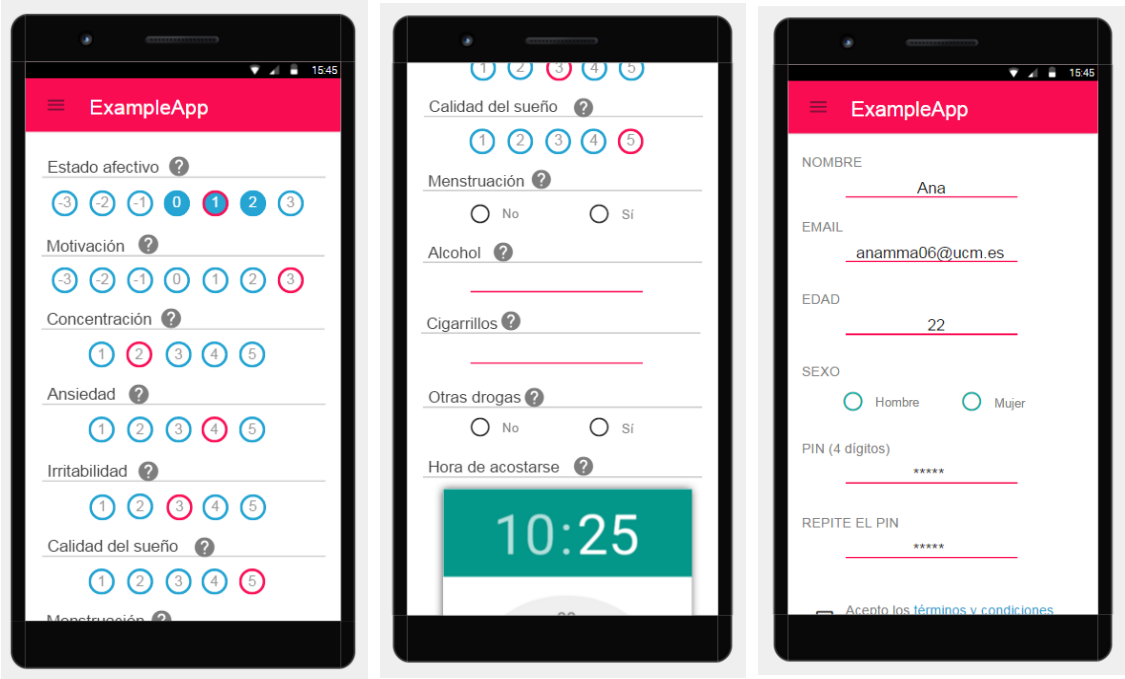


Figure 7: Seventh prototype

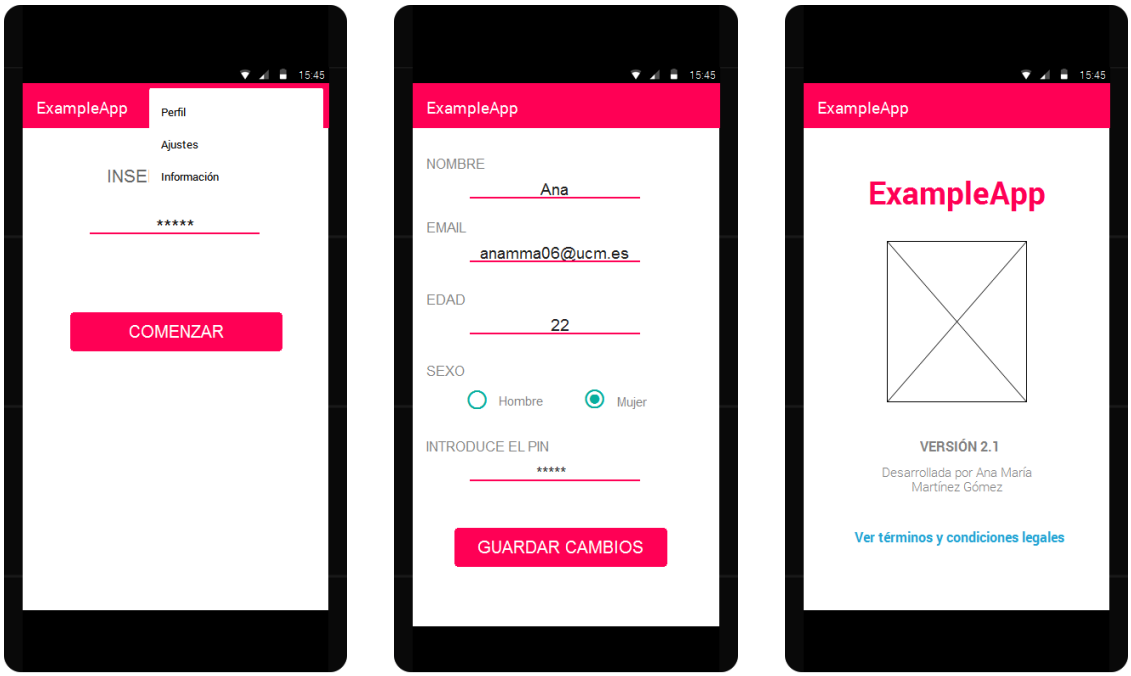


Figure 8: Eight prototype

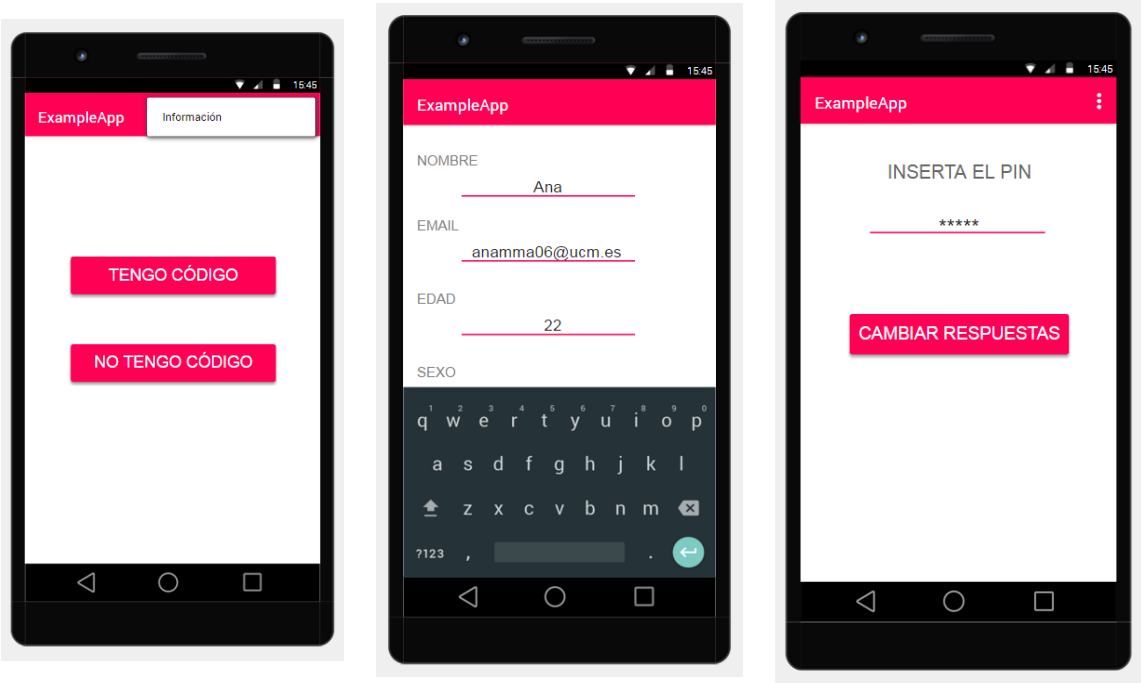


Figure 9: Ninth prototype

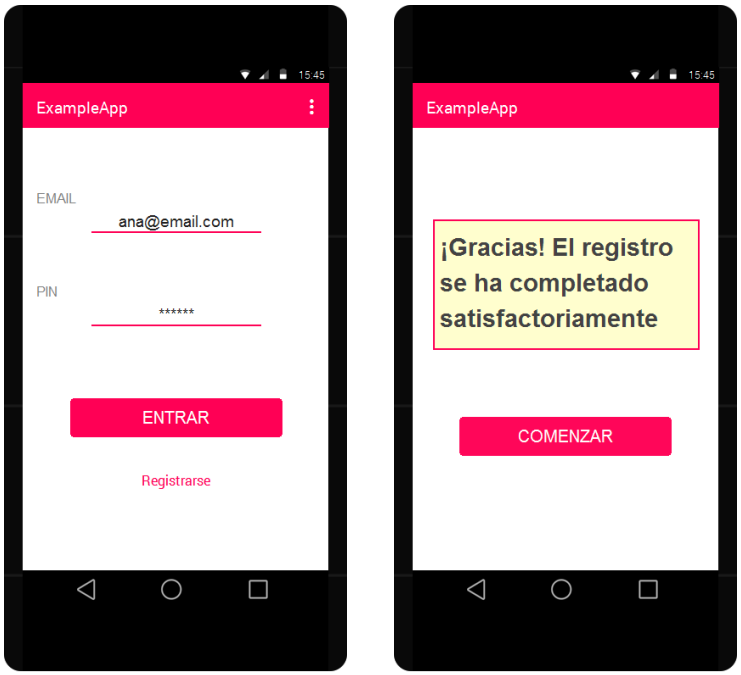


Figure 10: Tenth prototype

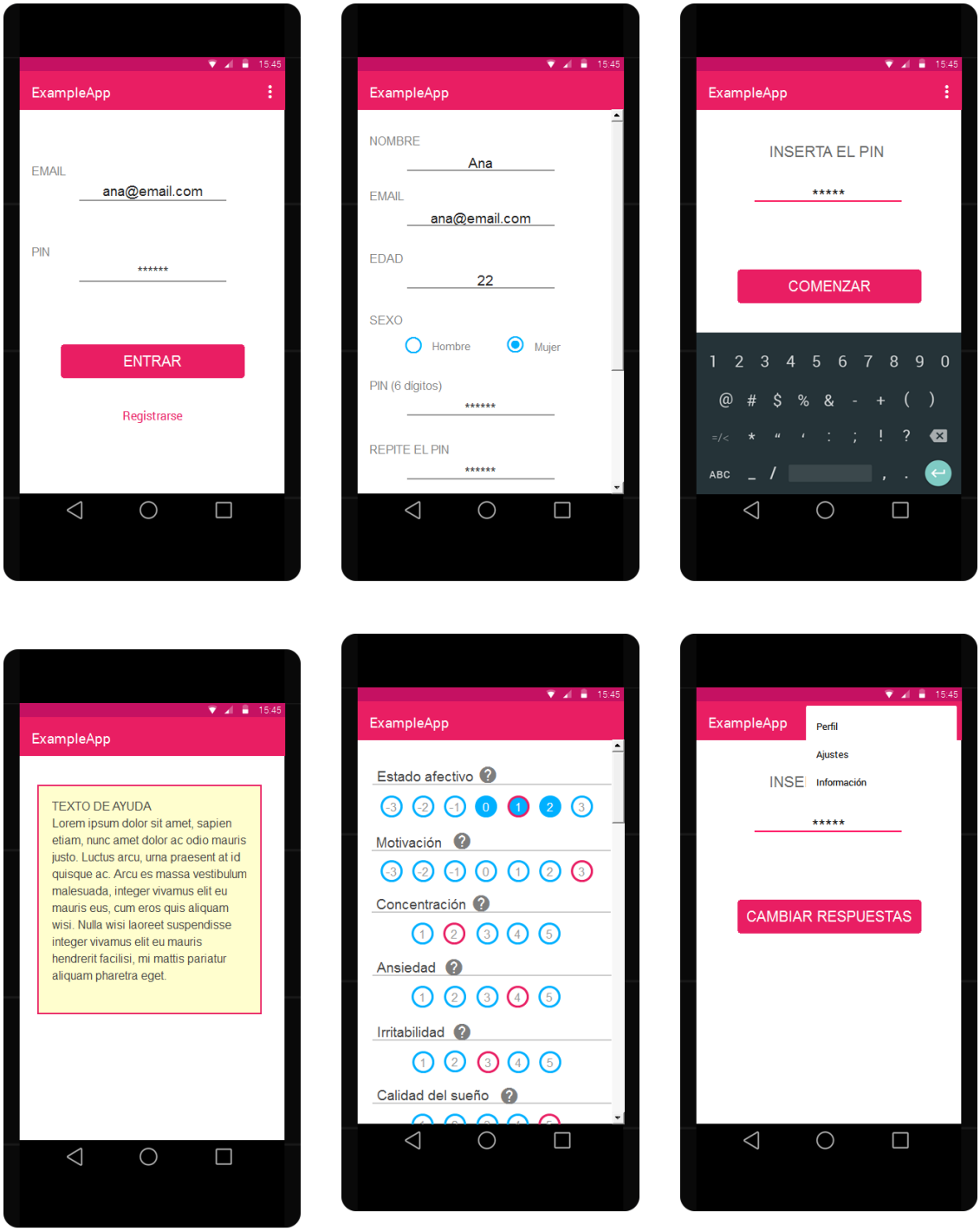
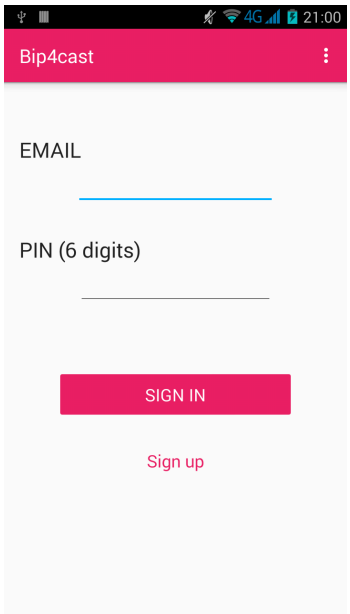


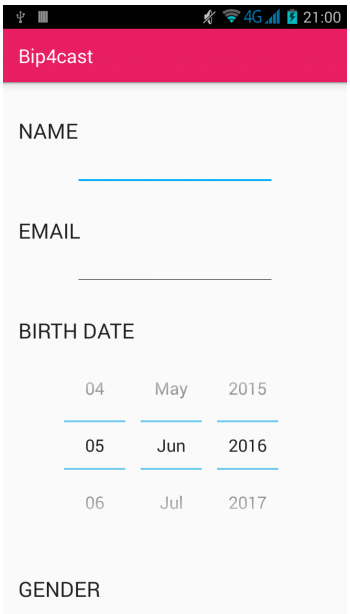
Figure 11: Final prototype

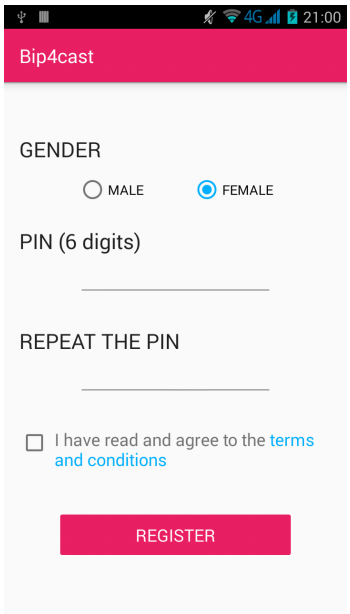
Appendix: Screenshots of the final application

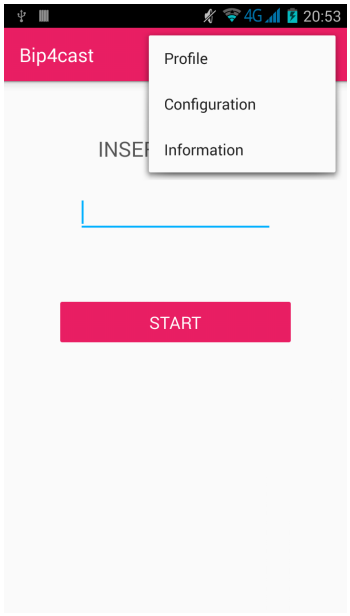
Elephone P3000S with Android 4.4.2

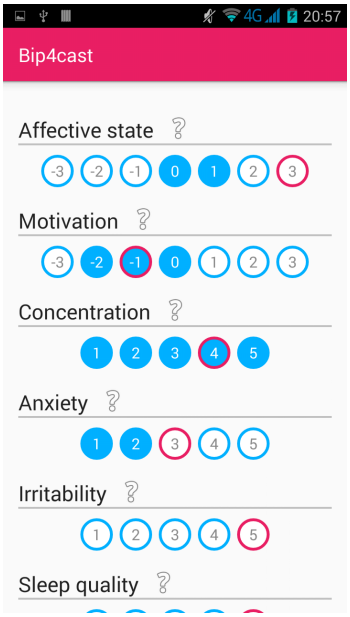
In this section you can see how the application looks in my *Elephone P3000S with Android 4.4.2*.

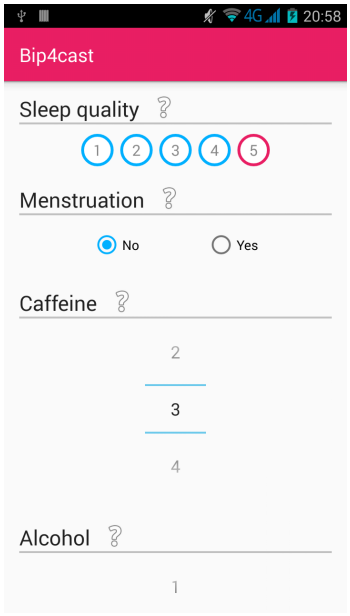


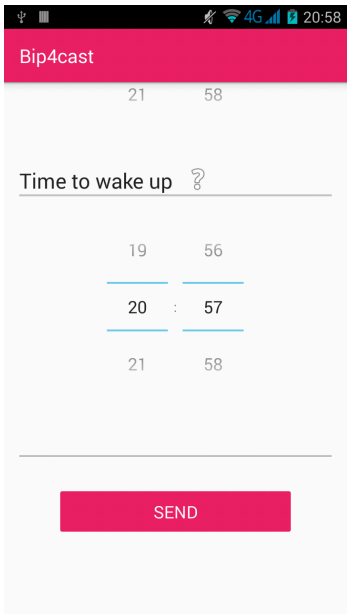


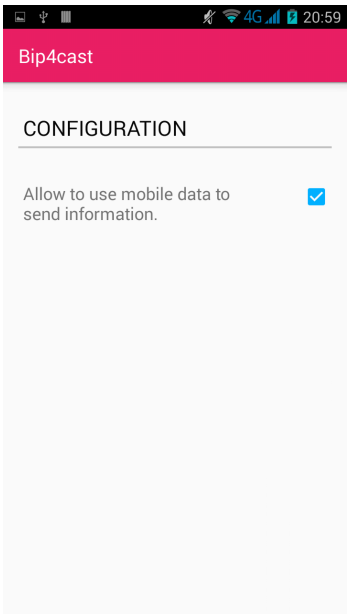


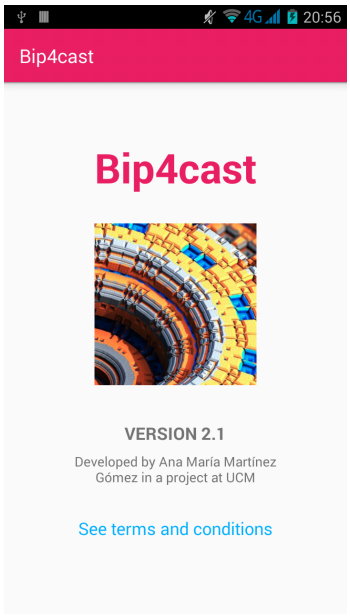

20:53
Bip4cast
Profile
Configuration
Information
INSE
START


20:57
Bip4cast
Affective state
-3 -2 -1 0 1 2 3
Motivation
-3 -2 -1 0 1 2 3
Concentration
1 2 3 4 5
Anxiety
1 2 3 4 5
Irritability
1 2 3 4 5
Sleep quality


20:58
Bip4cast
Sleep quality
1 2 3 4 5
Menstruation
No
Yes
Caffeine
2
3
4
Alcohol
1

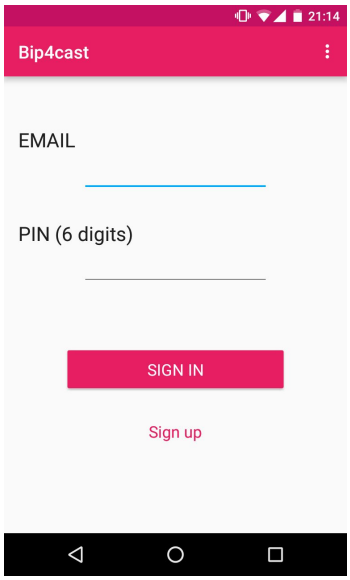

20:58
Bip4cast
21 58
Time to wake up
19 56
20 : 57
21 58
SEND

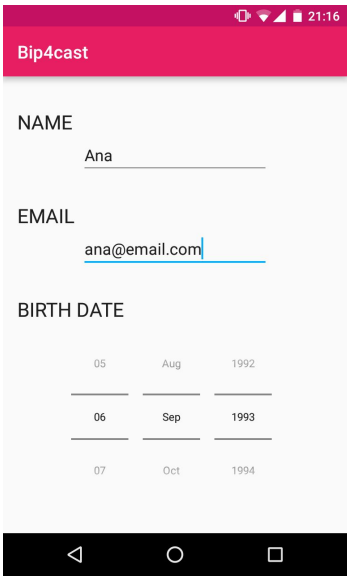

20:59
Bip4cast
CONFIGURATION
Allow to use mobile data to send information.

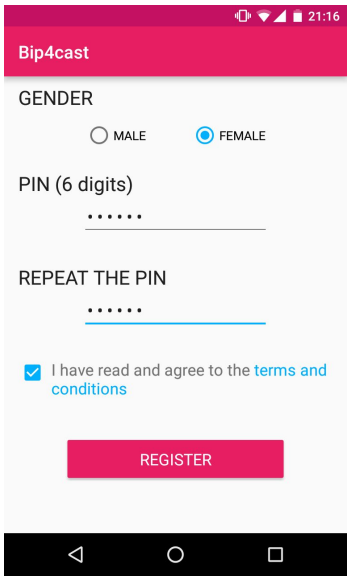

20:56
Bip4cast
Bip4cast
VERSION 2.1
Developed by Ana María Martínez Gómez in a project at UCM
See terms and conditions

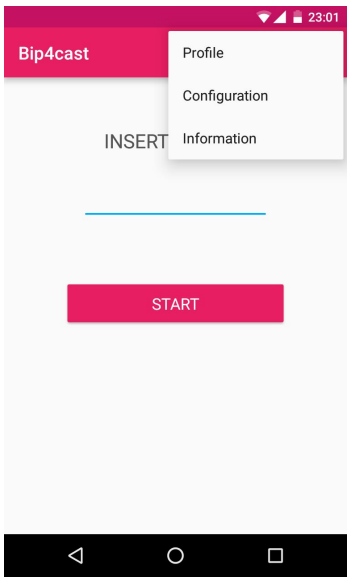
Nexus 4 with Android 6.0.1

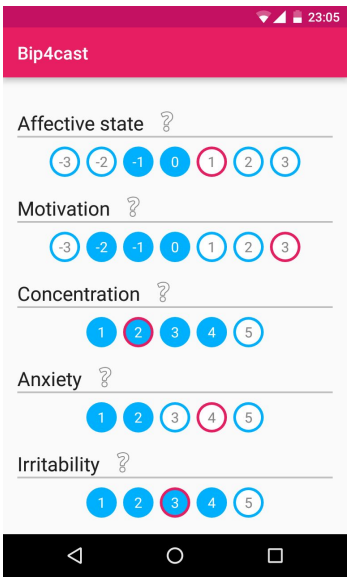
In this section you can see how the application looks in the *Nexus 4 with Android 6.0.1*, a newer version which uses *Material Design*.

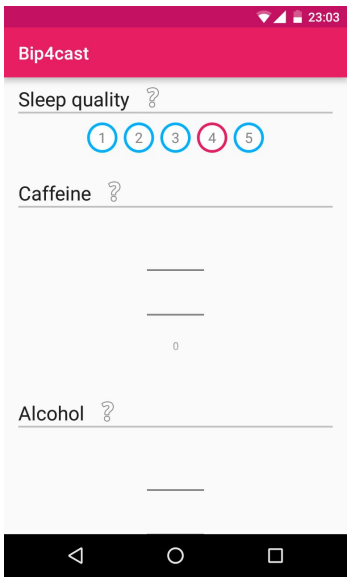


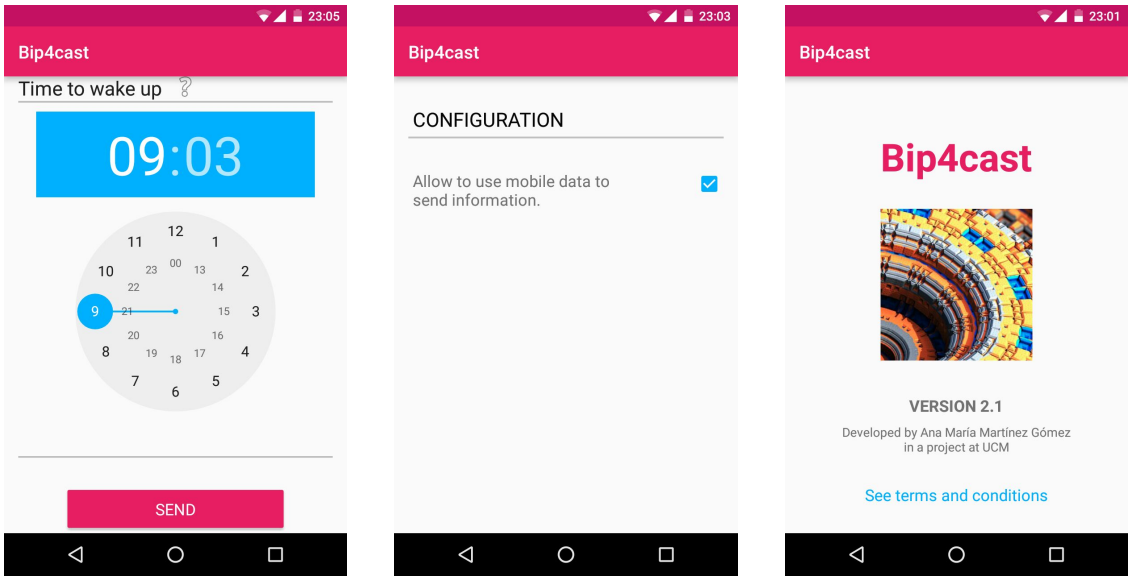











23:05
Bip4cast
Time to wake up
09:03
SEND
23:03
Bip4cast
CONFIGURATION
Allow to use mobile data to send information.
23:01
Bip4cast
Bip4cast
VERSION 2.1
Developed by Ana María Martínez Gómez in a project at UCM
See terms and conditions

Appendix: Usability testing questionnaire

The questionnaire I used for the usability testing is based on the PSSUQ. More information about this questionnaire can be found at [29]. The questions of my questionnaire, which was in Spanish as users where native Spanish speakers, were:

1. En general, estoy contento con lo fácil que resulta usar la applicación.

   1 2 3 4 5 6 7

   Comentarios:

2. Usar la aplicación es sencillo.

   1 2 3 4 5 6 7

   Comentarios:

3. Puedo proporcionar información sobre mi estado afectivo de forma efectiva con esta aplicación.

   1 2 3 4 5 6 7

   Comentarios:

4. Puedo proporcionar información sobre mi estado afectivo rápido con esta aplicación.

   1 2 3 4 5 6 7

   Comentarios:

5. Puedo proporcionar información sobre mi estado afectivo de forma eficiente con esta aplicación.

   1 2 3 4 5 6 7

   Comentarios:

6. Me siento cómodo usando la aplicación.

   1 2 3 4 5 6 7

   Comentarios:

7. Es fácil aprender a usar la aplicación.

   1 2 3 4 5 6 7

Comentarios:

8. Creo que podré hacer uso de la aplicación de forma productiva en un corto periodo de tiempo.

   1 2 3 4 5 6 7

   Comentarios:

9. Los mensajes de error que aparecen en la aplicación me indican de forma clara como solucionar los problemas.

   1 2 3 4 5 6 7

   Comentarios:

10. Cuando comento un error usando la aplicación, es fácil y rápido recuperarse de él.

    1 2 3 4 5 6 7

    Comentarios:

11. La información proporcionada en la aplicación es clara.

    1 2 3 4 5 6 7

    Comentarios:

12. Es fácil encontrar la información que necesito.

    1 2 3 4 5 6 7

    Comentarios:

13. La información proporcionada en la aplicación es fácil de entender.

    1 2 3 4 5 6 7

    Comentarios:

14. La información proporcionada me ayuda a proporcionar información sobre mi estado afectivo.

    1 2 3 4 5 6 7

    Comentarios:

15. La forma de organizar la información en las pantallas de la aplicación es clara.

    1 2 3 4 5 6 7

    Comentarios:

16. La interfaz de la aplicación es agradable.

    1 2 3 4 5 6 7

    Comentarios:

17. Me gusta utilizar la interfaz de la aplicación.

1 2 3 4 5 6 7

Comentarios:

18. La aplicación tiene todas las caracteristicas y opciones que esperaba que tuviese.

1 2 3 4 5 6 7

Comentarios:

19. En general, estoy satisfecho con la aplicación.

1 2 3 4 5 6 7

Comentarios:

Answered questionnaires

The users answers were the followings.

User 1

usuario 1

Questionario de usabilidad

1. En general, estoy contento con lo fácil que resulta usar la application.

   1 2 3 4 5 (6) 7

   Comentarios:

2. Usar la aplicación es sencillo.

   1 2 3 4 5 6 (7)

   Comentarios:

3. Puedo proporcionar información sobre mi estado afectivo de forma efectiva con esta aplicación.

   1 2 3 (4) 5 6 7

   Comentarios: Podría ser necesario hacer algun comentario de por que marcas

4. Puedo proporcionar información sobre mi estado afectivo rápido con esta aplicación.

   1 2 3 4 5 6 (7)

   Comentarios:

5. Puedo proporcionar información sobre mi estado afectivo de forma eficiente con esta aplicación.

   1 2 3 4 5 (6) 7

   Comentarios:

6. Me siento cómodo usando la aplicación.

   1 2 3 4 5 (6) 7

   Comentarios:

7. Es fácil aprender a usar la aplicación.

   1 2 3 4 5 6 (7)

   Comentarios:

8. Creo que podré hacer uso de la aplicación de forma productiva en un corto periodo de tiempo.

1 2 3 4 5 6 7

Comentarios:

9. Los errores que aparecen en la aplicación me indican de forma clara como solucionar los problemas.

1 2 3 4 5 6 7

Comentarios:

10. Cuando comento un error usando la aplicación, es fácil y rápido recuperarse de él.

1 2 3 4 5 6 7

Comentarios:

11. La información proporcionada en la aplicación es clara.

1 2 3 4 5 6 7

Comentarios:

12. Es fácil encontrar la información que necesito.

1 2 3 4 5 6 7

Comentarios:

13. La información proporcionada en la aplicación es fácil de entender.

1 2 3 4 5 6 7

Comentarios:

14. La información proporcionada me ayuda a proporcionar información sobre mi estado afectivo.

1 2 3 4 5 6 7

Comentarios:

15. La forma de organizar la información en las pantallas de la aplicación es clara.

1 2 3 4 5 6 7

Comentarios:

16. La interfaz de la aplicación es agradable.

1 2 3 4 5 6 7

Comentarios:

17. Me gusta utilizar la interfaz de la aplicación.

1 2 3 4 5 6 7

Comentarios:

18. La aplicación tiene todas las caracteristicas y opciones que esperaba que tuviese.

    1 2 3 4 5 6 (7)

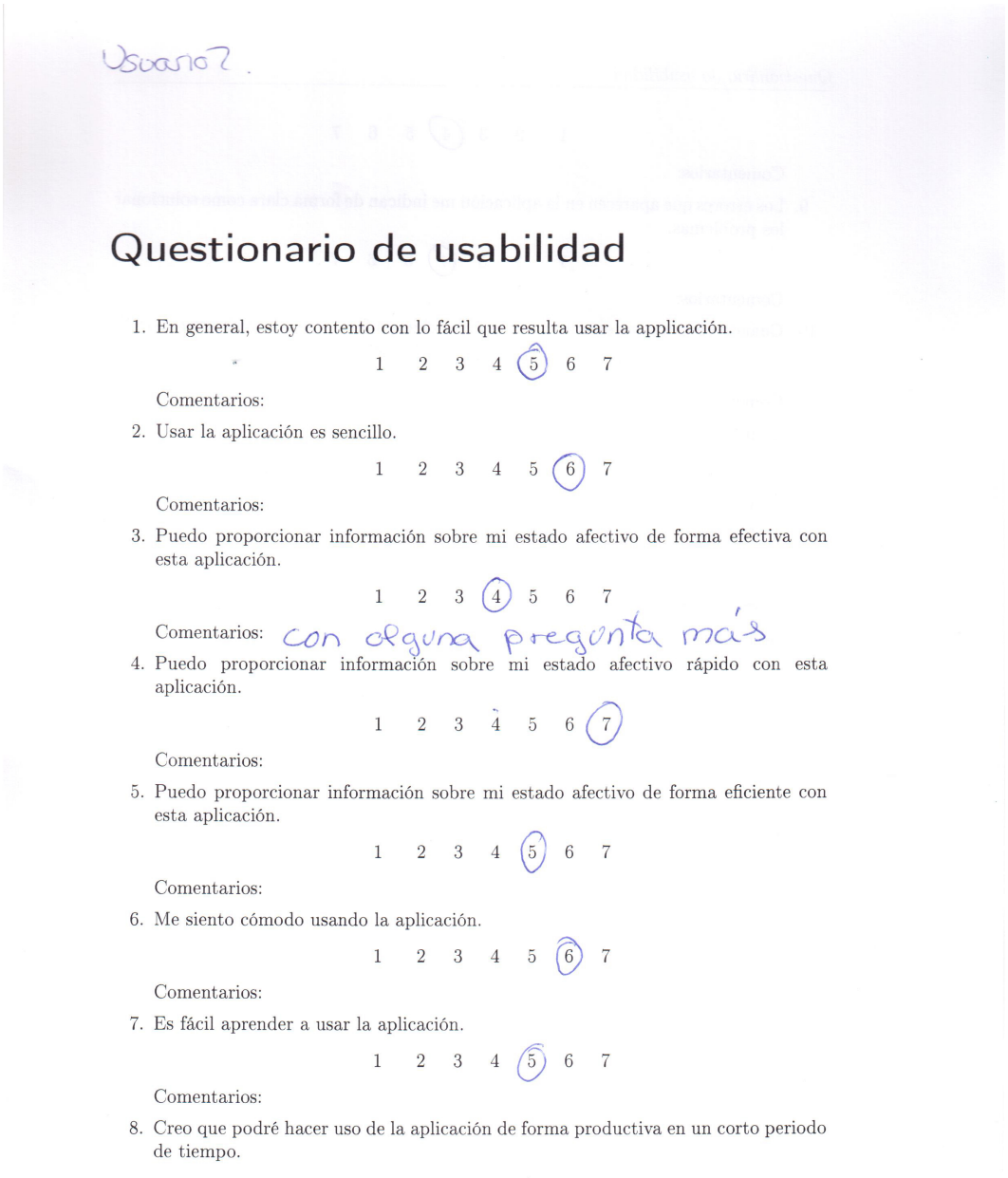
    Comentarios:

19. En general, estoy satisfecho con la aplicación.

    1 2 3 4 5 (6) 7

    Comentarios:

User 2



Usuario 7

Questionario de usabilidad

1. En general, estoy contento con lo fácil que resulta usar la applicación.

   1 2 3 4 (5) 6 7

   Comentarios:

2. Usar la aplicación es sencillo.

   1 2 3 4 5 (6) 7

   Comentarios:

3. Puedo proporcionar información sobre mi estado afectivo de forma efectiva con esta aplicación.

   1 2 3 (4) 5 6 7

   Comentarios: con alguna pregunta más

4. Puedo proporcionar información sobre mi estado afectivo rápido con esta aplicación.

   1 2 3 4 5 6 (7)

   Comentarios:

5. Puedo proporcionar información sobre mi estado afectivo de forma eficiente con esta aplicación.

   1 2 3 4 (5) 6 7

   Comentarios:

6. Me siento cómodo usando la aplicación.

   1 2 3 4 5 (6) 7

   Comentarios:

7. Es fácil aprender a usar la aplicación.

   1 2 3 4 (5) 6 7

   Comentarios:

8. Creo que podré hacer uso de la aplicación de forma productiva en un corto periodo de tiempo.

Questionario de usabilidad

1 2 3 (4) 5 6 7

Comentarios:

9. Los errores que aparecen en la aplicación me indican de forma clara como solucionar los problemas.

1 2 3 (4) 5 6 7

Comentarios:

10. Cuando comento un error usando la aplicación, es fácil y rápido recuperarse de él.

1 2 3 4 5 (6) 7

Comentarios:

11. La información proporcionada en la aplicación es clara.

1 2 3 4 5 (6) 7

Comentarios:

12. Es fácil encontrar la información que necesito.

1 2 3 4 (5) 6 7

Comentarios:

13. La información proporcionada en la aplicación es fácil de entender.

1 2 3 4 5 (6) 7

Comentarios:

14. La información proporcionada me ayuda a proporcionar información sobre mi estado afectivo.

1 (2) 3 4 5 6 7

Comentarios:

15. La forma de organizar la información en las pantallas de la aplicación es clara.

1 2 3 4 (5) 6 7

Comentarios:

16. La interfaz de la aplicación es agradable.

1 2 3 4 (5) 6 7

Comentarios:

17. Me gusta utilizar la interfaz de la aplicación.

1 2 3 4 (5) 6 7

Comentarios:

18. La aplicación tiene todas las caracteristicas y opciones que esperaba que tuviese.

1 2 (3) 4 5 6 7

Comentarios:

19. En general, estoy satisfecho con la aplicación.

1 2 (3) 4 5 6 7

Comentarios:

User 3

Questionario de usabilidad

USUARIO 3

1. En general, estoy contento con lo fácil que resulta usar la application.

   1 2 3 4 5 6 (7)

   Comentarios:

2. Usar la aplicación es sencillo.

   1 2 3 4 5 6 (7)

   Comentarios:

3. Puedo proporcionar información sobre mi estado afectivo de forma efectiva con esta aplicación.

   1 2 3 4 (5) 6 7

   Comentarios:

4. Puedo proporcionar información sobre mi estado afectivo rápido con esta aplicación.

   1 2 3 4 (5) 6 7

   Comentarios:

5. Puedo proporcionar información sobre mi estado afectivo de forma eficiente con esta aplicación.

   1 2 3 4 (5) 6 7

   Comentarios:

6. Me siento cómodo usando la aplicación.

   1 2 3 4 5 6 (7)

   Comentarios:

7. Es fácil aprender a usar la aplicación.

   1 2 3 4 5 6 (7)

   Comentarios:

8. Creo que podré hacer uso de la aplicación de forma productiva en un corto periodo de tiempo.

1 2 3 4 5 6 7

Comentarios:

9. Los errores que aparecen en la aplicación me indican de forma clara como solucionar los problemas.

1 2 3 4 5 6 7

Comentarios:

10. Cuando comento un error usando la aplicación, es fácil y rápido recuperarse de él.

1 2 3 4 5 6 7

Comentarios:

11. La información proporcionada en la aplicación es clara.

1 2 3 4 5 6 7

Comentarios:

12. Es fácil encontrar la información que necesito.

1 2 3 4 5 6 7

Comentarios:

13. La información proporcionada en la aplicación es fácil de entender.

1 2 3 4 5 6 7

Comentarios:

14. La información proporcionada me ayuda a proporcionar información sobre mi estado afectivo.

1 2 3 4 5 6 7

Comentarios:

15. La forma de organizar la información en las pantallas de la aplicación es clara.

1 2 3 4 5 6 7

Comentarios:

16. La interfaz de la aplicación es agradable.

1 2 3 4 5 6 7

Comentarios:

17. Me gusta utilizar la interfaz de la aplicación.

1 2 3 4 5 6 7

Comentarios:

18. La aplicación tiene todas las caracteristicas y opciones que esperaba que tuviese.

    1 2 3 4 (5) 6 7

    Comentarios: podría tener mas.

19. En general, estoy satisfecho con la aplicación.

    1 2 3 4 5 (6) 7

    Comentarios: