TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

EMOMOBILE

Automatic detection of emotional behaviour from daily user-mobile interactions

Autor

Francisco de Asís Lobón Villanueva

Director

Oresti Baños Legrán

Escuela Técnica Superior de Ingeniería y de Telecomunicación

—

Granada, 7 de septiembre de 2020

EMOMOBILE: Automatic detection of emotional behaviour from daily user-mobile interactions

Francisco de Asís Lobón Villanueva

Palabras clave: Android, Aplicación web, Servidor remoto, Teléfono inteligente, Aplicaciones para dispositivos móviles, Monitorización, Trastorno bipolar, Estado del ánimo.

Resumen

El trastorno bipolar es un trastorno en el estado de ánimo que necesita un seguimiento regular para conocer el estado de la persona que lo padece. En este trabajo se ha desarrollado un sistema compuesto por dos aplicaciones, una aplicación móvil y una aplicación web que permitan facilitar dicho seguimiento de forma remota, continua y ubicua. La aplicación móvil está dirigida a la persona que tiene el trastorno y será encargada de monitorizar al paciente a través de cuestionarios personalizados. La aplicación web está dirigida para un centro que albergará diferentes roles de usuario. El principal será el rol del administrador que será el encargado de gestionar el centro. En general esta aplicación será usada por los profesionales que se encargarán de hacer el seguimiento a las personas que usen la aplicación móvil. Además, los especialistas serán los que creen las preguntas y gestiónen los cuestionarios que realizarán los usuarios de la aplicación móvil. Por último, se ha realizado una evaluación inicial de la usabilidad del sistema con varios participantes.

EMOMOBILE: Automatic detection of emotional behaviour from daily user-mobile interactions

Francisco de Asís Lobón Villanueva

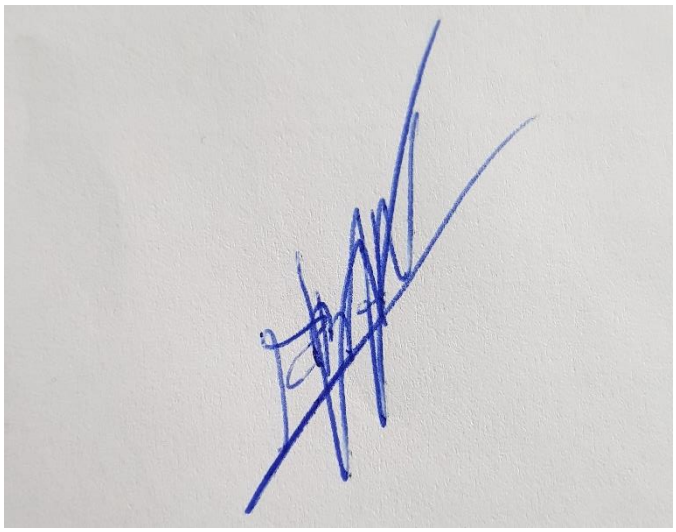
Keywords: Android, Web Application, Remote Server, Smartphone, Mobile Applications, Monitoring, Bipolar disorder, Mood disorder.

Abstract

Bipolar disorder is a mood disorder that needs regular monitoring to find out the status of the person who has it. In this work, a system has been developed consisting of two applications, a mobile application and a web application, which will facilitate such monitoring remotely, continuously and ubiquitously. The mobile application is directed to the person who has the disorder and will be in charge of monitoring the patient through personalised questionnaires. The web application is aimed at a centre that will have different user roles. The main one will be the role of the administrator who will be in charge of managing the centre. In general, this application will be used by the professionals who will be in charge of monitoring the people who use the mobile application. In addition, the specialists will be the ones who create the questions and manage the questionnaires that the users of the mobile application will carry out. Finally, an initial evaluation of the usability of the system has been carried out with several participants.

Yo, **Francisco de Asís Lobón Villanueva**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técniva Superior de Ingenierías Informática y Telecomunicación de la Universidad de Granada**, con DNI 77138051F, autorizo la ubicación de la siguiente copia de mi Trabajo de Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco de Asís Lobón Villanueva



Granada a 7 de septiembre de 2020.

D. **Oresti Baños Legrán**, Profesor del **Área de Ingeniería de Sistemas y Automática** del **Departamento de Arquitectura y Tecnología de Computadores** de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado **Emomobile**, ***Automatic detection of emotional behaviour from daily user-mobile interactions***, ha sido realizado bajo su supervisión por **Francisco de Asís Lobón Villanueva,** y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 7 de septiembre de 2020.

El director:

Digitally signed by
BAÑOS LEGRAN
ORESTI - 74681948M
Date: 2020.09.08
13:56:47 +02'00'

Oresti Baños Legrán

Agradecimientos

Agradecer a toda mi familia, a mi tutor y a mi pareja por todo el apoyo que me han dado a lo largo del TFG y también a las personas que han participado en la evaluación del sistema. También agradecer al departamento de **Arquitectura y Tecnología de Computadores** de la Universidad de Granada por haberme concedido la beca de colaboración.

Content

List of Figures

List of Tables

List of Codes

1. Introduction

1.1 Context

Bipolar disorder or Psychosis manic-depressive, It is a mood disorder that is characterised for marked changes in mood, thought, behaviour, energy, and the ability to do diary life activities.

It is not a passing mood or a state where you can go from one emotion to another in a short space of time, as many believe. This disorder affects the individual for months or years in stages, where the person suffering from it changes his or her normal state between manic or hypomanic episodes and depression.

Within bipolar disorder, there are several types that the DSM-V [1] classification lists: bipolar 1, bipolar 2, cyclothymic, bipolar or substance/medication-induced disorder, bipolar or substance-related disorder due to another medical condition, and bipolar or substance-related disorder, specified or unspecified.

To diagnose bipolar mood is necessary to know three episodes:

1. **Manic episode:**
   In which the person suffers a distinct period of activity or energy directed toward an object (or person), expansive or irritable abnormally and persistently, lasting at least one week and occurring most of the day, almost every day.

   The mood disturbance may be severe enough to cause a marked deterioration in social or occupational functioning or to require hospitalisation to prevent harm to oneself or others.

2. **Hypomanic episode:**
   A distinct period of abnormally high, expansive, or irritable activity or energy and persistently increased activity or energy, lasting at least four consecutive days and occurring most of the day, almost every day.

   Unlike the manic episode, it is not severe enough to cause a marked deterioration in social or occupational functioning or to require hospitalisation. In the case of psychotic features, the episode is manic.

   The hypomanic episode is more challenging to detect than the manic episode because it’s milder, usually detected through observation by others who know the person well.

3. **Major depressive episode:**
   Period in which the person feels depressed or manifests a loss of interest or pleasure.

The symptoms cause significant discomfort or impairment in social, occupational, or other important areas of functioning.

In Table 1**,** a characterisation table of the three episodes is shown along with the characteristics of all the types of bipolar disorder listed above.

Episodes		
Maniac	**Symptom:**	At least three should occur and represent a noticeable change in behaviour.
		- Inflated self-esteem or grandiosity.
		- Decreased need for sleep (e.g., feels rested after only 3 hours of sleep)
		- More talkative than usual or anxious to talk.
		- Brainstorming or subjective experiences of competing thoughts.
		- Distracted (shows attention to unimportant or irrelevant external stimuli)
		- Increased activity toward an object or psychomotor agitation (activity without purpose - not toward an object)
		- Excessive participation in activities that have a high potential for painful consequences (participating in free shopping, sexual indiscretions, or foolish business investments).
	Duration:	At least a week, he shows up most of the day, almost every day.
	Risk:	Severe, may require hospitalisation.
Hypomanic	**Symptom:**	The same as the maniac with the difference counted above that he is milder and does not present psychotic features.
	Duration:	At least four days in a row, it occurs most of the day, almost every day.
	Risk:	It is not severe enough, but if there are psychotic features, then it is a manic episode.
Major depressive	**Symptom:**	At least five must be given and be during the same two weeks period and represent changes in a previous performance. At least one of the symptoms is a depressed mood or, loss of interest or pleasure.
		- Depressed mood most of the day, almost every day, as indicated by a subjective report (they feel sad, empty, hopeless, ...) In children and adolescents, they may have an irritable mood.
		- Noticeably diminished interest or pleasure in all, or almost all, activities most of the day, nearly every day (as indicated by subjective observation).
		- Significant weight loss without dieting or weight gain, or decreased or increased appetite nearly every day. In children, failure to achieve expected weight gain is considered.
		- Insomnia or hyper insomnia almost every day.
		- Psychomotor agitation or delay almost every day (observable by others; not merely subjective feelings of restlessness or slowing down).
		- Fatigue or loss of energy almost every day.

		- Feelings of worthlessness or excessive or inappropriate guilt (which may be delusional) nearly every day (not just self-reproach or guilt about being sick).
		- Decreased ability to think or concentrate, or indecision, almost every day (always with observation of others and subjective).
		- Recurrent thoughts of death (not just fear or agony), recurrent suicidal ideation without a specific plan, or a suicide attempt or specific plan to commit suicide.
	Duration:	At least two weeks
	Risk:	Severe

Table 1.1: Bipolar disorder episode characterisation, according to DSM-V [1].

1.2 Motivation

Bipolar disorder is an illness that is present from the time it first occurs in adolescence or early adulthood and throughout a person's lifetime. It affects at least 0.6%-7% of the population (depending on the type of bipolar disorder), and it is estimated that those who have suffered the major depressive episode have a 12%-30% risk of suicide [1].

The way it is usually treated through drugs, which balance the mood of the person, and through psychotherapy with an expert, although as this illness is random, more meetings are usually held. In contrast, the person has some episode, if not the meetings are held every few months.

In addition, people who have mood disorders need a person who can take care of them, and they can not live alone as they may have a more significant lack of control of their state. It should also be noted that those who care for people with this disorder are usually their family members who also suffer psychological and emotional deterioration when dealing with people with this disorder. Therefore, the difficulty that this disorder represents is that the person who suffers it does not have a follow-up or control of his state continuously in time. Either subjectively, through questionnaires, or objectively, through monitoring the person's activity (mobile phone use activity, calls, physical activity, GPS tracking, etc). It implies that when the person begins to have the symptoms of an episode, some time goes until the monitoring of his condition begins. It can mean a worsening of the person's condition since each day that passes can worsen if the corresponding treatment is not taken, whether it be regulating the doses of the drugs, adding new ones or starting psychotherapy.

1.3 Work approach

Since the problem is not having a continuous monitoring of the bipolar symptoms, the question we would ask ourselves would be: could we monitor the person to know his or her status either objectively or subjectively?

The answer is in mobile devices. Most people have mobile phones, and of those people who have a mobile phone, it is a smartphone.

Therefore, in response to the question, we can monitor people with this disorder using the technologies we usually use, such as mobile phones, so that the expert in charge of diagnosing or monitoring them not only has the status of the person when they come to the clinic, but also, with the data collected by the mobile phone, we can generate a real-time history that shows us in more detail how the person with the disorder is doing daily.

In conclusion, a system can be created that objectively or subjectively collects information from the person with the disorder and that the expert can handle that information to follow the person continuously. With this, we manage to make it easier for the experts to monitoring on people with bipolar disorder.

1.4 Aims

The main objective is to develop a system for monitoring persons using mobile technologies to help both the person and the doctors by having a history and a better control of their condition.

In order to achieve this goal, the following subobjectives are identified:

- Analyse publications on the use of mobile technologies in persons with bipolar disorder objectively and subjectively.
- Design an architecture oriented to the automatic administration and collection of remote digital questionnaires.
- Implement the prototype according to the design requirements.
- Evaluate the system through a usability study with a group of people.

1.5 Project structure

Chapter 1:

The current chapter explains why this project emerges, the context of the illness, what the problem is, a possible resolution, and what the objectives are.

Chapter 2:

In this chapter, we will present several investigations related to the two ways we have to follow a person, subjectively through questionnaires or objectively through monitoring the person's activity.

Chapter 3:

In this chapter, we will focus on the design of the system through the requirements and the architecture that will be used.

Chapter 4:

In this chapter, we will explain which resources have been used in the implementation of the system, and we will tell through snippets the most outstanding parts of the system's functionalities.

Chapter 5:

In this chapter, a small study of the system usability will be carried out.

Chapter 6:

We will conclude this chapter by the objectives and checking that we have fulfilled them, giving an opinion about the work done and about the future work.

Chapter 7:

Finally, this chapter will focus on thanking the people who have been involved in the process of doing this project.

1.6 Planning and budget

Planning

Activity	**Time**
Introduction	**60**
State of Art	**60**
Design	**8**
Objective	2
Requirements	1
Architecture	5
Implementación	**92**
Android Application	35
Web Application	57
Assessments	**10**
Conclusions	**10**
Total	**240**

Table 1.2: Expected time.

Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Introduction																	
State of Art																	
Design																	
Implementation																	
Assessment																	
Conclusion																	

Table 1.3: Weekly Gantt diagram, expected time.

Activity	**Time**
Introduction	**60**
State of Art	**60**
Design	8
Objective	2
Requirements	1
Architecture	5
Implementación	**720**
Android Application	320
Web Application	400
Assessments	**10**
Conclusions	**10**
Total	**868**

Table 1.4: Real-time.

Week	1	2	3	4	5	6	7	8	+38				46	47	48	49	50
Introduction																	
State of Art																	
Design																	
Implementation																	
Assessment																	
Conclusion																	

Table 1.5: Weekly Gantt diagram, real-time.

In this project, it was expected to have been able to finish the implementation in less time, but I had to learn more about Android, so it took me more time to finish the application. Also the project had a change in the implementation due to the collaboration grant obtained from the Department of Computer Architecture and Technology (ATC). Due to this, the web application was extended scaling to clinical centres. At that time, the web application and the Android application was not implemented, so it could be scaled. Apart from this, there were several investigations about wearables that made the implementation process take even longer. This research was completed successfully but was not implemented in the final application due to time issues.

Budget

Based on the salary tables of the collective agreement of the commerce sector of Granada [24], a graduate in computer science is considered to be in group 2 and therefore should earn around 20,440 € per year which is approximately 1700 € per month. To know what you would have to charge per working hour 1700 € / 5 weeks * 40 hours/week = 8.5 € per hour. So the project would be around 868*8.5 = 7.378 € to be profitable. From the above-mentioned salary, IRPF, SS, and web server and domain costs would have to be deducted.

2 State of the art

This section aims to find out about the work done based on the studies already carried out, by grouping them into two groups that coincide in the two ways of monitoring a person using mobile technologies, both objectively and subjectively.

2.1 Objective

Automatic tracking of self-reported behaviour is spreading rapidly, thanks to the emergence of new detection technologies. This has opened up new ways of capturing a person's behaviour. In the case of bipolar disorder, there are more studies with various applications, including the use of multiple wearables.

The Table 2.1 at the end of this section shows a summary of the data reported automatically (not from the questionnaires), as well as a brief description of the system of each following items.

In the study [8], they developed the Lorevimo mobile application in which mood is recorded, reviewed and visualised. It uses a smartwatch that collects information on physical activity, sleep and heart rate, and encourages the user twice a day (once in the morning and once at night) to participate in questions about manic and depressive symptoms and medication adherence.

The authors of [9], they designed and implemented the mobile application MoodRhythm which tracks social rhythms (SRM quantifies the social rhythms that result from a patient's interaction with a social environment). This application collects data from smartphone sensors and also implements a time selection system for wakefulness, first contact with another person, start of the day, dinner and bedtime, as well as allows for the addition of personalised activities and the tracking of mood and energy through a scale. The purpose of the application is to allow patients to set daily target times for activities and to track how closely they are meeting these target times. The notes can be used to record additional information, such as the amount of medication taken or factors that may have affected a patient's routine or mood. The application is designed to provide a summary of a person's success in meeting their pacing goals for both the current and previous day.

In this study [10] they created the SIMBA mobile application that collects both self-reported and sensor data. They used various open-source tools to collect this data. Self-reported moods were assessed once a day by means of a questionnaire with two elements.

In the subsequent study [11], PSYCHE was developed. A portable T-shirt based system with integrated fabric electrodes and sensors capable of acquiring information on the electrocardiogram, breathing and body posture to detect a pattern of objective physiological parameters to support the diagnosis. A mobile phone was also used to collect the data and send it to a server where the professional could view the information. Besides, they implemented a new ad hoc methodology of advanced biological signal processing capable

of effectively recognising four possible clinical moods in the participants (depression, mixed state, hypomania and euthymia).

Finally, the study [12] and its subsequent version [13], the Monarca mobile application was developed, which records both objective and subjective data from the participants. On the one hand, they collect data from the mobile phone sensors, and on the other hand, they use a questionnaire that the participants have to fill in every night. This questionnaire is composed by the qualification of the mood, sleep duration, medication intake, if irritated or not, activity level using a scale, mixed mood, cognitive problems, alcohol consumption, stress and warning signs.

Name	**System**	**Auto-reporting data**
Lorevimo	Mobile application based on data collection and analysis from a Smartwatch.	● Dream ● Activity ● Heart rate
MoodRhythm	Mobile application based on data collection and questionnaires, and their subsequent analysis.	● Accelerometer ● Microphone ● GPS ● Voice ● Activity ● SMS ● Call log
SIMBA	Mobile application based on data collection and questionnaires, and their subsequent analysis.	● GPS ● Accelerometer ● Screen Status
PSYCHE	Portable system based on a T-shirt with electrodes from which data is obtained in a mobile application and its subsequent analysis.	● Electrocardiogram ● Breathing ● Ambient light ● Temperature ● Noise ● Activity
MONARCA	Mobile application based on data collection and questionnaires, and their subsequent analysis.	● Accelerometer ● Call log ● Screen ● Running applications ● Activity

Table 2.1: Summary of the systems along with auto-reporting data.

2.2 Subjective

Several studies use various questionnaires to assess each of the states as well as questionnaires that determine the functional impairment associated with bipolar disorder.

To quantify mania (ASRM [2]), depression (QIDS [3]), anxiety (GAD-7 [4]), and quality of life (EQ-5D [5]). These are some studies on the standard questionnaires related to this disorder on which some of the applications cited below have been based. There are also studies on sleep and affect status.

The authors of [6], created a web-based system. Capable of programming and initiating interactive content independently for each participant. Depending on the scheduled date, participants were invited to answer a questionnaire at a randomly scheduled time within a block of 3-4 hours in the morning and evening. This was achieved through the use of timed SMS generation that opened the device's browser to access the questionnaires. Participants could choose the times at which they wanted to receive the message. The questionnaires were carried out from the website that was opened from the message. They also had a separate program that provided on-demand graphical feedback on any of the mood elements that could be extended from last week to last month. The questionnaires were personalised.

It can be said that the previous study makes use of the mobile phone even if it is not through an application and that ultimately it can be used by any type of smartphone that uses an Internet connection.

In this study [7] they designed and implemented a mobile application that made use of a standard questionnaire on the state of mania. This questionnaire was to be conducted every day in the evening at a pre-set time convenient for each participant by means of a notification.

3. Design

Based on the study I have done in the previous points, I will focus on developing a system to monitor users through questions.

This system consists of two applications. The first application will be for smartphones that will allow users to register, identify themselves, access their data and see if they have forms or not for the current day. Also, this system will be in charge of the synchronisation with the second system through the internet and will warn the user of the forms available.

The second application will be a web page. For greater flexibility, this website will be generic and will support multiple health centres. Each health centre will be composed of users of the centre who will have different roles, administrator, specialist and secretary. Each of these roles will have some actions that can be performed.

Besides, the questions will be customisable, there will be both private questions for each specialist and global questions for the centre, and they will be able to be associated with a previously created category.

For each user of the mobile application, there will be a table with the days of the week in which the questions can be assigned, and it will be periodic. This way of asking questions is not arbitrary, but is made on purpose based on the questionnaires that are used to measure different states. ASRM can be done every day. QIDS is done every seven days. GAD7 is done every two weeks (although it could also be done every week). EQ5D is a questionnaire that has several questions that can be asked with different frequencies, for example, your health status today on a scale 1 to 100 can be asked every day, and so you could know how good or bad that person is. The table will also be interactive, and various actions can be carried out on it.

For both the questions and the question assignment table, a history will be created with the modifications that are made.

Although there are different users, we can group them in two groups, on the one hand, we have the users of the mobile application and on the other hand the users of the centre that includes the administrator, specialist and secretary.

In the Figure 3.1 below, we can see the two systems that compose the application:

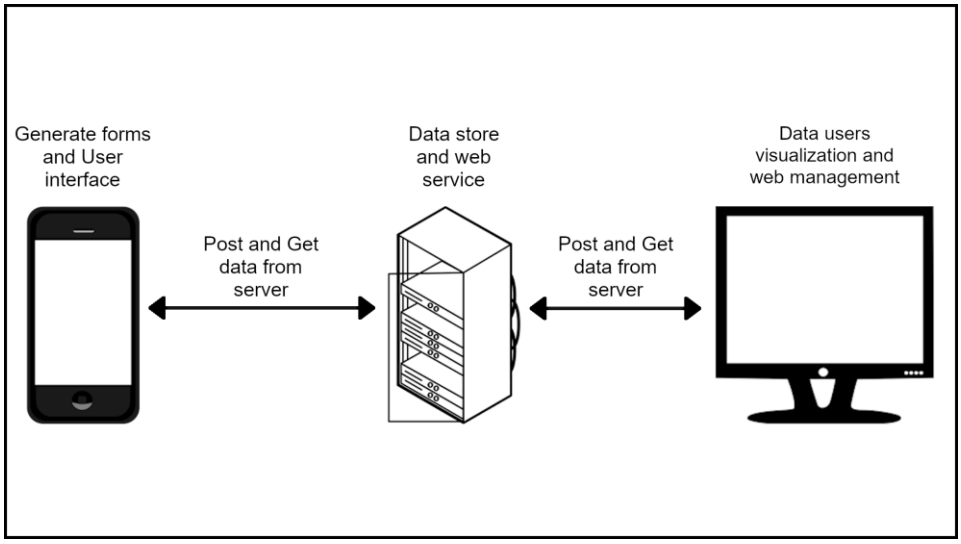


Figure 3.1: Top-level view of the system.

3.1 Requirements

The actors acting in the system are the two groups of users mentioned above, mobile application users and centre users.

Functional requirements

The list of functional requirements I'm going to separate into two parts. On one hand the system requirements of the mobile application and on the other hand the requirements of the web application:

Requirements for the Mobile Application:

- **FR-1**: Identify and register users.
- **FR-2**: Display general and personal data of the user.
- **FR-3**: Modify personal data.
- **FR-4**: Recover password in case of forgetting the user.
- **FR-5**: Notify the user of the forms they have on the current day.
- **FR-6**: Provide the user with an interface to view and complete the forms.
- **FR-7**: Send data to the server.
- **FR-8**: Receive data from the webserver.
- **FR-9**: Save personal data in the device.

Requirements for the Web Application:

- **FR-9**: Identify the users of the web system.
- **FR-10**: Register new users, both users of the mobile application and users of the centre.
- **FR-11**: Modify data of the users of the centre.
- **FR-12**: Cancel or register the users of the centre.
- **FR-13**: Delete users.

- **FR-14**: Confirm registered users from the mobile application.
- **FR-15**: Assign or release users from the mobile application to a user in the centre.
- **FR-16**: Activate or deactivate users.
- **FR-17**: Create, modify or delete question categories.
- **FR-18**: Create, modify or delete questions.
- **FR-19**: Display a question history.
- **FR-20**: Assign questions to users of the mobile application using a table
    - **FR-20.1**: Add, delete and change questions in the table.
    - **FR-20.2**: Save table.
- **FR-21**: Display a history of related/assigned questions from/to users of the mobile application.
- **FR-22**: Receive data from the mobile application.
- **FR-23**: Display question answers.
- **FR-24**: Notify new registered users.
- **FR-25**: Save data of users, questions, categories and answers in a database.

Non-functional requirements

As with the functional requirements, I will separate them into two groups:

Requirements for the Mobile Application:

- **NFR-1**: The system should detect if a reboot occurs in the device and reboot.
- **NFR -2**: The system must provide warning messages if there is an error in the server.

Requirements for the Web Application:

- **NFR -3**: The web application must have a "Responsive" design to ensure proper display on multiple personal computers, tablets and smartphones.
- **NFR -4**: The web application must guarantee that SQL injections cannot be performed.
- **NFR -5**: Only technical support can service a new centre by creating the centre, the centre administrator user, and the first category.
    - **NFR -5.1**: The first category of the centre may not be modified or removed.
- **NFR -6**: Categories cannot be deleted if you have associated questions, but you can hide them from the site by making the associated questions not displayed.
- **NFR -7**: Questions cannot be deleted from the database, they will only appear as deleted questions, but they can be removed from deleted questions to be used again.
- **NFR -8**: The administrator can manage all users.
- **NFR -9**: The specialist can only manage his users.
- **NFR -10**: A Specialist can only view global questions as well as their private questions.
- **NFR -11**: If a Specialist creates a global question, he/she cannot modify it. He will have to contact the administrator to do so.

In the following table shows the functions that will be performed in more detail by each of the users:

<table>
<tr><td>Mobile Application Users</td><td colspan="2">Actions</td></tr>
<tr><td></td><td colspan="2">- Register.
- Identify.
- Recover password.
- Change personal data except for the email, if the user want to change it you must contact the technical service.
- Answer questionnaires.</td></tr>
<tr><td>Centre Users</td><td>Rol</td><td>Actions</td></tr>
<tr><td></td><td>Administrator</td><td>See NFR-8.

- Application user management:
● Create
● Delete
● Confirm
● Assigning a specialist
● Release
● See information (name, surname, etc)
● Activate/Deactivate
● Assigning questions
● Consult history of related/assigned questions
● Display answers

- Question management:
● Create
● Delete
● Modify
● View question history
● View all global and private questions for each specialist

- Category management:
● Create
● Delete
● Modify
● Hide

- Centre management:
● Create plant users
● Delete users from the centre
● Change users of the plant
● Terminate/discharge users from the centre</td></tr>
<tr><td></td><td>Specialist</td><td>See NFR-9.

- Application user management:
● Create</td></tr>
</table>

		● Release ● See information only about your users (name, surname, etc). ● Activate/Deactivate ● Assigning questions ● Consult history of related/assigned questions ● Display answers **See NFR-10.** - Question management: ● Create ● Delete, only your private questions ● Modify, only your private questions ● View question history ● View all global and private questions for each specialist
	Secretary	- Application user management: ● Create ● Confirm ● Assigning a specialist ● See information (name, surname, etc)

Table 3.1: Functions that can be performed by each user.

Breakdown of terms

Delete users (RF-13): refers to both application and site users.

Confirm users (RF-14): when a user is registered, from the application, it must be confirmed by the administrator or secretary of the centre, this is done to separate the users (active, not active and pending confirmation).

Release users (RF-15): this means that a user can be released so that it can be assigned to another specialist.

3.2 Architecture

Figure 3.3 shows in great detail the interaction of each actor with the system as well as the interactions of each component.

The architecture consists of a system of two modules. These modules are the two applications that the system has.

On the one hand, we have the web application module. This module is divided into two sub-modules, the Front-End and the Back-End. The Front-End is composed of the Dashboard component that is responsible for providing the data of the Back-End sub-module to the actors. Depending on the actor that interacts with this module, it will send and receive different data. The Administrator will send and receive Users Data, Questions Data, Categories Data and Centre Data. The Specialist will send and receive Users Data and Questions Data. The Secretary will send and receive only Users Data. The Back-End is composed by the following components: User Manager manages the user data, Question Manager manages the questions, Category Manager manages the categories, Centre Manager manages the centre, and finally the DataBase saves all the system data. The data flow from the Administrator to the database would be as follows: Administrator sends User Data to the Dashboard, this, in turn, sends it to the User Manager and DataBase receives and saves the User Data from the User Manager. An example is shown below Figure 3.2.

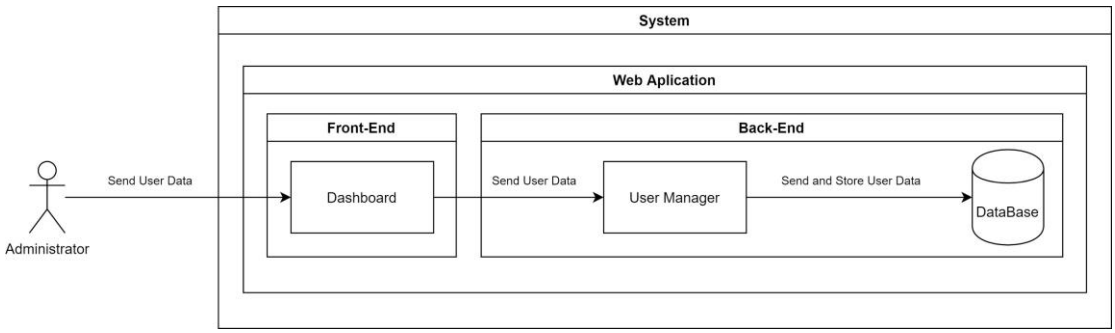


Figure 3.2: Example data flow.

On the other hand, we have the mobile application module. This module is composed by the following components: User Manager is in charge of managing the user data, receives and provides User Data from the User, and sends and obtains User Data from the DataBase component; Notification Manager is in charge of providing Questions Data to the User and obtaining Questions Data from the DataBase; Answer Manager receives Answers Data from the User and sends Answer Data to the DataBase; DataBase keeps the user information, the questions and the answers; DataBase Synchronization Manager is in charge of synchronising the DataBases of the two applications.

3. Design

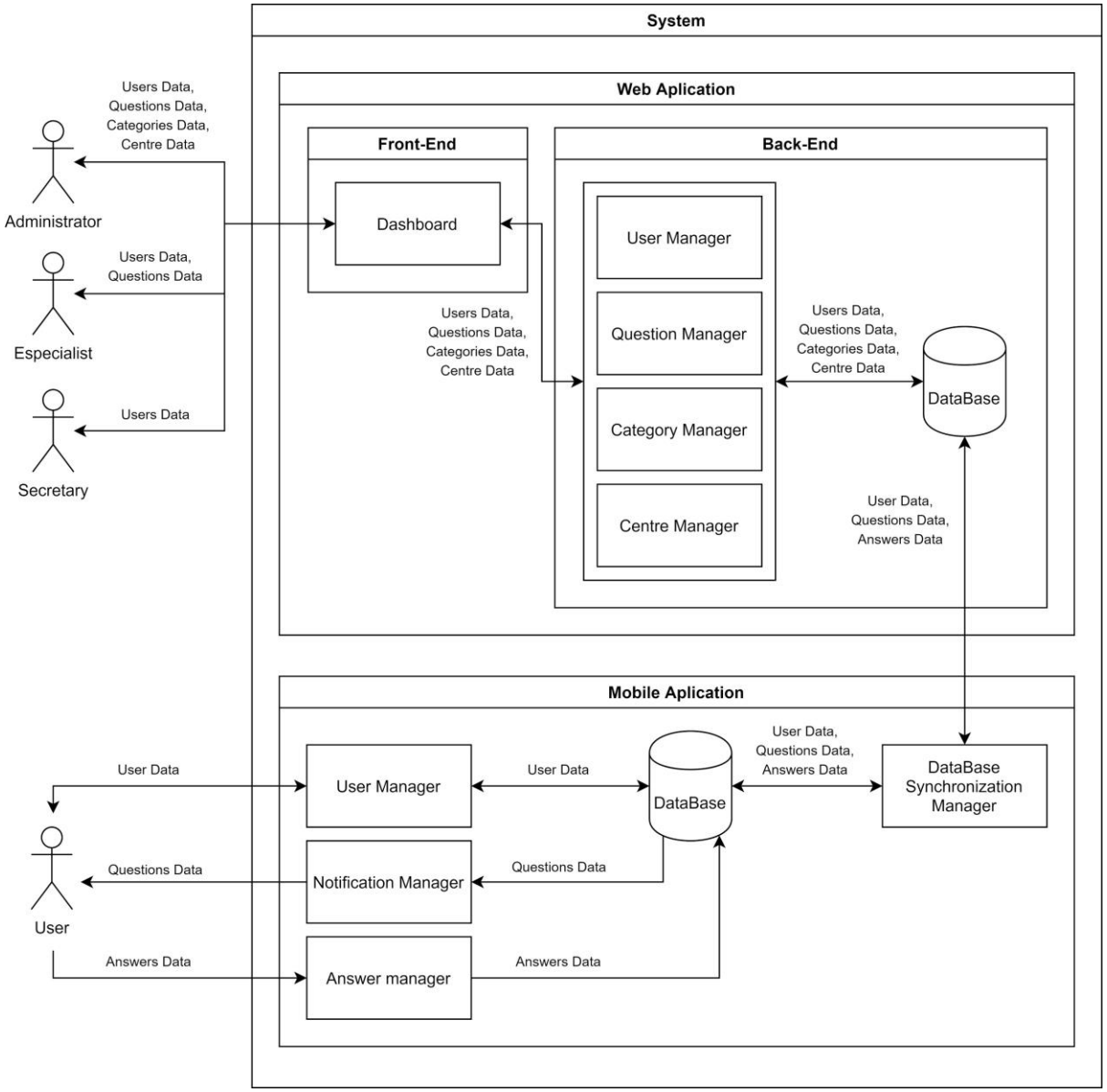


Figure 3.3: System Flowchart.

4 Implementation

4.1 Overview

As seen in previous chapters, the system will be composed of two applications. On the one hand, the mobile application which is the tool for monitoring people, and on the other hand, the web application that will be used by the medical centre where the experts will have a view of these results.

In the following sections of this chapter, the explanation of each of the two systems is shown as well as the implementation of fundamental parts.

4.2 Mobile Application

The mobile application has been designed for Android as it is the most used operating system in mobile devices at least in Spain with 39% compared to IOS with 10% on average, besides being the most affordable phones for anyone. And it is the device that has been counted on to make the application.

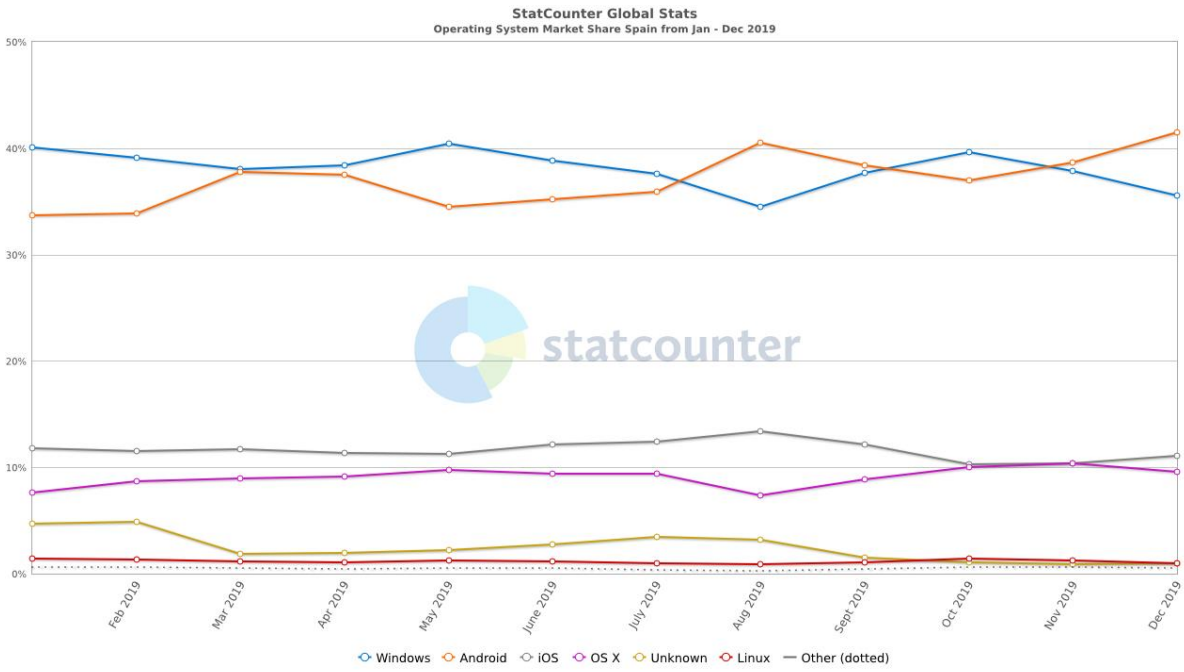


Figure 4.1: Use of different operating systems in Spain [14].

The main characteristics of the mobile application that will be discussed in the following subsections will be:

- Permissions, dependencies and Android version of the mobile application.
- How data requests are made to the server.
- Storage methods used in the mobile application.

- Background services.
- Application startup method when rebooting the device.
- Viewing and using the different screens of the application.

4.2.1 Permissions, dependencies and Android version

The following permissions are required to make the application work:

```
<uses-permission
      android:name="android.permission.INTERNET"
/>
<uses-permission
      android:name="android.permission.RECEIVE_BOOT_COMPLETED"
/>
```

Code 4.1: Permissions.

INTERNET: to be able to send and receive information from the server
RECEIVE_BOOT_COMPLETED: to start the application if the smartphone is restarted. It will be discussed in more detail in a later section.

As for the dependencies, the following ones have been used:

```
// ROOM DATA BASE
def room_version = "2.2.0"
implementation "androidx.room:room-runtime:$room_version"
annotationProcessor "androidx.room:room-compiler:$room_version"

// Volley
implementation 'com.android.volley:volley:1.1.1'
```

Code 4.2: Dependencies.

There are more dependencies, but these two are the most important, the others are for the creation of layouts. The use of these two dependencies is explained in more detail in the following subsections.

Finally, this application is made to work in mobiles with API version 21 (Android 5.0) up to API 29 (Android 10.0).

4.2.2 Request data from server

To send and receive data from the server, we have used the Volley library that is included in the previous section as a dependency of the project.

This library serves to facilitate the use of *HTTP* requests, and also allows making both asynchronous and synchronous requests.

In this case for sending and receiving data via *HTTP* synchronous requests have been used since the response from the server is needed to continue with the normal flow of the application.

The following code shows what asynchronous *HTTP* request looks like:

```
public int loginUserFromServer(Context mContext, final List<String> postParams) {
    // LOGIN_URL: Website URL (https://www.behaviourtracker.es/login.php)
    final String LOGIN_URL = ServerConstants.serverUrl + "login.php";
    int code = 3;
    RequestQueue mRequestQueue;
    mRequestQueue = Volley.newRequestQueue(mContext);
    RequestFuture<String> future = RequestFuture.newFuture();

    // POST PARAM to the website
    StringRequest request = new StringRequest(LOGIN_URL, future, future) {
        @Override
        public Map<String, String> getParams() {
            Map<String, String> params = new HashMap<String, String>();
            params.put("email", postParams.get(0));
            params.put("password", postParams.get(1));
            return params;
        }

        @Override
        public int getMethod() {
            return Method.POST;
        }
    };
    mRequestQueue.add(request);

    try {
        String response = null;

    // TIMEOUT after 5 second waiting
        try {
            response = future.get(5000, TimeUnit.MILLISECONDS);

        } catch (InterruptedException e) {
            // Received interrupt signal, but still don't have response
            // Restore thread's interrupted status to use higher up on the call
stack
            Thread.currentThread().interrupt();
        }

        try {

            if(response != null){
                JSONObject parseJSON = new JSONObject(response);
                code = parseJSON.getInt("code");
                if (code == 0){
                    mUserDB = new UserDB(
                            parseJSON.getString("uuid"),
                            parseJSON.getString("admin_user"),
                            parseJSON.getString("health_center"),
                            parseJSON.getString("active_user"),
                            postParams.get(0), // email
                            parseJSON.getString("name"),
                            parseJSON.getString("surname"),
                            postParams.get(1), // password
                            parseJSON.getString("date_of_birth"),
                            parseJSON.getString("gender"),
                            parseJSON.getString("registration_date"),
                            parseJSON.getString("update_date")
                    );
```

```
                    code = 10;
                }
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    } catch (ExecutionException e) {
        e.printStackTrace();
    } catch (TimeoutException e) {
        e.printStackTrace();
    }

    return code;
}
```

Code 4.3: Request Login User.

This code shows asynchronous *HTTP POST* request used to identify the application, being synchronous we have a five seconds *TIMEOUT*. The reception of all data requests from the server is done through *JSON* format to make it easier to obtain the data received from the server and vice versa.

```
{
        'code' : '0',
        'uuid' : 'uuid',
        'health_center' : 'health_center',
        'admin_user' : 'admin_user',
        'active_user' : 'active_user',
        'name' : 'name',
        'surname' : 'surname'',
        'date_of_birth' : 'date_of_birth',
        'gender' : 'gender',
        'registration_date' : 'registration_date',
        'update_date' : 'update_date'
}
```

Figure 4.2: Data representation in JSON.

All *HTTP* request functions return an error code (`int code`). The next table shows what each code means.

Code	Explanation
0	Ok. No error.
1	Error. No User.
2	Error. Incorrect password.
3	Error. Server error or internet error.
10	Ok. No error. New User.

Table 4.1: Error code HTTP requests.

4.2.3 Data Storage

Two systems are used for data storage: one is the database for data storage, and the other is private preferences.

DataBase

In the case of the database, the API ROOM has been used, which is recommended when creating databases and provides an abstraction layer of *SQLite* that allows easier access to the database.

In this case, the database consists of three components:

- DataBase: Provides the name of the database and is the point from which the database is accessed by normal functions.
- Entity: These are classes that will represent the database tables.
- Dao: Contains the queries of each one of the entities.

The app uses the ROOM database that obtains the objects to access the data (DAO) associated with that database. Then, the app uses each DAO to obtain entities from the database and save the changes made in those entities in the database. Finally, the app uses an entity to get and set values that correspond to table columns within the database [15].

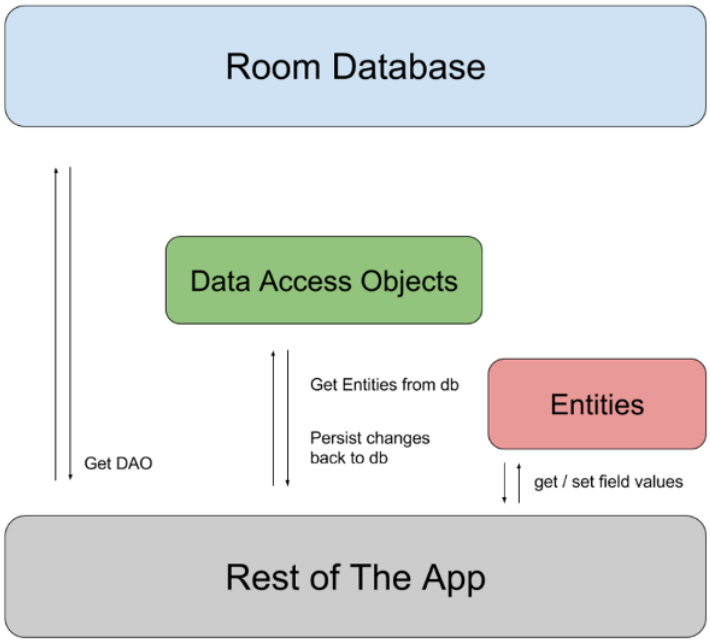


Figure 4.3: Room architecture diagram [15].

The following tables have been used for this application:

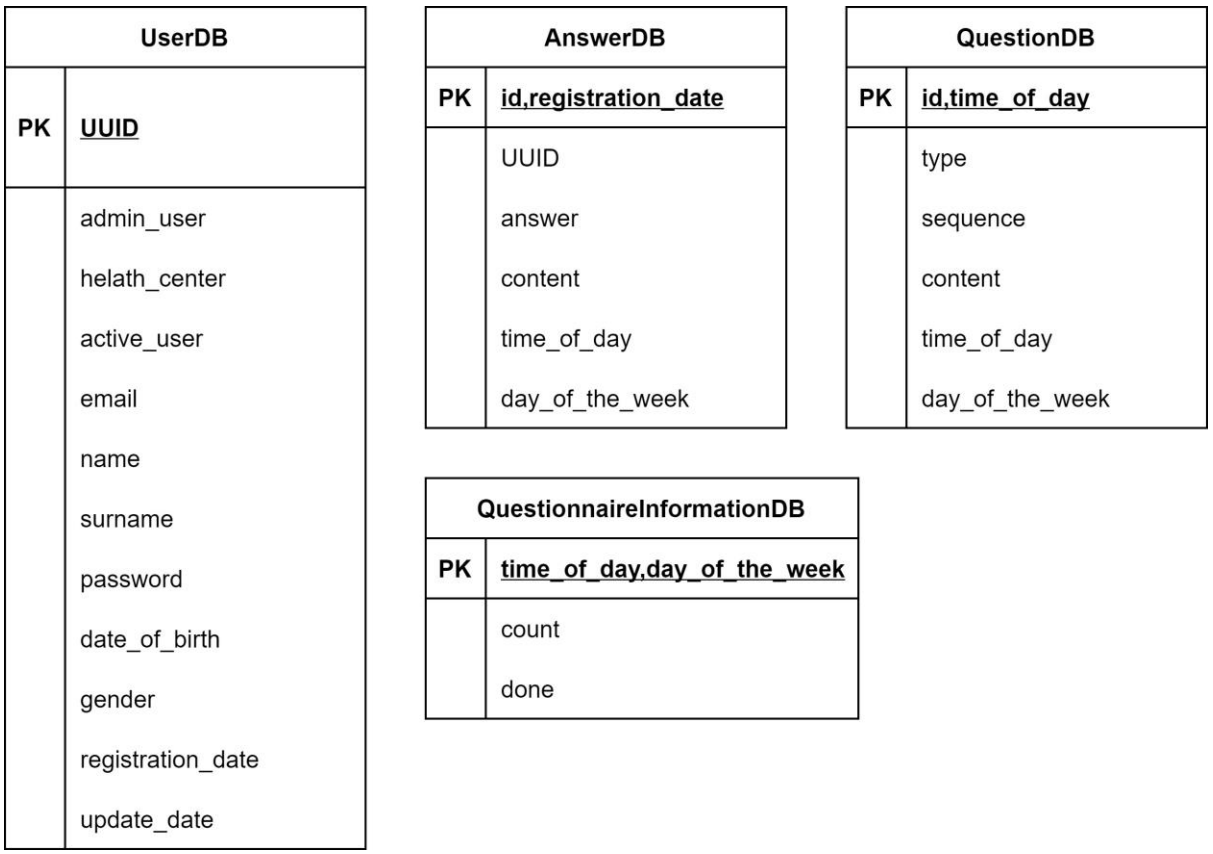


Figure 4.4: App DataBase.

As shown in the figure above, all user data, as well as answers and questions, are stored. In addition, *QuestionnaireInformationDB* is used for the user interface and serves to find out which questionnaires the user has taken and how many questions each questionnaire has.

The following code shows an example of the *QuestionnaireInformationDB* database.

Entity

```
@Entity(tableName = "QuestionnaireInformation",primaryKeys =
{"time_of_day","day_of_the_week"})
public class QuestionnaireInformationDB{
   @NonNull
   @ColumnInfo(name = "time_of_day")
   private int timeOfDay;
   @NonNull
   @ColumnInfo(name = "day_of_the_week")
   private int dayOfTheWeek;
   @NonNull
   @ColumnInfo(name = "count")
   private int count;
   @NonNull
   @ColumnInfo(name = "done")
   private boolean done;

   public QuestionnaireInformationDB(){}

   @Ignore
   public QuestionnaireInformationDB(int time_of_day, int day_of_the_week, int
count, boolean done) {
```

```
        this.time_of_day = time_of_day;
        this.day_of_thew_eek = day_of_the_week;
        this.count = count;
        this.done = done;
    }

    public int getTimeOfDay() {
        return time_of_day;
    }

    public void setTimeOfDay(int time_of_day) {
        this.time_of_day = time_of_day;
    }

                .
                .
                .
}
```

DAO

```
@Dao
public interface QuestionnaireInformationDBDao {

    @Query("SELECT * FROM QuestionnaireInformation ORDER BY time_of_day ASC")
    List<QuestionnaireInformationDB> loadAllQuestionnaireInformation();

    @Query("SELECT * FROM QuestionnaireInformation WHERE time_of_day =
:time_of_day and day_of_the_week = :day_of_the_week")
    QuestionnaireInformationDB
loadQuestionnaireInformationByTimeOfDayAndDayOfTheWeek(int time_of_day,int
day_of_the_week);

    @Query("DELETE FROM QuestionnaireInformation ")
    void deleteAllQuestionnaireInformation();

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insert(QuestionnaireInformationDB  questionnaireInformationDB );

    @Update
    void update(QuestionnaireInformationDB  questionnaireInformationDB );
}
```

DataBase

```
public class BehaviourTrackerDB {
    private static volatile BehaviourTrackerDB sDB = null;

                    ...

    private QuestionnaireInformationDB mQuestionnaireInformationDBDao ;

    private BehaviourTrackerDB(Context context){
        if (sDB != null){
            throw new RuntimeException("Use getInstance() method to get the single
instance of this class.");
        }else {
            Context appContext = context.getApplicationContext();
            AppDatabase database = Room.databaseBuilder(appContext,
AppDatabase.class, "BehaviourTracker")
                    .build();

                    ...

            mQuestionnaireInformationDBDao = database.getFormInformationDBDao();
        }
```

```
    }

    public static BehaviourTrackerDB get(Context context) {
        if (sDB == null) {
            synchronized (BehaviourTrackerDB.class) {
                if (sDB == null) {
                    sDB = new BehaviourTrackerDB(context);
                }
            }
        }
        return sDB;
    }


    /*
       ******************
        DATABASE QUERYS
       ******************
    */

    /* USER */
                    ...

    /* QUESTION */

                    ...

    /* ANSWER */

                    ...

    /* QUESTIONNAIREINFORMATION */

    public List<QuestionnaireInformationDB> loadAllFormInformation(){return
mQuestionnaireInformationDBDao.loadAllFormInformation();}

    public void
deleteAllFormInformation(){mQuestionnaireInformationDBDao.deleteAllQuestionnaireI
nformation();}

    public void insert(QuestionnaireInformationDB
questionnaireInformationDB){mQuestionnaireInformationDBDao.insert(questionnaireIn
formationDB);}
    public void update(QuestionnaireInformationDB
questionnaireInformationDB){mQuestionnaireInformationDBDao.insert(questionnaireIn
formationDB);}

                    ...

}
```

Code 4.4: DataBase example.

The following sections will show the interactions that the system makes with the database.

Shared Preferences

Shared Preferences [16] is an API that provides a key-value-style storage method. This storage method can be private or shared by many other applications. This way, it can save the user's configuration. In this application, whether the user is remembered or not and his *UUID* are saved using this API, and all this is private for the application.

Class *UserSettings*

```
public class UserSettings {
   private static volatile UserSettings sUserSettings = null;
   private final String NODATA = "NODATA";
   private final String PREFERENCES = "UserSettings";
   private Context context;


   private UserSettings (Context context){
       if (sUserSettings != null){
           throw new RuntimeException("Use getInstance() method to get the single
instance of this class.");
       }else {
           this.context = context;
       }
   }

   public static UserSettings getInstance(Context mContext){
       if (sUserSettings == null){
           synchronized (UserSettings.class){
               if (sUserSettings == null){
                   sUserSettings = new UserSettings(mContext);
               }
           }
       }
       return sUserSettings;
   }

   public void update (String type, String value){
       SharedPreferences sharedPreferences =
context.getSharedPreferences(PREFERENCES, Context.MODE_PRIVATE);
       SharedPreferences.Editor editor = sharedPreferences.edit();
       editor.putString(type,value);
       editor.apply();
   }

   public String get (String type){
       SharedPreferences sharedPreferences =
context.getSharedPreferences(PREFERENCES, Context.MODE_PRIVATE);
       String result = sharedPreferences.getString(type,NODATA);
       return result;
   }
}
```

Managed data

```
public enum DataType {
   REMEMBER,UUID
}
```

Code 4.5: Private preferences.

The code above shows the *UserSettings* class that serves to manage the data of the *DataType* listing with the update and get methods. *Update()* updates the current value of the *DataType* by the value given in value, and *Get()* returns the value that the *DataType* had and in case it did not have any data it would return *NODATA*.

To finish this section, we should say that both for the main class of the *BehaviourTrackerDB* database and the *UserSettings* class are *Singleton* classes to be able to use them in several processes at the same time. So there is only one instance of each one.

4.2.4 NotificationService and SynchronizationService

This application uses two background services that take care of tasks when the user is not using the application.

On the one hand, *NotificationService* is the service in charge of setting the alarms according to the time of each questionnaire of the current day. *NotificationService* generate the notifications when these alarms are activated and eliminating all the notifications and alarms in case the user logs out of the application.

On the other hand, *SynchronizationService* is in charge of sending the data from the database to the server and deleting the data it has sent.

The flow of a service in the background is shown in Figure 4.5.

The code used for this service is similar to the code that appears in google at [17]. This service works like a queue, that is, for each call to the service the task is queued until it has finished doing the first task it was doing, when it has finished it goes on to the next one and so on in a loop until it finishes all the tasks. This is useful to have a service that performs multiple tasks sequentially. In the case of this application, the same development has been used, but instead of doing the tasks in the same main service execution thread, it is done from a second one because the HTTP requests and the database requests cannot be made in the main thread (either from an Activity or from a Service in the background).

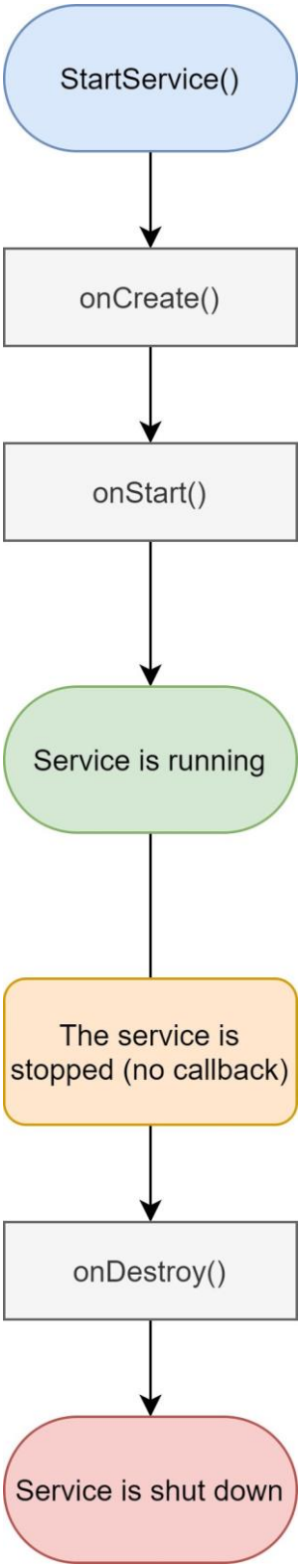


Figure 4.5: Service life cycle.

The main difference is that to do multiple tasks it uses a *TASK_ID* that serves to identify the task to be performed, in addition in the case of *NotificationService* is passed other parameters depending on what task will be performed.

Notification Service

```
    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper){
            super(looper);
        }

        @Override
        public void handleMessage(Message msg){
            int time = msg.getData().getInt("TIME", -1);
            int day = msg.getData().getInt("DAY", -1);
            final String UUID = msg.getData().getString("UUID","1");
            Intent intent;
            PendingIntent pendingIntent;

            switch (msg.arg2) {
                case 0:
                    break;
                case 1:
                    break;
                case 2:
                    break;
                default:
                    break;
            }
            stopSelf(msg.arg1);
        }
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        if(intent != null){
            intent.getIntExtra("TASK_ID",-1) + " UUID: " +
            intent.getExtras().getString("UUID") +  " UUID: " +
            intent.getStringExtra("UUID"));
            Message msg = mServiceHandler.obtainMessage();
            msg.arg1 = startId;
            msg.arg2 = intent.getIntExtra("TASK_ID", -1);
            Bundle bundle = new Bundle();
            bundle.putInt("TIME",intent.getIntExtra("TIME", -1));
            bundle.putInt("DAY",intent.getIntExtra("DAY", -1));

            String uuid = intent.getStringExtra("UUID");
            if(uuid != null) {
                bundle.putString("UUID", uuid);
            }

            msg.setData(bundle);

            //mServiceHandler.handleMessage(msg); In the same Thread

            mServiceHandler.sendMessage(msg);  // In a new Thread
        }
        return START_NOT_STICKY;
    }
}
```

Code 4.6: Example of how messages are handled in Services.

The previous code is a small extract of the *NotificationService* class to visualise how the data are collected in *onStartCommand()* from the person who has called the service and how they are used to send them to the queue via *sendMessage()* like a message.
The data depends on the task to be performed. The following table shows for each task the call that is made and its corresponding data that is sent.

Task	Data
0	- TASK_ID = 0 - UUID
1	- TASK_ID = 1 - TIME - DAY
2	- TASK_ID = 2

Table 4.2: Notification Service Task Data.

Once we know what data is sent and how that data is handled and queued, we will describe what is done in each of the service's tasks.

Task 0:

Task 0 is executed when the user logs into the application, when the device is rebooted and when it is 00:00 hours.

This task performs the following:

- Launch Synchronization Service to send the answers that are currently available.
- Set a silent notification. This notification does not cause the device to vibrate or ring, and has an associated action that takes you to the main screen of the application when you click.
- Check that there are no questions for the current day. If there are questions, the task is completed.
- If there are no questions, they are deleted (*QuestionDB* and *QuestionnaireInformationDB*).
- *HTTP* request to the server to acquire the new questions for that day in *JSON* format.
- If there are new questions, they are saved in the database. If there are no questions, the task is terminated.
- For each questionnaire of that day, its corresponding information is generated in *QuestionnaireInformationDB,* and an alarm is generated for the indicated time. Each alarm has an associated event, and it is that when it is activated, it will do the task 1 of this same service.

Figure 4.6 shows a flow chart of what Task 0 does and Figure 4.7 shows what the notification looks like on the mobile device.

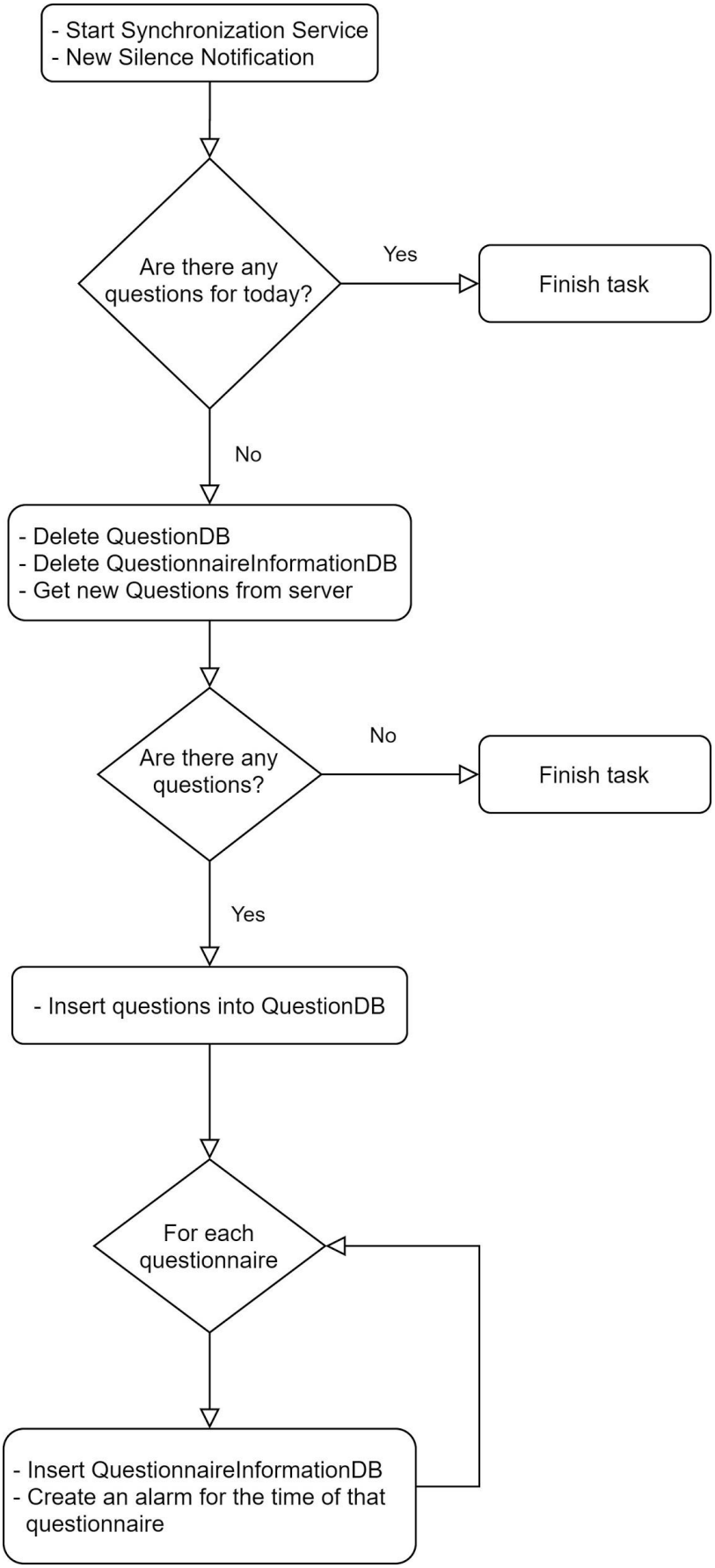


Figure 4.6: Notification Service Task 1 Flowchart.

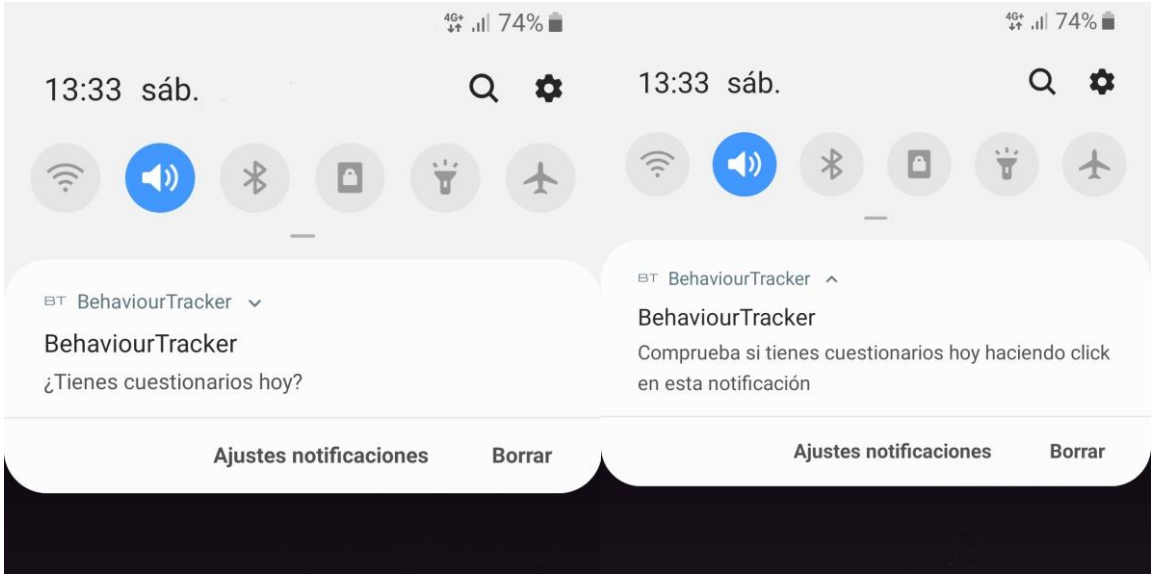


Figure 4.7: Daily Notification view.

Task 1:

This task is only performed when there are active alarms from task 0, if so, this task generates a standard notification that when pressed takes to the corresponding questionnaire of the specific time.

Figure 4.8 shows the notification generated from a questionnaire at 6:00 in the morning.

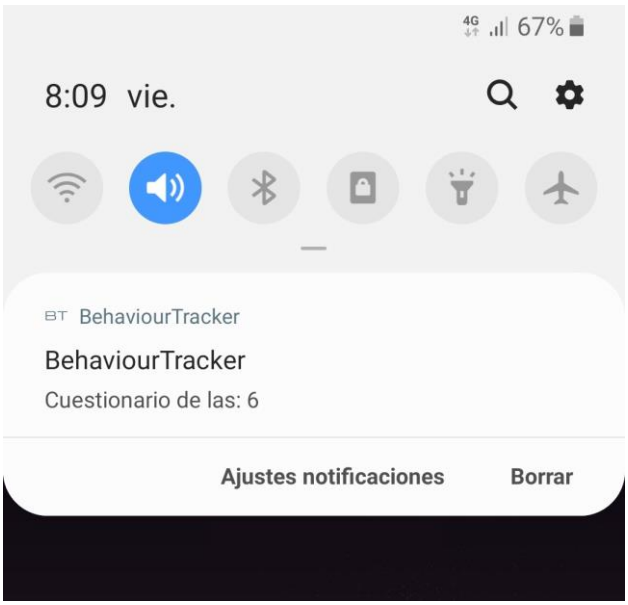


Figure 4.8: Questionnaire Notification view.

Task 2:

This last task is performed when the user logs out of the application and is responsible for removing all notifications and alarms at that time.

Synchronisation Service

This service works the same as the previous one, although in this case, it only performs one task. This is because it is intended to be extended in future work.

The task that it performs is basically to obtain from the database all the answers that there are at that moment and send them one by one to the server, every time an answer is deleted from the database when it is sent and successfully saved in the server.

This service is executed with task 0 of *NotificationService*, and every time a questionnaire is made to send the information at that moment.

4.2.5 BootReceiver

In case the mobile is restarted, a *BroadcastReceiver* is used to start the application, for which I use the following code. It has been developed following the Android specifications in [18].

Declaration in AndroidManifest.xml

```
<receiver
   android:name=".Class.BroadcastReceiver.BootReceiver"
   android:enabled="false">
   <intent-filter>
       <action android:name="android.intent.action.BOOT_COMPLETED" />
   </intent-filter>
</receiver>
```

Implementation class BootReceiver

```
public class BootReceiver extends BroadcastReceiver {
  BehaviourTrackerSystem mSystem;
   @Override
   public void onReceive(Context context, Intent intent) {
       mSystem = BehaviourTrackerSystem.getInstance(context);
       if (intent.getAction().equals("android.intent.action.BOOT_COMPLETED")) {
           if (mSystem.isUserRemembered()){
               mSystem.setDailyAlarmNotification();
               Intent newintent = new Intent(context, NotificationService.class);
               newintent.putExtra("UUID",mSystem.getUserUUID());
               newintent.putExtra("TASK_ID", 0);
               context.startService(newintent);
           }
       }
   }
}
```

Code 4.7: BroadcastReceiver.

When there is a user who is remembered, *BroadcastReceiver* is enabled for the application to run again after a restart, in case the user is not remembered this option is disabled. This function sets the daily notification and starts *NotificationService* with task 0 (as described above).

4.2.6 User interface and use

In this section, we are going to explain the different layouts that the application has as well as its normal use and some more important diagrams of what the system does.

Login screen

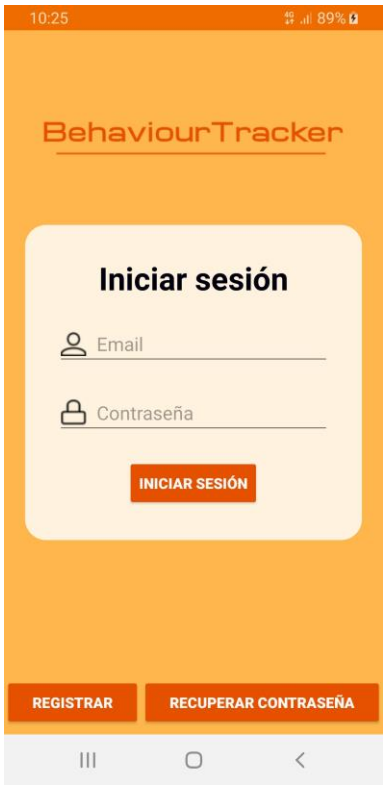


Figure 4.9: Login screen.

This screen consists of an identification form using Email and Password and a login button. At the bottom of the screen there are two buttons, the first one on the left takes to the registration layout and the second one takes to a web page to retrieve the password. Both Email and Password are checked, and if they fail, they alert to what is wrong (see **Figure 4.10** below).

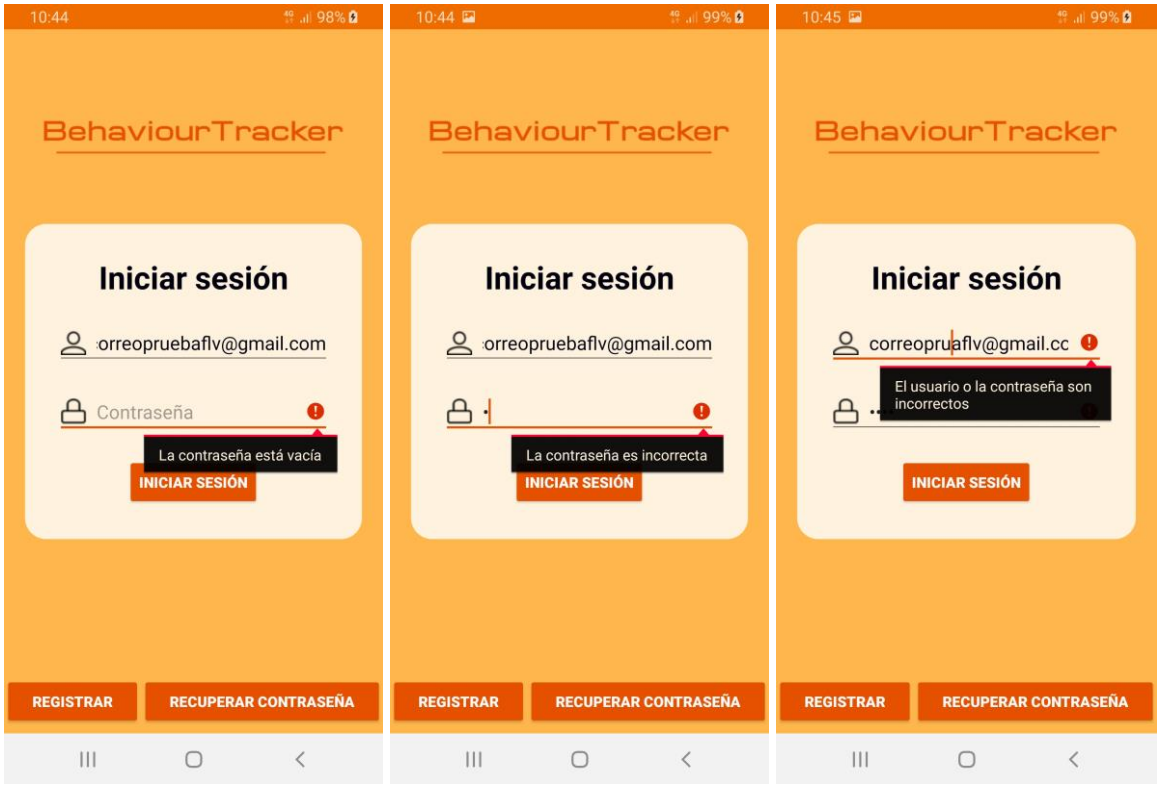


Figure 4.10: Login screen errors.

If there has been an error in the server or the user does not have Internet, a warning will appear in the form of a dialog box to know that there is a network or server error.

Figure 4.11: Network error.

The following sequence diagram represents how the system interacts when it starts and what happens if a user is remembered.

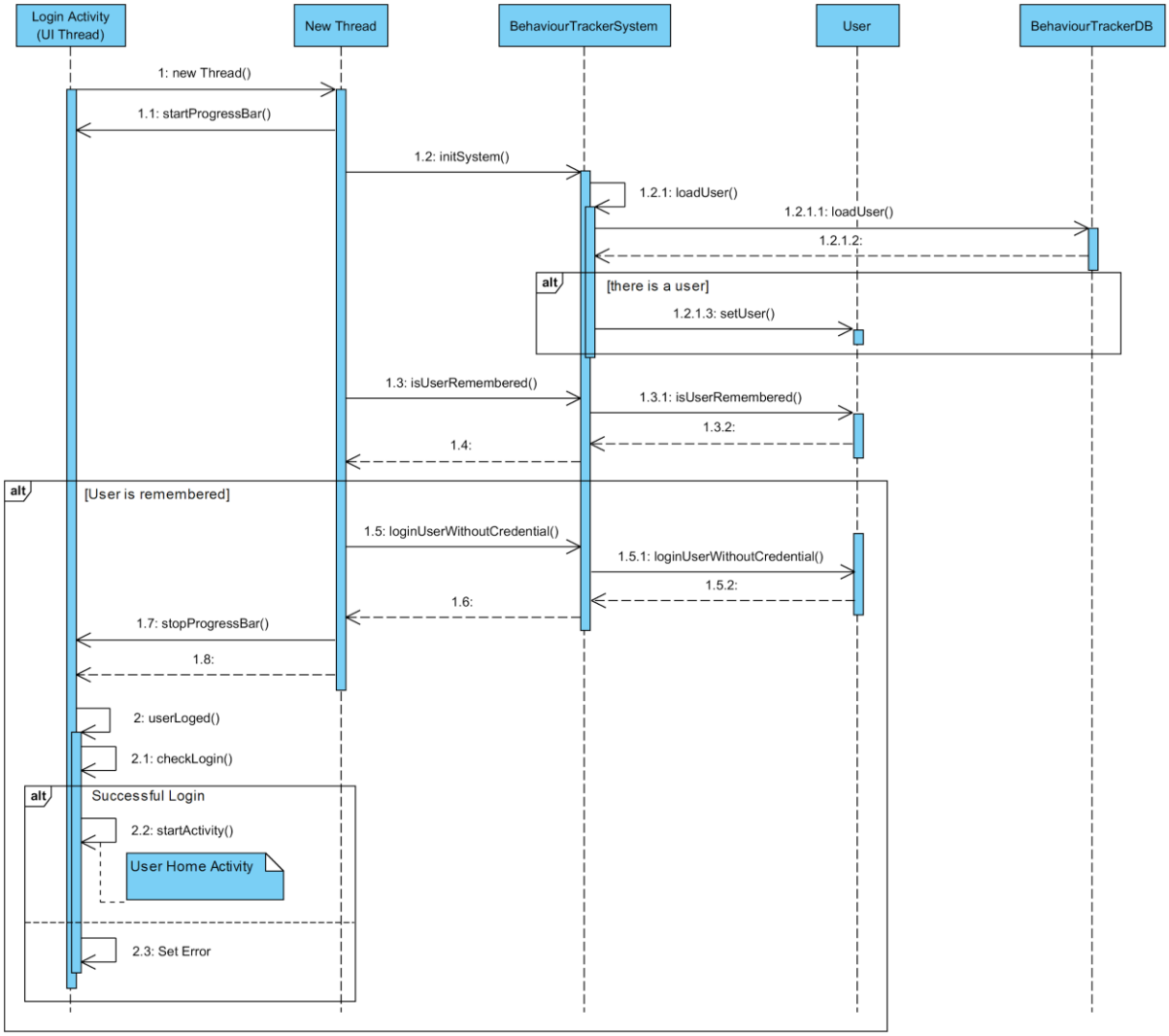


Figure 4.12: Sequence diagram when the system starts, and there is a user remembered.

As can be seen in the diagram, *LoginActivity* creates a new thread to start the system, and this is because there is an interaction with the *BehaviourTrackerDB* database and therefore this task cannot be performed from the main thread. The first call after creating the new thread is *startProgressBar()* which starts the *Loading Screen* (this layout is shown at the end of this section), so the user cannot interact with the application until the system startup has been completed. Then, the *initSystem()* function is called, which is in charge of loading the user and for this purpose it calls the *loadUser()* function to acquire the user from the *BehaviourTrackerDB* database, this returns a user if it has one and if not, it would not return anything, in case there is a user, a new instance of the *User* class is created and established. Once it has the user or not, it ask if that user is remembered, and if so, then it proceeded to log in without the need to establish credentials, for them the function *loginUserWithoutCredential()* is used. When this function is finished, it returns an error code equal to the one shown in subsection 4.2.2. This stops the *Loading Screen* and returns the control to the main thread. Now with the error code provided by *loginUserWithoutCredential(),* the function *userLoged()* is passed, and it checks it through *checkLogin()*, if the error code is 0, then the user can log in correctly. Therefore the *User Home Activity* screen starts. In case

of an error, it would appear that the user and password are incorrect and the next screen would not be accessed.

The following sequence diagram shows how the user logs in when not remembered.

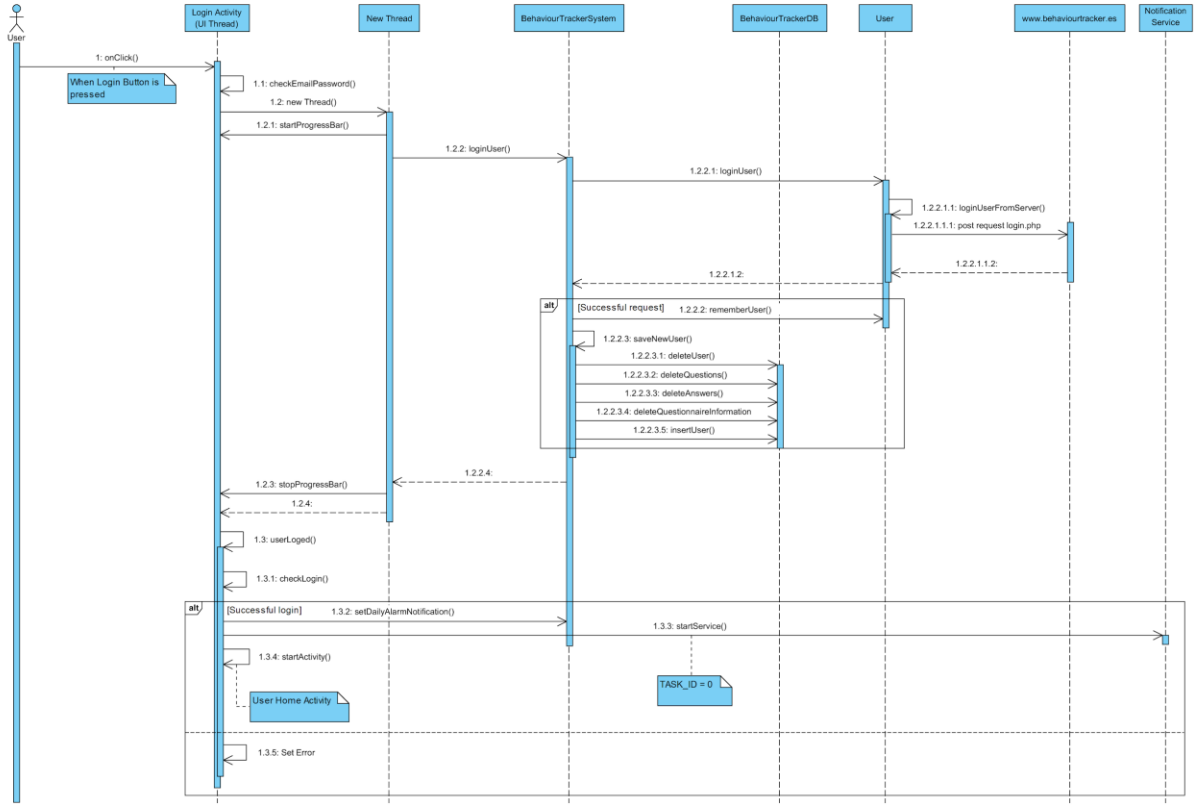


Figure 4.13: Sequence diagram when no user.

Once the system has been initialised and there is no user remembered, it is the user who is responsible for logging in by entering their credentials and pressing the login button. So once it does all this, *LoginActivity* captures the button event pressed in *onClick()*, and the first thing it does is check through *checkEmailPassword()* that the data entered by the user are an email and a password. In case it wasn't, an error would appear saying that the email is not valid, for example, because it lacks the "@", and that it hasn't introduced a password. Then, as in the previous diagram, run a new thread and start the *Loading Screen* layout so the user can't do anything while the data is being checked. Therefore the function *loginUser()* is executed. It is the User class who is in charge of logging in and has two ways to log in. The first one is that if in the login there was already a user (because when the user logs out of the application, the user is not deleted from the database but simply stopped remembering and therefore there could be a user in the database) then it checks the email and password with the user that was there. If they are correct it does not make any call to the server and if they are not then it makes the call to the server through a *POST* request as seen in the diagram, and it returns a *JSON* as seen in subsection 4.2.2. If the request has been successful, then the user is remembered by *rememberUser(),* and a new user is saved *saveNewUser()* although this function takes care of deleting all the data from the previous user and inserting a new user in the *BehaviourTrackerDB* database. All this according to the error code as it has been mentioned several times before. It finishes the thread returning the error code corresponding to the previous function calls and stopping the *Loading Screen*. As

in the previous diagram, the function *userLoged()* is called to log in. First, the error code is checked in *checkLogin()*. If it is successful, then the daily alarm of 00:00 hours is set by *setDailyAlarmNotification()* that performs task 0 of *NotificationService* explained above in subsection 4.2.4. The next thing it does is to execute task 0 of *NotificationService* in case the user is already active and has questionnaires for that day that can be performed and not wait for the next day. Finally, it would start the *User Home Activity* screen. In case of error, as in the previous diagram, it would be announced to the user.

```
private boolean checkLogin(int code){
   boolean canLogin = false;

   switch (code){
       case 0:
           canLogin = true;
           break;
       case 1:
           String noUserOrPassword =
getResources().getString(R.string.no_user_or_password);
           mTextView_email.setError(setSpannableError(noUserOrPassword));
           mTextView_password.setError(setSpannableError(noUserOrPassword));

           break;
       case 2:
           String passwordIncorrect =
getResources().getString(R.string.password_incorrect);
           mTextView_password.setError(setSpannableError(passwordIncorrect));
           break;
       case 3:
           new AlertDialog.Builder(this)
                                   .setTitle(R.string.notice)
                                   .setMessage(R.string.network_error)
                                   .setPositiveButton(R.string.accept, new
DialogInterface.OnClickListener() {
                                       @Override
                                       public void onClick(DialogInterface
dialogInterface, int i) {
                                           dialogInterface.dismiss();
                                       }
                                   }).show();
           break;
   }

   return canLogin;
}
```

Code 4.8: Function checkLogin().

Sign up screen

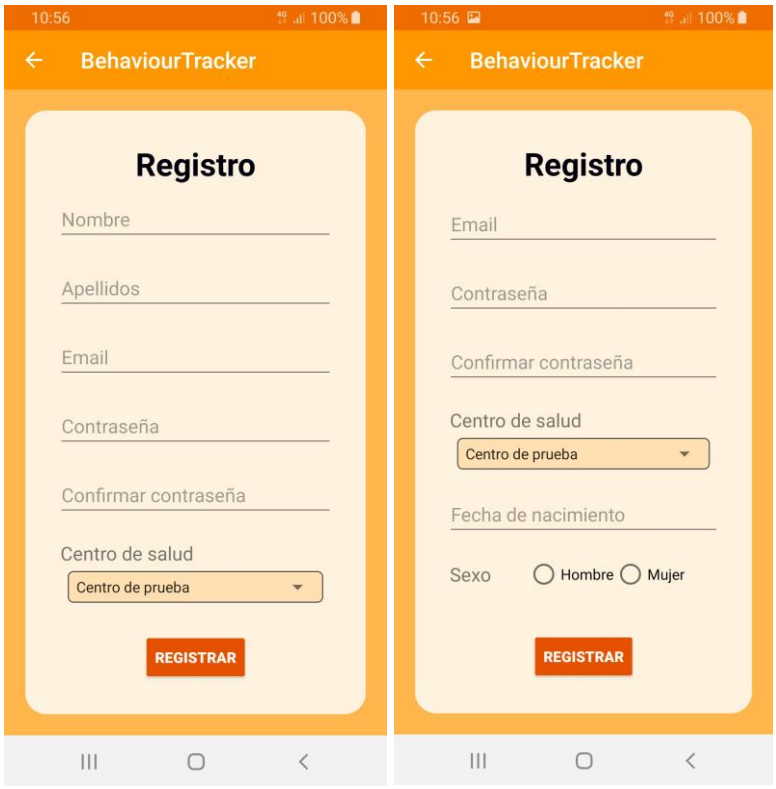


Figure 4.14: Sign up screen.

This layout is composed of a scrollbar containing a form with all the entries of the record. All of them have to be filled out in order to register, and all of them are checked before sent to make sure the information is correct.

The most important thing in this layout is that the email is checked to make sure that it does not exist and that the password has a series of requirements such as that it must have between 6 and 20 characters and at least one lower case, one upper case, one alphanumeric character and one digit, as well as that it coincides with the password confirmation field. On the other hand, when the layout is loaded, the health centres available through the server's database are obtained. Finally, when the user presses the register button, it is sent to the server, and if the data is correct, the server is in charge of generating a UUID for that user.

Once the user registers, a dialog box will appear to tell him whether he wants to log in or not.

Figure 4.15: Login confirmation after registration.

User home screen

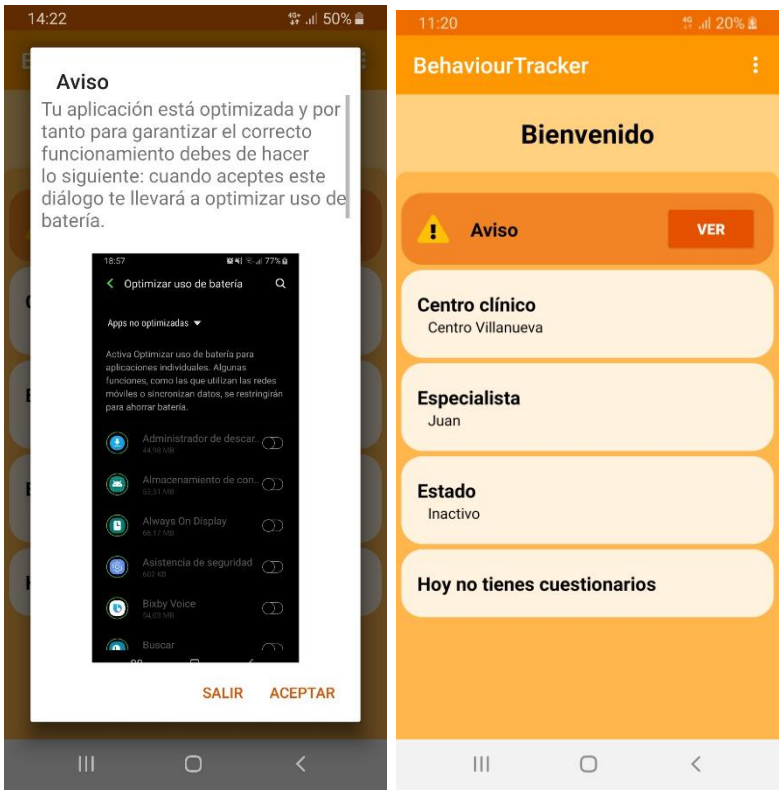


Figure 4.16: User home screen after first login.

If this is the first time that the application is installed. Once the user logs in and acceses the initial welcome screen, a dialog box will appear warning that it needs to update the battery usage for this application to work, this is because Android does not allow the use of normal alarms when it is in sleep or standby mode from version 6. 0 (API 23), for this, it has to use some special functions as shown in [19]. Still, there is another problem, and that is that each smartphone manufacturer by using its abstraction layer on Android makes the alarms still not work even doing it as said in the previous documentation. Therefore it has to use the white list. This white list is the applications that are exempt from battery optimisation. It is the user's responsibility to make the application not to be optimised because by default, when an application is installed on Android, it is automatically optimised. Besides that, it can't be done automatically due to each manufacturer's restrictions.

The warning, as shown in the right image of Figure 4.16**,** will continue to appear until the user does so in order for the application to function normally. When he does so, the warning will be removed (see Figure 4.17).

After the notice has been removed the welcome screen shows the most relevant data such as the clinical centre, the specialist who is taking the user at that moment, the state in which the user is and the user’s questionnaires that have to the current day. These data are updated every time you enter this screen, for the centre, specialist and status an *HTTP* call is made to the server to update them, and for the questionnaires, the database is accessed, and the information is loaded from *QuestionnaireInformationDB*.
The status can be active or inactive. This means that when a user is inactive, it means that he or she is registered at the indicated plant and with the indicated specialist but has not yet been confirmed by the plant or because the user is registered but not activated by any specialist. The user must, therefore, wait until the user has been confirmed and is active.

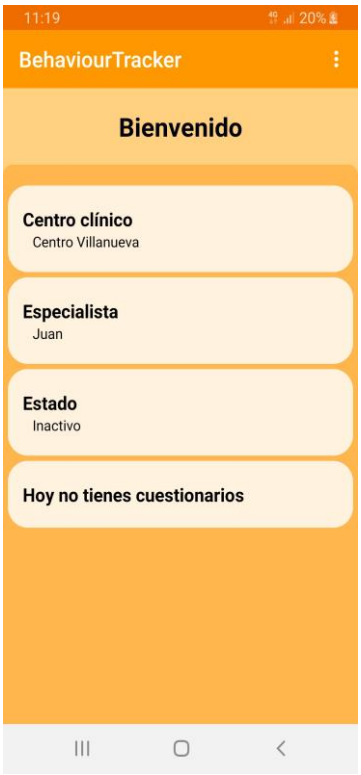


Figure 4.17: User home after the warning.

As for the questionnaires, if one day, the user does not have questionnaires, it will look like in Figure 4.17.

When the user has questionnaires for that day, the application shows how many questionnaires there are, the time he has the questionnaire and the number of questions he has. If the time is before the time of the quiz, they cannot be taken, and the button with the most transparent background will be shown to highlight that it is not enabled. If, on the contrary, this background can be done, it will be highlighted. In the case of a questionnaire, the information will be green and instead of a button, a "DONE" text will appear.

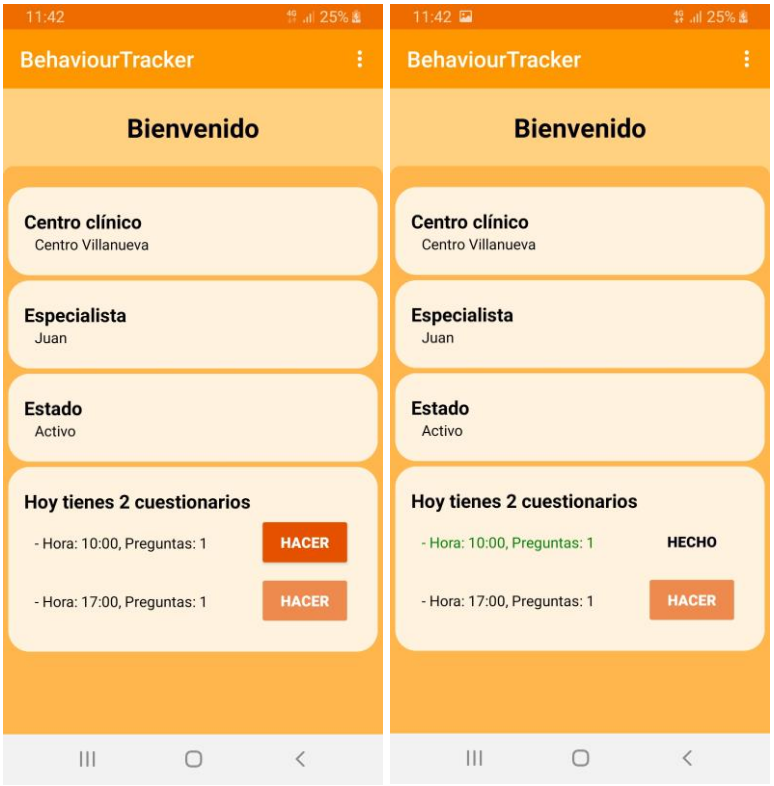


Figure 4.18: User home how are the forms shown.

Profile screen

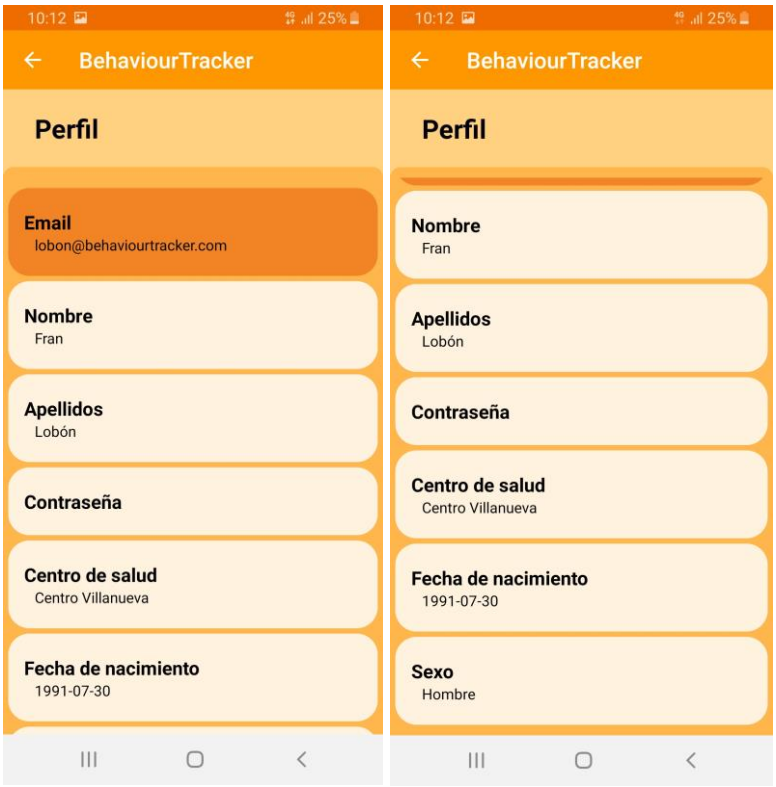


Figure 4.19: Profile screen.

Figure 4.20: Floating menu.

From the floating menu (Figure 4.20), the profile can be accessed and logged out. The profile (see Figure 4.19) shows all the personal data entries that the user had to fill in when registering. By clicking on any entry, the user can change it. All entries can be changed except the Email that cannot be changed, so it appears in another colour.

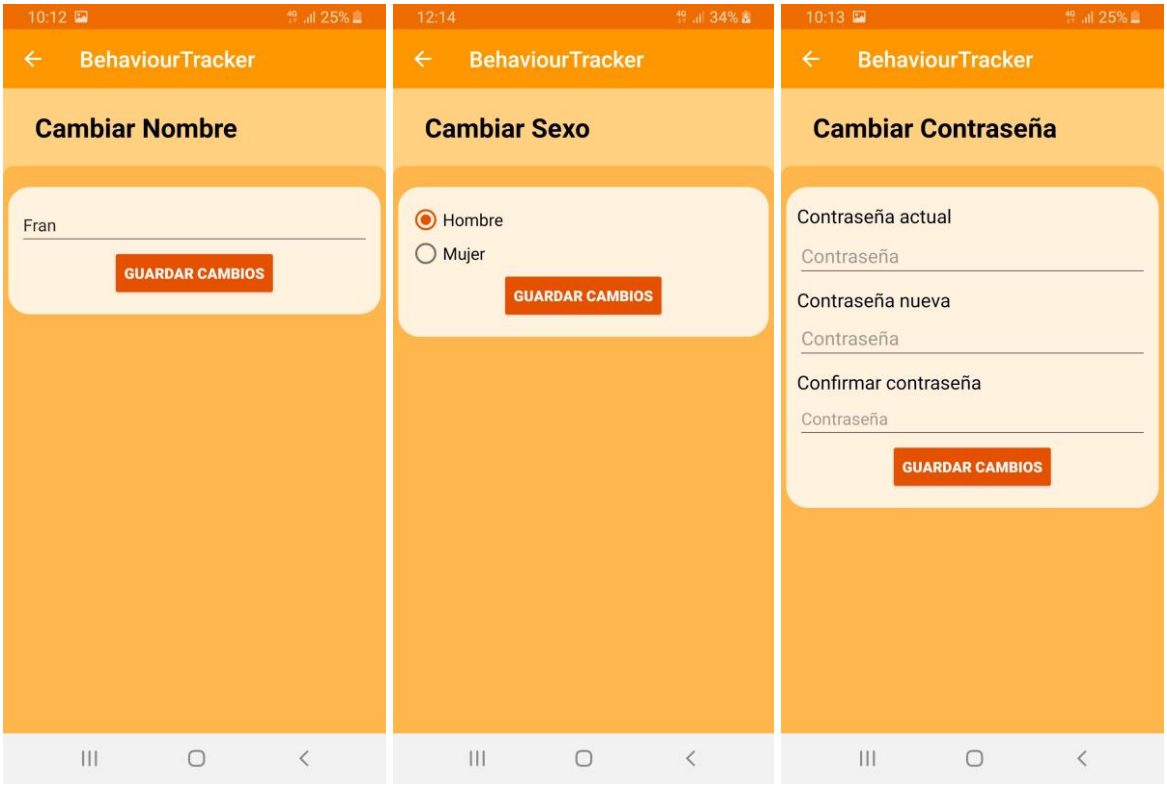


Figure 4.21: Profile screen, changing data example.

In the figure above, it can be seen different data that can be changed and the way in which they are changed, in the first image, for example, the user can change the name by writing the new one and saving the changes, in the second image the user select the sex and in the third image the user must enter the current password along with the new one and the confirmation of it so that it can be changed. All data are checked before being sent.

Questionnaire screen

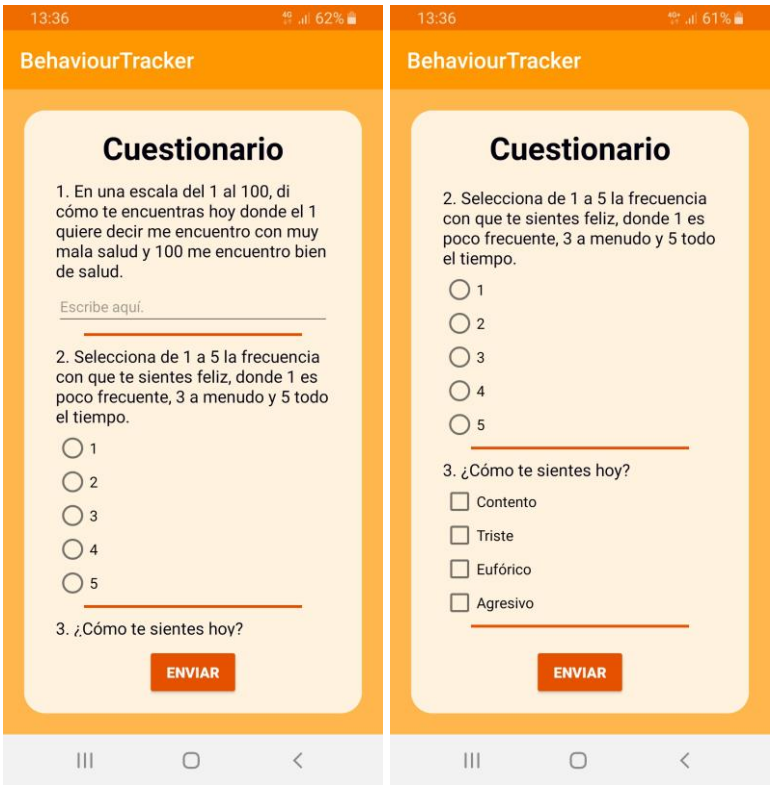


Figure 4.22: Questionnaire screen.

The questionnaires can be accessed through the notification or through the application itself as we saw in the user view. This layout is composed of a scrollbar to be able to scroll through all the questions the user has for that specific day and time. When the user has finished answering the questions, he will press the *SEND* button, and it will be sent to the server through the *SynchronizationService* if there is a connection, in case of error they will remain stored in the database until a new synchronisation.

The questions will be of three types: text, single selection and multiple selection.

- Text: question 1 in Figure 4.22 in which we can put a text answering the question.
- Single selection: question 2 in Figure 4.22 where we can select a single option from those available, if another one is selected it will be changed to the last one we have selected.
- Multiple choice: question 3 in Figure 4.22 allows the user to select several options from those available.

With these three types, there will be more flexibility in creating questions.

This layout is dynamically created according to the questions available, their type and the sequence each one has.

Loading screen

Apart from this, the application has a loading screen when performing database and server call operations (Figure 4.23). This screen serves to prevent the user from doing anything while the data is being loaded so that everything works correctly.

Figure 4.23: Loading screen.

4.3 Web Application

As described above, the website has been designed to provide the clinical centre with an intuitive interface for managing both the mobile application users and the centre's users and managing questions and categories. It will also allow specialists to have control of their users in order to create questionnaires in an interactive table and see the results of these questionnaires.

The web design has been done with *HTML5*, *AJAX*, *Bootstrap 4*, *Javascript*, *CSS* and *PHP 7*. In addition, a database has been implemented to store all the data of the web application through *MYSQL*.

The following subsections will describe each of the functions performed by the web application as well as some implementations of the most important parts.

Below is a list of the subsections to be discussed:

- Database.
- Structure of the web.
- User identification on the web.
- User management.
- Questions management.
- Category management.
- Centre management.
- Mobile application request management and password recovery.

Before entering to develop each one of these subsections, it is going to be shown how the scripts are structured in the web using a sitemap.

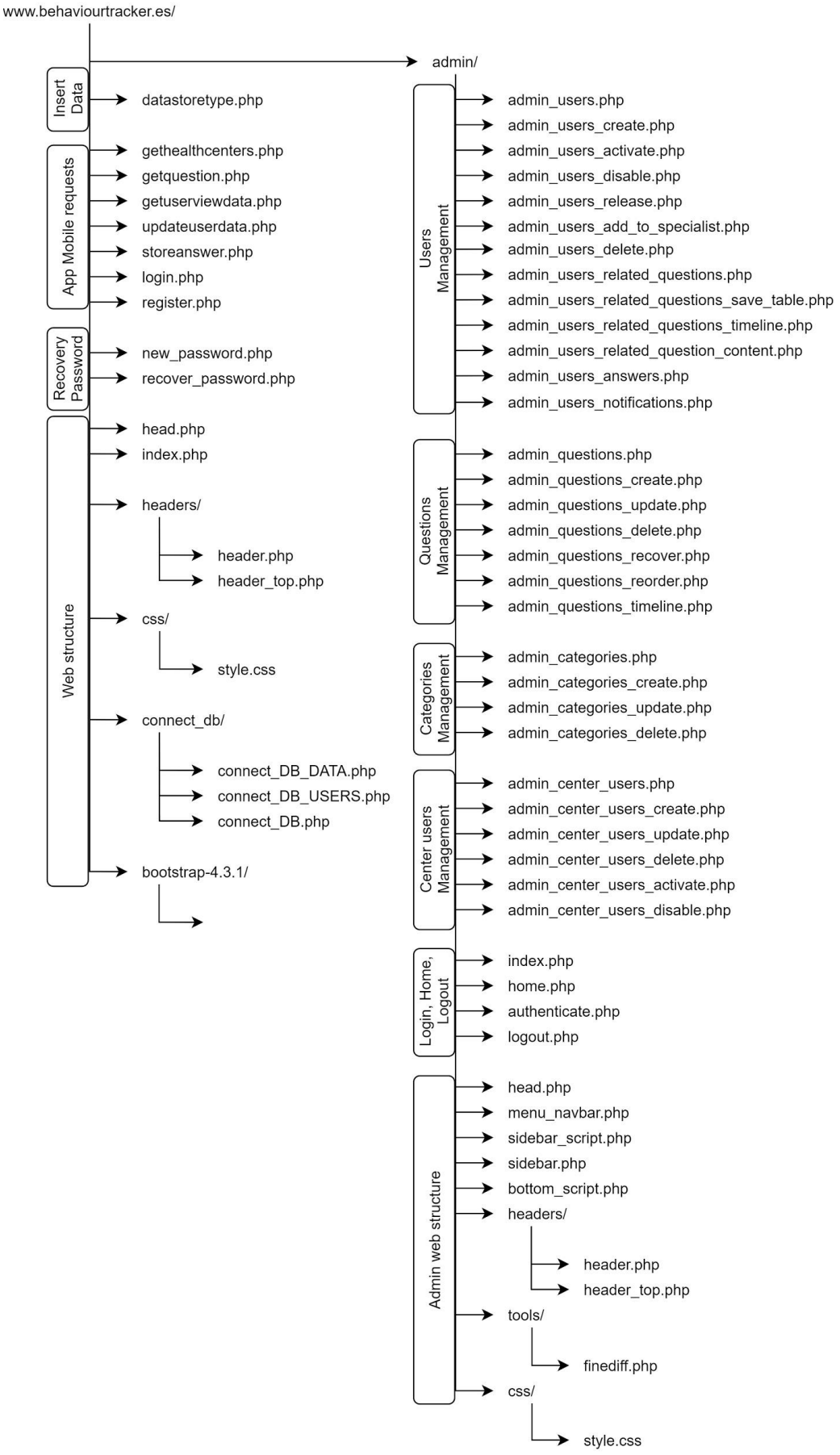


Figure 4.24: Sitemap representation.

4.3.1 Database

The data generated in the system is stored in two databases, on the one hand, there is the database where only normal data is stored, and on the other hand, there is the database where all users, as well as the sites, are stored. This is designed so that normal data is decoupled from the more sensitive data.

Database Data

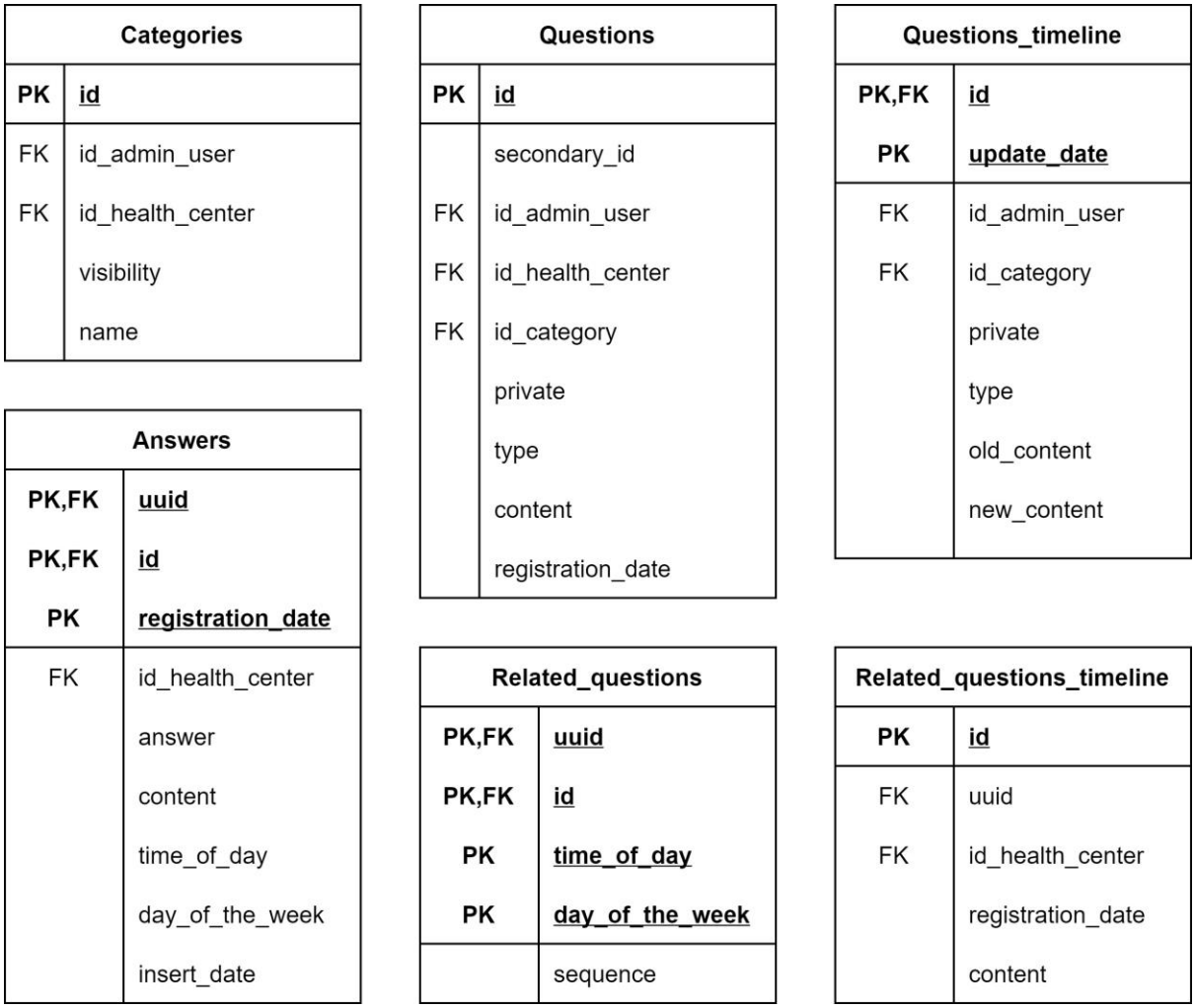


Figure 4.25: Database Web Data.

In the figure above, it can be seen the database where only the data is stored as it is:

- Categories: it is associated with a centre and user of the centre (which in this case are always associated with the administrator of that centre).
- Questions: it is associated with a user of the centre, centre and a category. The *content* field stores in *JSON* format the question with its possible options.
-

Without options: {“question” : “¿How are you?”}

With options: {“question” : “¿How are you in a scale 1 to 5?”, “options” : [“1”,“2”,“3”,“4”,“5”]}

Figure 4.26: Question representation in JSON format.

- Questions timeline: the *ID* and the update date are the primary key of the *Questions* table. It is associated with the user of the centre that updated the question and the current category. In this history, both the old and the new content is saved. Later in the question management section, we will discuss why this is so.

- Answers: With primary key, the *UUID* of the Users table from the Users database and the *ID* of the *Questions* table and the registration date. In this table the *insert_date* row serves to know when the question has arrived at the server, this will help for future work.

- Related questions: This table is the relationship between a user and a question. That’s why it uses as primary key the *UUID* of the *Users* table from the *Users* database and the *ID* of the *Questions* table as well as the time and day of the week the user has to answer that question. It also has a sequence to know the order of the questions in the questionnaire. The keys are made in this way so that the same question cannot be answered in a questionnaire.

- Related questions timeline: It is associated with a user of the application and a centre. This table saves in the *content* row in *JSON* format all the related questions that are saved every time the related questions table of an application user is saved. This will be explained later in the user management section.

Row content: [["1","1","10","1"],["2","2","10","1"],["5","5","10","1"], ...]

Data representation of each array: [“id”,”sequence”,”time_of_day”,”day_of_the_week”]

Figure 4.27: Table: Related questions timeline. Row: content, information detail.

Database Users

Users	
PK	**uuid**
PK	**email**
FK	id_admin_user
FK	id_health_center
	active_user
	verified
	to_remove
	name
	surname
	password
	date_of_birth
	gender
	registration_date

Admin_users	
PK	**id**
PK	**email**
FK	id_health_center
	active_user
	rol
	password
	name
	surname
	registration_date

Health_centers	
PK	**id**
FK	id_initial_category
	active
	name

Password_reset	
PK,FK	**email**
PK	**token**
PK	**registration_date**

Figure 4.28: Database Web Users.

In the figure above, you can see the database where the sensitive system data is stored:

- Users: The primary key is the unique identifier *UUID* and *email*. Each user has associated a user of the centre and a centre. The field *active_user* represents if the user is active or not. If he is active it means that a specialist is following him, and if not, it means that the user is inactive inside the centre. The field *verified* represents when a user registers and has not yet been confirmed by the plant. The *to_remove* field represents that the user wants to be removed from the system. Users use a *UUID* instead of an *ID* to be differentiated within the system from others.

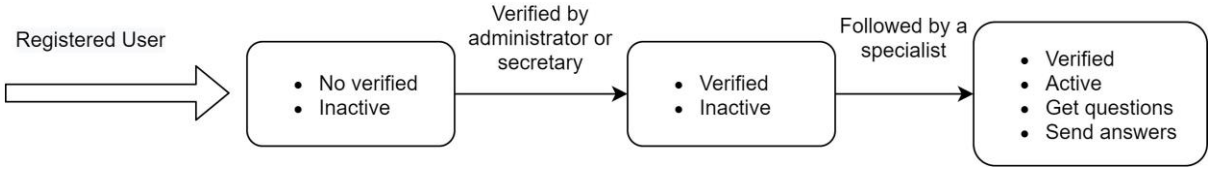


Figure 4.29: User verification and activation.

- Admin users: Just like users use email and an identifier as their primary key. It is associated with a centre. The field active_user represents that a specialist is active or not.

- Health centers: It is associated with an initial category. This category is the only one that cannot be removed from the center, besides it makes the system of questions and categories work. The active field represents whether the center is active or not.

- Password reset: It has as primary key the email from the Users table, a token and a registration date. This table is used for password recovery. Later we will explain how it works.

Connection

Three different codes are used to connect to the database depending on which database we need at any given time or if we need both databases.

DB connection Data

```php
<?php
    $db_name_data = "behaviourtracker_data";
    $mysql_username = "BehaviourTracker";
    $mysql_password = "password";
    $server_name = "localhost";
    $DB_DATA = mysqli_connect($server_name, $mysql_username, $mysql_password,
$db_name_data);

    if(!$DB_DATA){
        die("Connection failed: ".mysqli_connect_error());
    }
?>
```

DB connection Users

```php
<?php
    $db_name_users = "behaviourtracker_users";
    $mysql_username = "BehaviourTracker";
    $mysql_password = "password";
    $server_name = "localhost";
    $DB_USERS = mysqli_connect($server_name, $mysql_username, $mysql_password,
$db_name_users);

    if(!$DB_USERS){
        die("Connection failed: ".mysqli_connect_error());
    }
?>
```

Connection to both DB

```php
<?php
    $mysql_username = "BehaviourTracker";
    $mysql_password = "password";
    $server_name = "localhost";
```

```
    $DB = mysqli_connect($server_name, $mysql_username, $mysql_password);

    if(!$DB){
        die("Connection failed: ".mysqli_connect_error());
    }
?>
```

Code 4.9: DB Connections.

```
SELECT * from behaviourtracker_users.users,
behaviourtracker_data.related_questions_timeline where … ;
```

Code 4.10: Example of how to use both DB.

4.3.2 Web Structure

The website is structured in two ways, the first in user identification and password recovery and the second, when an administrator enters the dashboard.

The entire website uses a responsive design thanks to *Bootstrap 4*, so it can be used on any device. However, some features are not supported for mobile devices.

The two main structures are shown in the following figures.

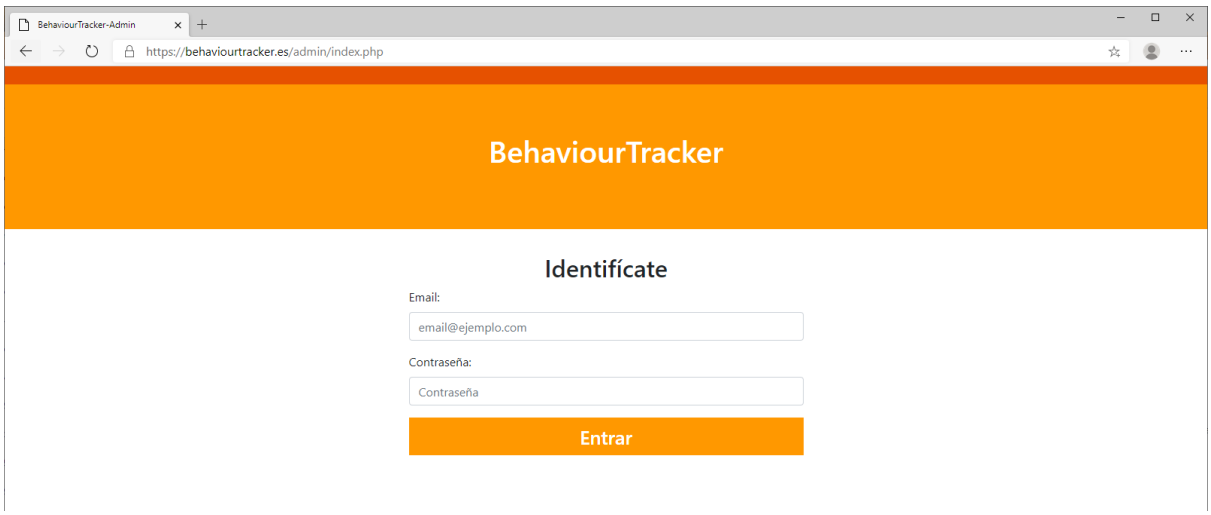


Figure 4.30: Web structure, first view.

This first structure is simple and has a basic structure with a header composed of the header top and the main header followed by the content of the website. It is mainly used when a user of the centre is identified and for password recovery.

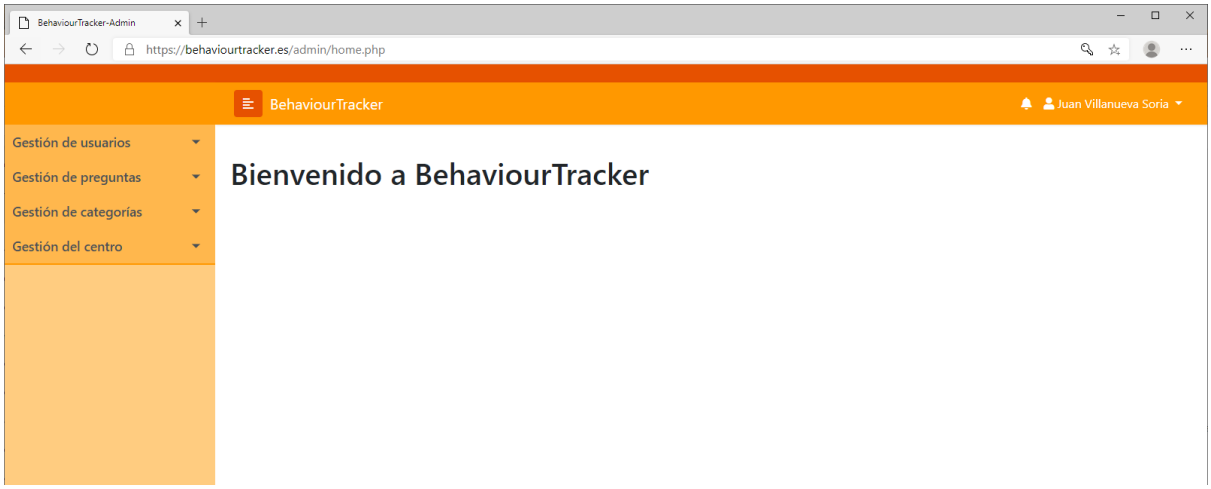


Figure 4.31: Web structure, home.

```php
<?php
session_start();

if (!isset($_SESSION['loggedin'])) {
    header('Location: index.html');
    exit();
}
?>

<!DOCTYPE html>
<html lang="es">
    <head>
        <title>BehaviourTracker-Admin</title>
        <?php require_once("head.php");?>
    </head>

    <body>
        <div class="wrapper">
            <?php include('sidebar.php'); ?>
            <div class="" id="content">
                <?php include('menu_navbar.php'); ?>
                <div id="main-content" class="col-sm-12">
                    <h1>Bienvenido a BehaviourTracker</h1>
                </div>
            </div>
     </div>

     <?php require_once("bottom_script.php");?>
     <?php require_once("sidebar_script.php");?>

    </body>

</html>
```

Code 4.11: Web structure, home.php.

In the dashboard, it can be seen that there is a structure formed by a header, a navigation menu, a sidebar and the main content of the website.

The navigation menu from left to right is composed of a button that serves to hide or unfold the sidebar, the name of the website, a bell that serves to know if a new user has registered and the user who has logged in which is a drop-down list where it contains a button to close the session.

The highlights of all this are the bell and the sidebar.

On the one hand, the bell uses a *Javascript* code that runs every 20 seconds (although this time can be increased so that it takes longer and does not make as many requests to the database), and serves to notify the administrator or secretary (the specialist does not register users, so the bell does not appear) that there is a new user registered in the centre.

```
<script type="text/javascript">
  function load_notification(view = ''){
    $.ajax({
      url:"admin_users_notification.php",
      method:"POST",
      dataType:"json",
      success:function(data){
        $('#notification-content').html(data.notification_content);
        if(data.notification_count > 0){
          $('#notification-count').html(data.notification_count);
        }else{
          $('#notification-count').html('');
        }
      }
    });
  }

  jQuery(document).ready(function($) {
    load_notification();

    $(document).on('click', '#notificationMenu', function(){
      load_notification();
    });

    setInterval(function(){
      load_notification();
    }, 20000);
  });
</script>
```

Code 4.12: New registered user notification.

The code above, calls the **admin_users_notification.php** script which returns the email, first and last name of each unconfirmed user as well as the number of unconfirmed users. With this data, it generates the notification that we see below.

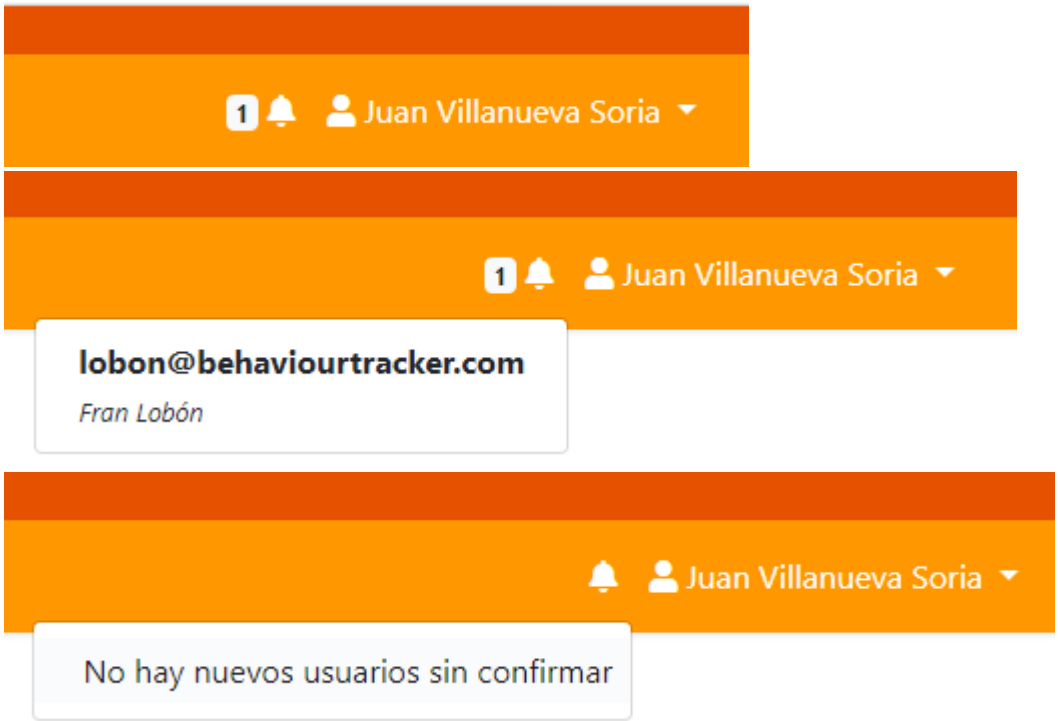


Figure 4.32: No unconfirmed new users.

This bell is a drop-down menu in which a number appears on the left, indicating that there are one or more users. If it is displayed the email and the name of the new user appears, if we click on the user, it sends us to the confirmation page which will be discussed in another section later. Once there are no more unconfirmed users, if the bell is displayed, a message appears saying that there are no new unconfirmed users.

On the other hand, the sidebar is used to navigate through the website.

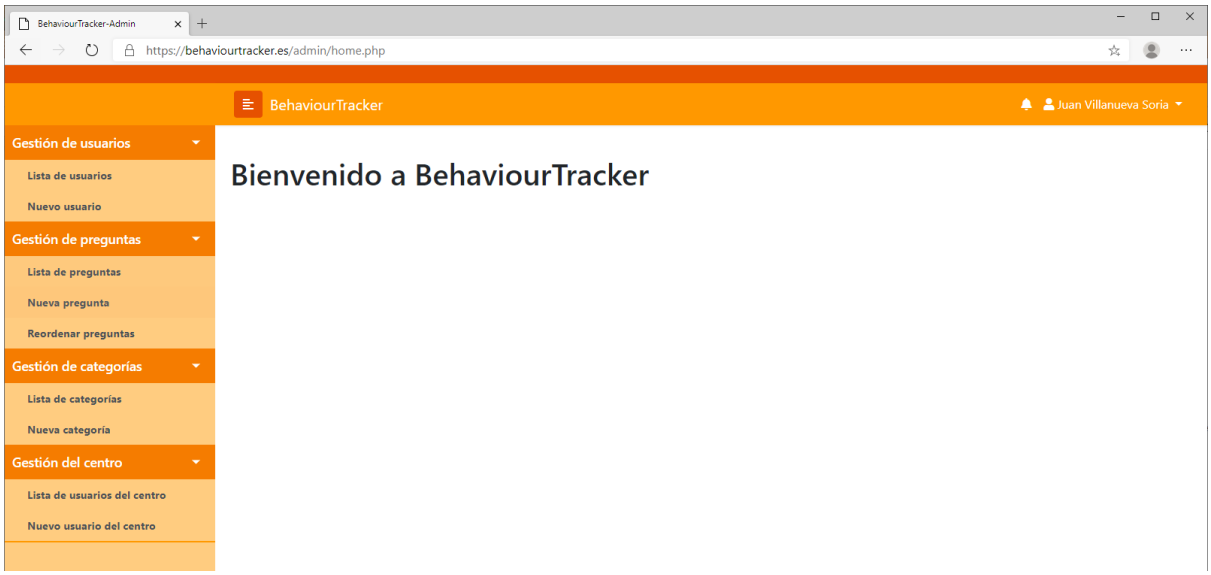


Figure 4.33: Sidebar content.

Before continuing with the following sections, we will comment that this website uses tables to display content, such as categories (Figure 4.33). The Javascript DataTable API [20] has

been used to create the tables. This API provides everything I need to display the data in table form, as well as paging, a search engine, etc.

DataTable inicialization

```
<script type='text/javascript'>
      $(document).ready(function () {
        $('#category-table').DataTable({
          "language": {
            "url": "//cdn.datatables.net/plug-ins/1.10.15/i18n/Spanish.json"
          },
          scrollY: '50vh',
          scrollX: true,
          scrollCollapse: true,
          paging: false
        });
        $('.dataTables_length').addClass('bs-select');
      });
</script>
```

Table html

```
<div class="table-responsive">
  <table id="category-table" class="table table-bordered table-striped table-sm"
cellspacing="0" width="100%">
    <thead>
      <tr>
        <th class="th-sm">#</th>
        <th class="th-sm">Nombre</th>
        <th class="th-sm">Visible</th>
        <th class="th-sm">Acciones</th>
      </tr>
    </thead>
    <tbody>
      <?php
      for($i=0; $i < count($rows_category); $i++) {
        $row = '<tr>';
        $row .= '<td class="center-content-table text-center">'.$i.'</td>';
        $row .= '<td class="center-content-
table">'.$rows_category[$i]['name'].'</td>';
        if($rows_category[$i]['visibility']) {
          $row .= '<td class="center-content-table">Si</td>';
        }else{
          $row .= '<td class="center-content-table">No</td>';
        }
        $row .= ' <td class="text-center center-content-table">
                    <a class="btn btn-secondary"
href="admin_categories_update.php?id='.$rows_category[$i]['id'].'">Modificar</a>
                    <a class="btn btn-secondary"
href="admin_categories_delete.php?id='.$rows_category[$i]['id'].'">Eliminar</a>
```

```
                    </td>';
            $row .= '</tr>';

            echo $row;
        }
        ?>
      </tbody>
    </table>
</div>
```

Table view

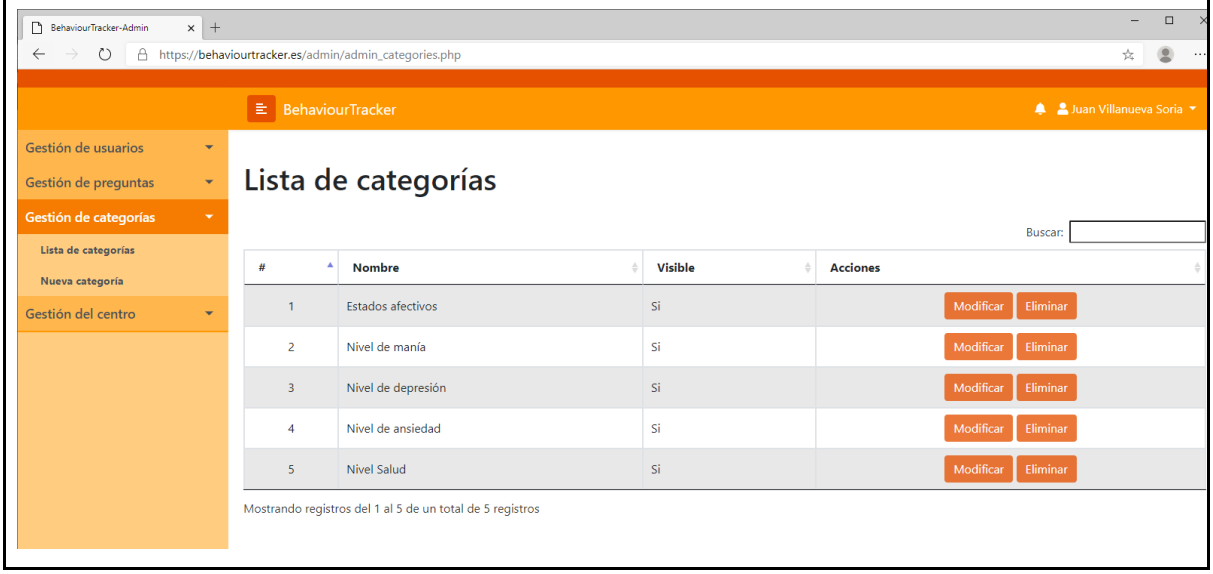


Figure 4.34: Datatable code and view example.

4.3.3 Web user identification

The identification of users of the centre from the web is made using *PHP* sessions for which I have used the following code.

```
<?php
session_start();

if(isset($_POST['email']) && !is_null($_POST['email']) && $_POST['email'] != ""){
  if(isset($_POST['password']) && !is_null($_POST['password']) &&
$_POST['password'] != ""){

    require "../connect_db/connect_DB_USERS.php";
    $stmt = mysqli_stmt_init($DB_USERS);
    $query = "SELECT email, password, admin_users.name, surname, verified_user,
admin_users.id, id_health_center, rol, health_centers.id_initial_category,
health_centers.active from admin_users, health_centers where
admin_users.id_health_center = health_centers.id and email = ? ;";
```

```
    $params = array('email' => "",
                    'password' => "",
                         ...
                   );

    if(mysqli_stmt_prepare($stmt,$query)){
      mysqli_stmt_bind_param($stmt, "s", $_POST['email']);
      mysqli_stmt_execute($stmt);

      mysqli_stmt_bind_result($stmt,$params['email'],
                              $params['password'],
                                   ...
                              );

      if(mysqli_stmt_fetch($stmt)){
        mysqli_stmt_close($stmt);

        if($params['verified_user'] && $params['active']){
          if (password_verify($_POST["password"],$params['password'])) {
            session_regenerate_id();

            $_SESSION['loggedin'] = TRUE;
            $_SESSION['name'] = $params['name'];
            $_SESSION['surname'] = $params['surname'];
            $_SESSION['email'] = $_POST['email'];
            $_SESSION['id'] = $params['id'];
            $_SESSION['id_health_center'] = $params['id_health_center'];
            $_SESSION['rol'] = $params['rol'];
            $_SESSION['id_initial_category'] = $params['id_initial_category'];

            header("location: home.php");

                   ...
```

Code 4.13: Authenticate.php.

This code is executed when we log in. First, we verify that the email and password are correct and then we build a *PHP* session with the values that are passed to *$_SESSION* that will be necessary to make the web work and to restrict access to certain parts of the code that depends on the user of the centre.

The following code shows how to log out if the user is not logged in.

```
<?php
session_start();

if (!isset($_SESSION['loggedin'])) {
    header('Location: index.html');
    exit();
```

```
}
?>
```

Code 4.14: Sign out if the user is not logged in.

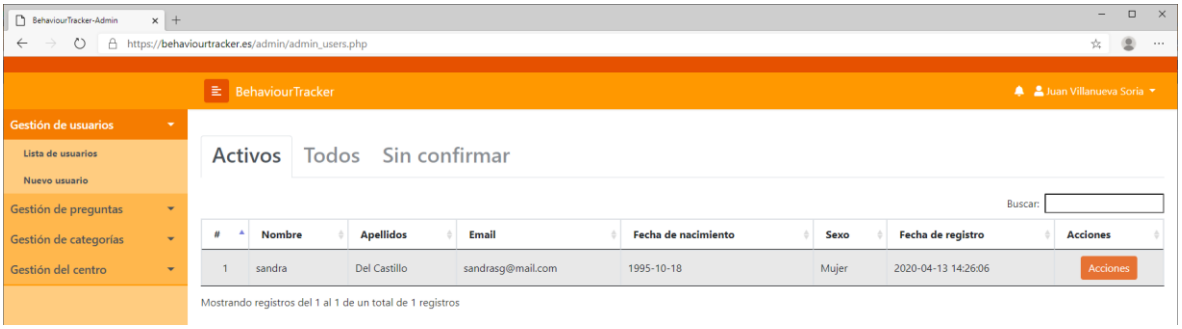
4.3.4 User management

The mobile application users can be managed by all the users of the centre, but each of the users of the centre will have its limitations. All this is covered in chapter 3.

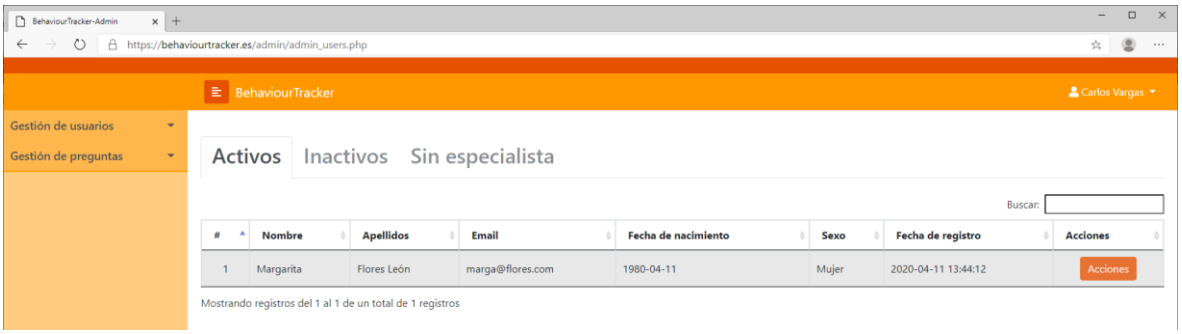
In this section, each of the scripts that make up the user management will be defined.

admin_users.php

Admin view



Specialist view



Secretary view

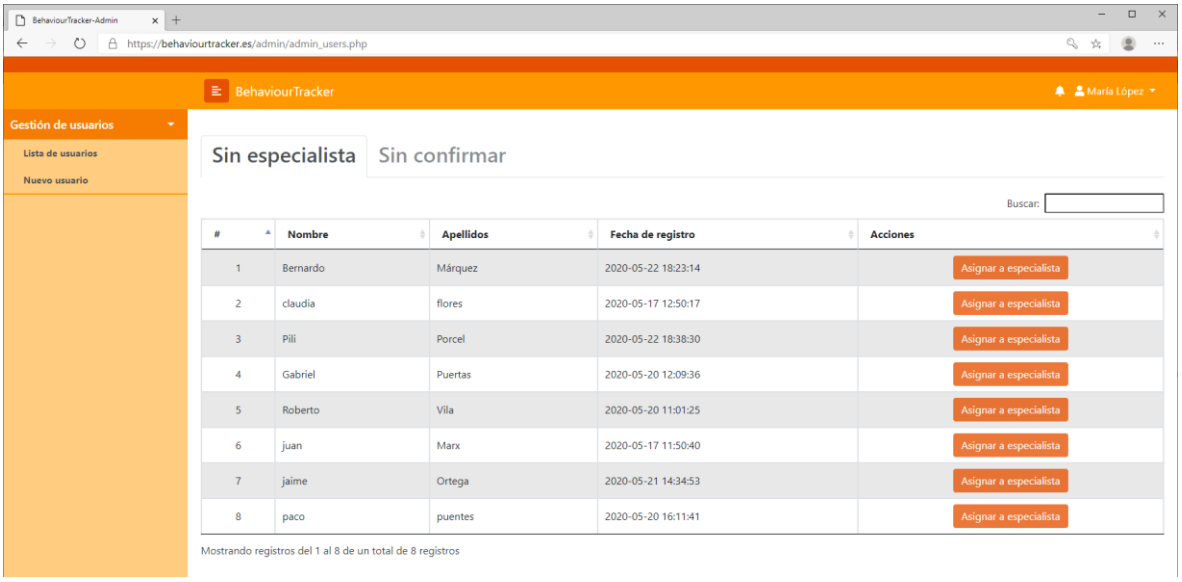


Figure 4.35: Users list.

The user list can be accessed through the sidebar in user management and user list.

This script is in charge of showing the user lists as well as providing the actions that can be performed. Depending on the status of the user, he or she will be located in one or another of the existing tabs. The figure above shows the view of each user in the centre.

The tabs are designed using the *Bootstrap* tab style.

```
<ul class="nav nav-tabs" role="tablist">
  <li class="nav-item">
    <a class="nav-link active" data-toggle="tab" href="#myUsers" role="tab" aria-
controls="myUsers" aria-selected="true"><h2>Activos</h2></a>
  </li>
  <li class="nav-item">
    <a class="nav-link" data-toggle="tab" href="#allUsers" role="tab" aria-
controls="allUsers" aria-selected="false"><h2>Todos</h2></a>
  </li>
                              ...
</ul>

<div class="tab-content">
  <div class="tab-pane fade show active" id="myUsers" role="tabpanel" aria-
labelledby="myUsers-tab">
 <br>
 <div class="table-responsive">
  <table id="user-table"
```

Code 4.15: Tabs code example.

In the code above, it can be seen how the tabs are created, using a list of tabs. Then the *DIV* is used with the *CSS* class *tab-content* and the next *DIV* that has the identifier of the first tab of the list, and that contains the main content of the tab that in this case is the table of users of the administrator.

All the tabs have been made in the same way in the rest of the website.

In the administrator's view, it can be seen three tabs:

- “Activos”: List of users that are currently active.
- “Todos”: List with all users whether they are active or not, or are from another specialist.
- “Sin confirmar”: List of users that have registered with this centre from the mobile application and are currently unconfirmed.

In the specialist view, there are also three tabs:

- “Activos”: As in the administrator view, it shows a list with the users that the administrator has active.
- “Inactivos”: List with the users that the administrator has associated with it but that are inactive.
- “Sin especialista”: List with the inactive users that are associated with the administrator. This is done in this way because a user has to be associated with someone and therefore as the user that will never disappear from the centre is the administrator. An inactive user associated to the administrator can be activated by another specialist and at that moment becomes part of the active or inactive users of the specialist that has followed him. We will discuss later how a specialist can release a user to be followed by another specialist.

Finally, in the secretary's view, there are two tabs:

- No specialist: As in the specialist view, it shows the inactive users that are associated with the administrator.
- Unconfirmed: Same as in the administrator view, shows the users registered in the centre but who are unconfirmed.

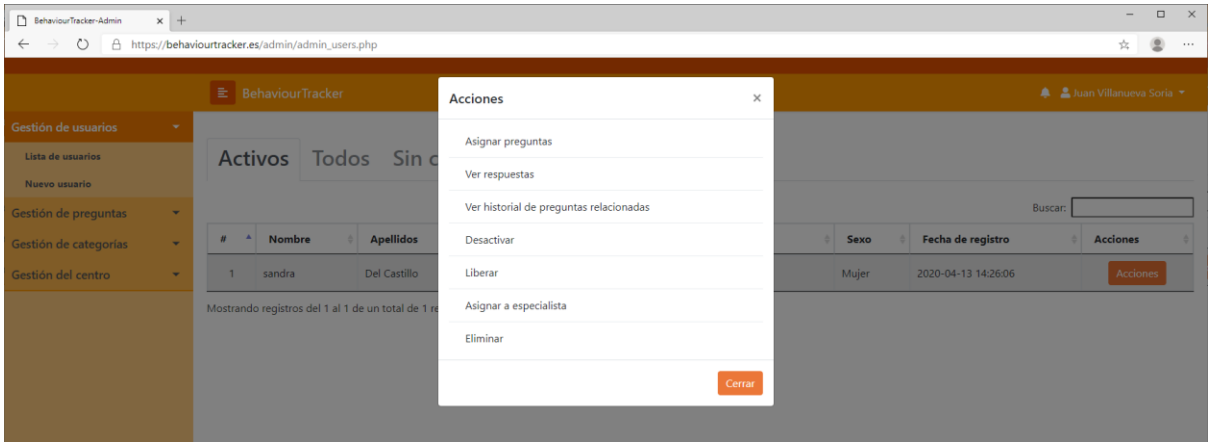


Figure 4.36: User actions.

This action panel has been implemented using *Bootstrap Modals*.

Modal Actions

```
<div class="modal fade" id="modal-actions" tabindex="-1" role="dialog" aria-
hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Acciones</h5>
                <button type="button" class="close" data-dismiss="modal" aria-
label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">

            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-
dismiss="modal">Cerrar</button>
            </div>
        </div>
    </div>
</div>
```

Show modal actions on click

```
<script type='text/javascript'>
function showModalActions(data,table){
  var modal = document.getElementById('modal-actions');
  var acciones = "";

  switch (table) {
    case "user-table":
    <?php if($_SESSION['rol'] == 0){ ?>
```

```
    acciones = '
     <div class=""> \
       <a class="dropdown-item" href="admin_users_related_questions.php?uuid=' +
data[0] + '">Asignar preguntas</a> \
     <div class="dropdown-divider"></div> \
       <a class="dropdown-item" href="admin_users_answers.php?uuid=' + data[0] +
'">Ver respuestas</a> \

                          ...

                        break;

                          ...

  }

  modal.children[0].children[0].children[1].innerHTML = acciones;
  $('#modal-actions').modal('show');
 }
 </script>
```

Code 4.16: Modal actions example.

When the actions button is pressed, the function *showModalActions()* is executed, so, depending on the table in which it has been pressed, the actions to be displayed are created, then they are inserted in the body of the modal and finally the modal is displayed using *$('#modal-actions').modal('show')*.

All Modals have been done in the same way in the rest of the website.

The actions available depending on the user's role in the site and the tab they are. The code above shows the actions an administrator can perform on an active user.

The following table shows the actions that can be performed by each site user depending on which tab they are on.

User	**Tab**	**Actions**
Administrator		
	Actives	• Assigning questions • See answers • View history of related questions • Disable • Release • Assign a specialist • Delete
	All	• Assigning questions

		● See answers ● View history of related questions ● Activate ● Release ● Assign a specialist ● Delete
	Unconfirmed	● Confirm ● Delete
Specialist		
	Actives	● Assigning questions ● See answers ● View history of related questions ● Disable ● Release
	Disabled	● Assigning questions ● See answers ● View history of related questions ● Activate ● Release
	Without specialist	● Activate
Secretary		
	Without specialist	● Assign a specialist
	Unconfirmed	● Confirm

Table 4.3: Users centre actions according to the tab.

admin_users_create.php

This script can be accessed from the sidebar, user management and new user.

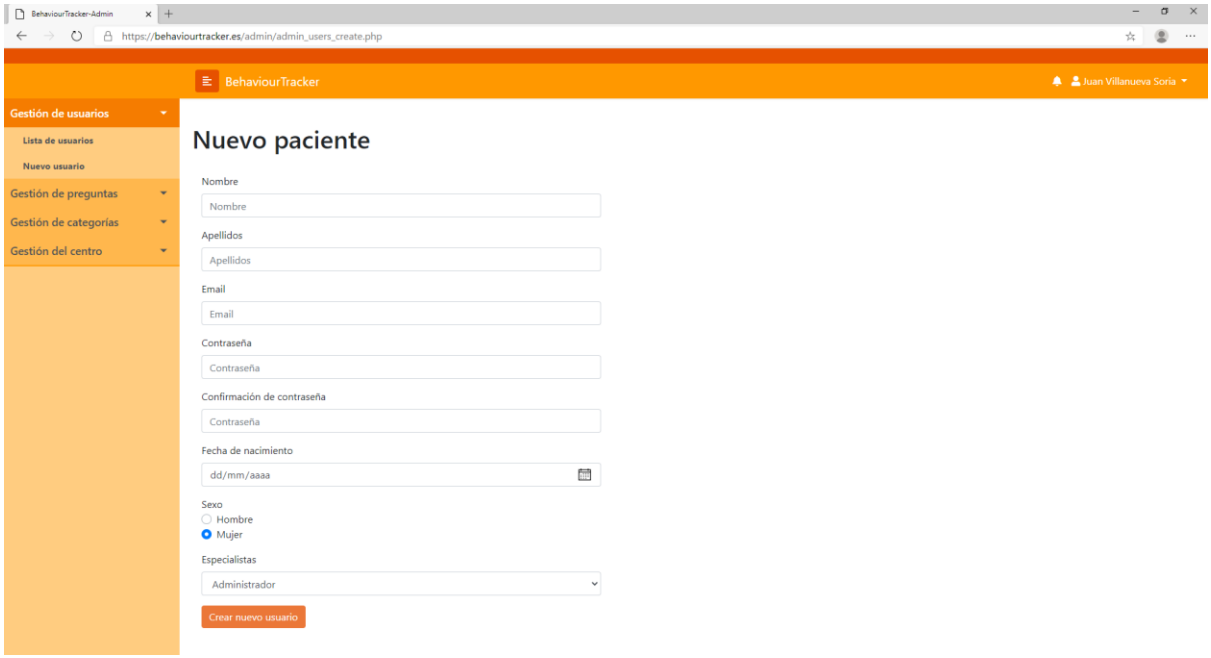


Figure 4.37: New user form.

This script takes care of creating a new user of the application. As well as filling in the user's details, we have the option of directly choosing the specialist who is going to follow that user.

When creating the users from the website, they don't have to be confirmed as they are registering from the centre.

admin_users_activate.php

This is one of the actions that can be performed on users within the action menu.

This script is in charge of activating the user.

admin_users_disable.php

This is one of the actions that can be performed on users within the action menu.

This script is in charge of disabling the user.

admin_users_release.php

This is one of the actions that can be performed on users within the action menu.

This script is in charge of releasing the user, that is, if the user is a specialist, it releases him, and he becomes an inactive user of the administrator.

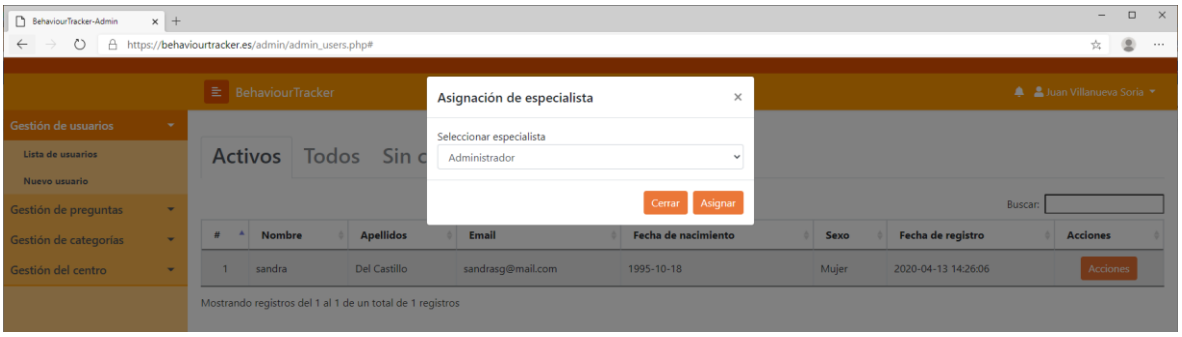
admin_users_assign_to_specialist.php

This is one of the actions that can be performed on users within the action menu. The action of assigning a specialist and confirming use this same script.

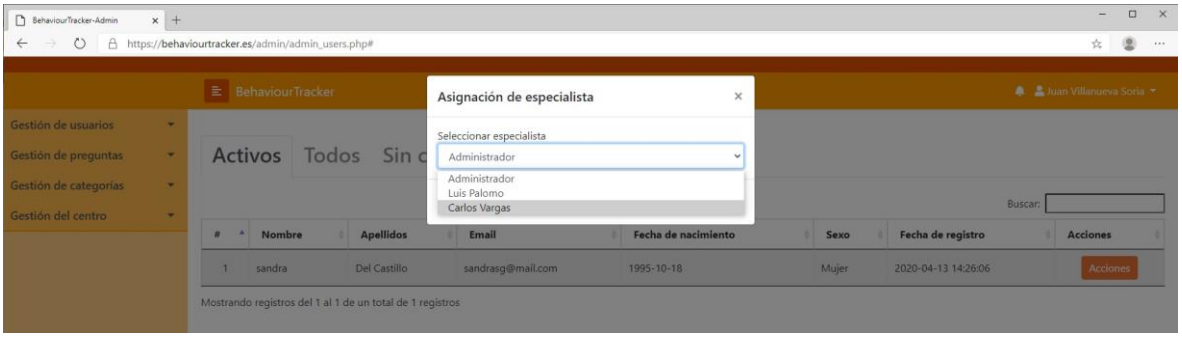
This script is in charge of associating a user with a specialist. Before running the script, the centre user must first select the specialist to be associated with it. The same functionality of the Bootstrap mode is used. In this case, the specialists and the administrator are inserted in the body of the modal. Once the specialist to be associated with the user has been selected, and the assign button is pressed, the specialist assignment script is executed.

The following figure is an example of how this is done.

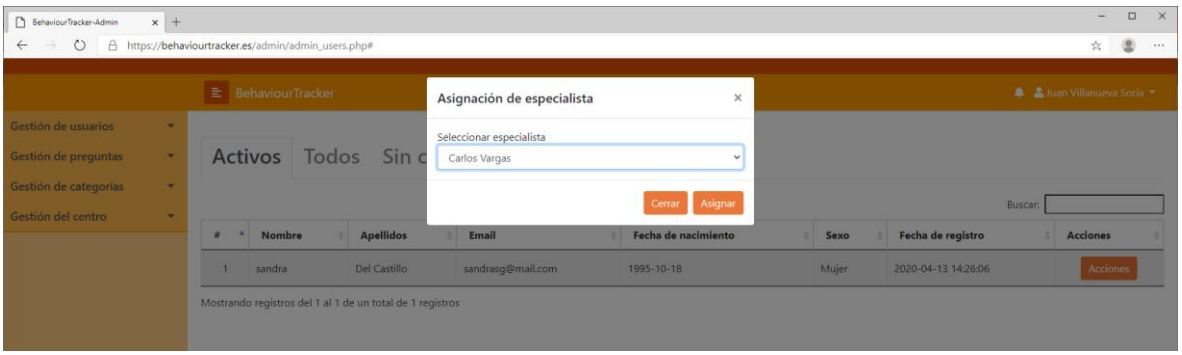
Select Assign a specialist from the actions menu.



Select the new specialist to be assigned



Press the button Assign



Now the user Sandra Del Castillo is disabled and assigned to the specialist Carlos

Vargas.

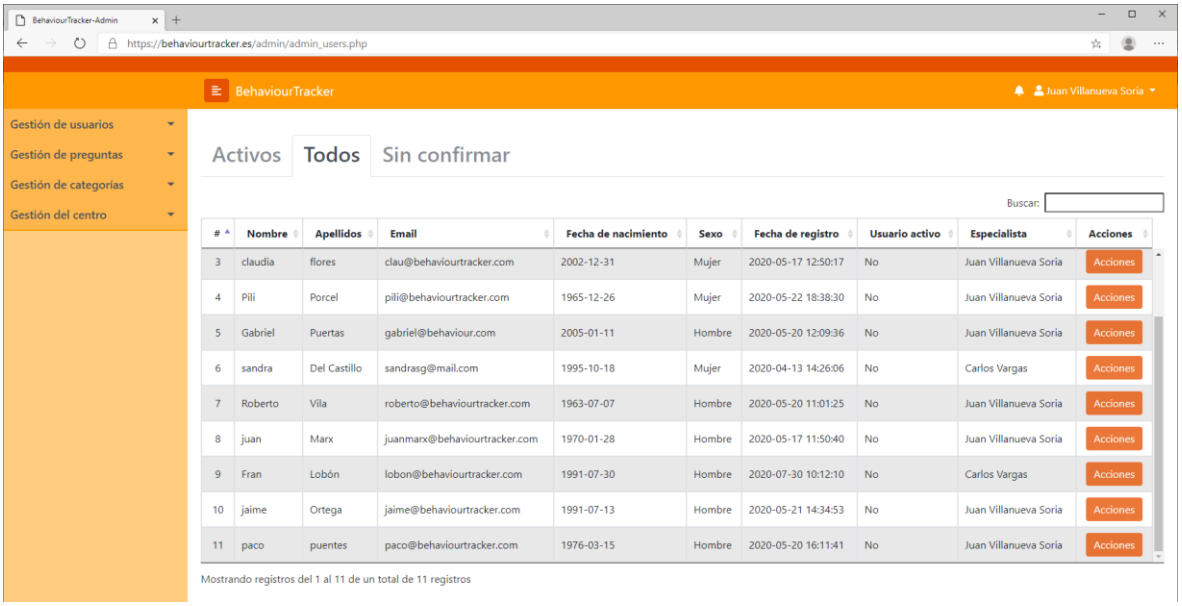


Figure 4.38: How to assign one user to a specialist.

admin_users_delete.php

This is one of the actions that can be performed on users within the action menu.

This script is in charge of deleting a user, although it is not deleted from the database but simply makes it not appear in this centre anymore. Then the support would proceed to its removal.

Before being deleted, it asks by a Bootstrap mode if it wants to be deleted, so that if by mistake you press on the action of deletion, that is not done until it is confirmed.

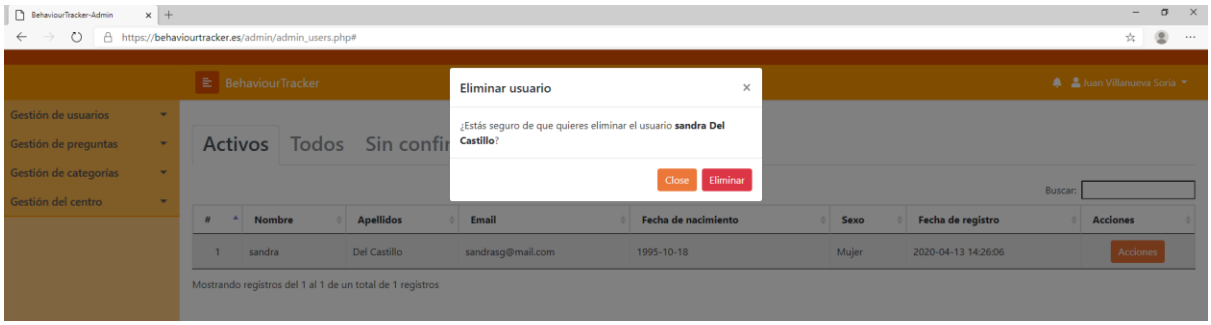


Figure 4.39: Delete Modal view.

admin_users_related_questions.php

This is one of the actions that can be performed on users within the action menu.

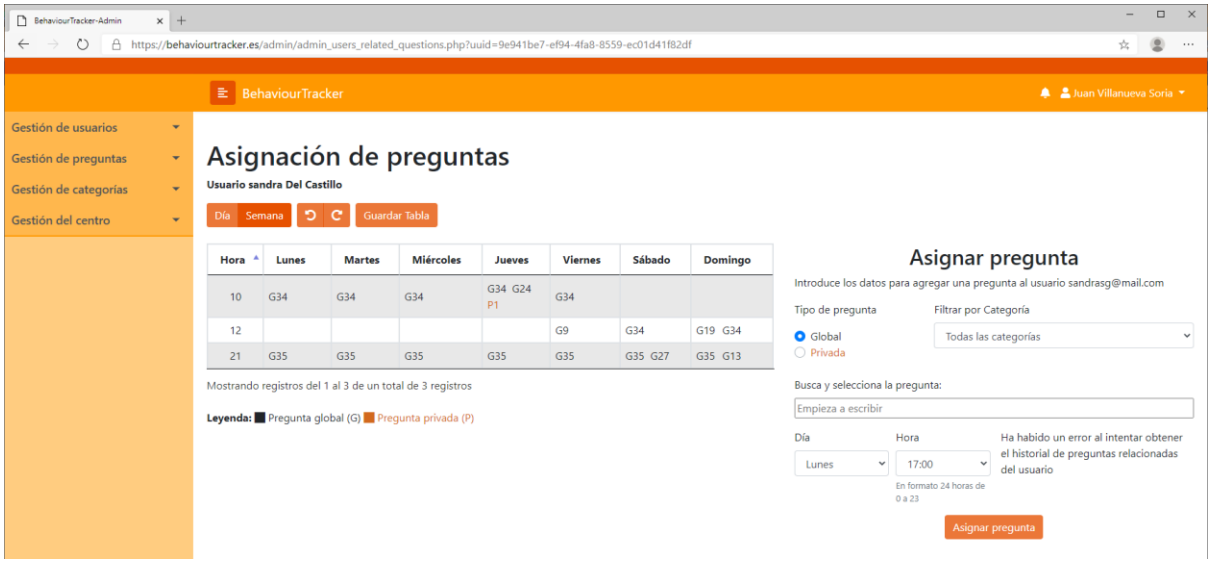


Figure 4.40: Question assignment.

This script assigns the questions to a user using a table that shows the days of the week. The questions are assigned by an hour, and there can be several at the same time, but they have to be different.

The table shows the question identifier that when the question is global is shown as G + the identifier, and when it is private is shown as P + the identifier and also changed colour as seen in the figure above. If somebody want to know which questions are, just pass the mouse pointer over each one of them, and a box will appear with the question information as shown in the figure below.

Figure 4.41: Show question on mouseover.

This picture disappears when we move the mouse to another point.

Another option we have is to show the table by day, for this we have a day and week selector above the table along with other options. If we select the day, the following table will appear.

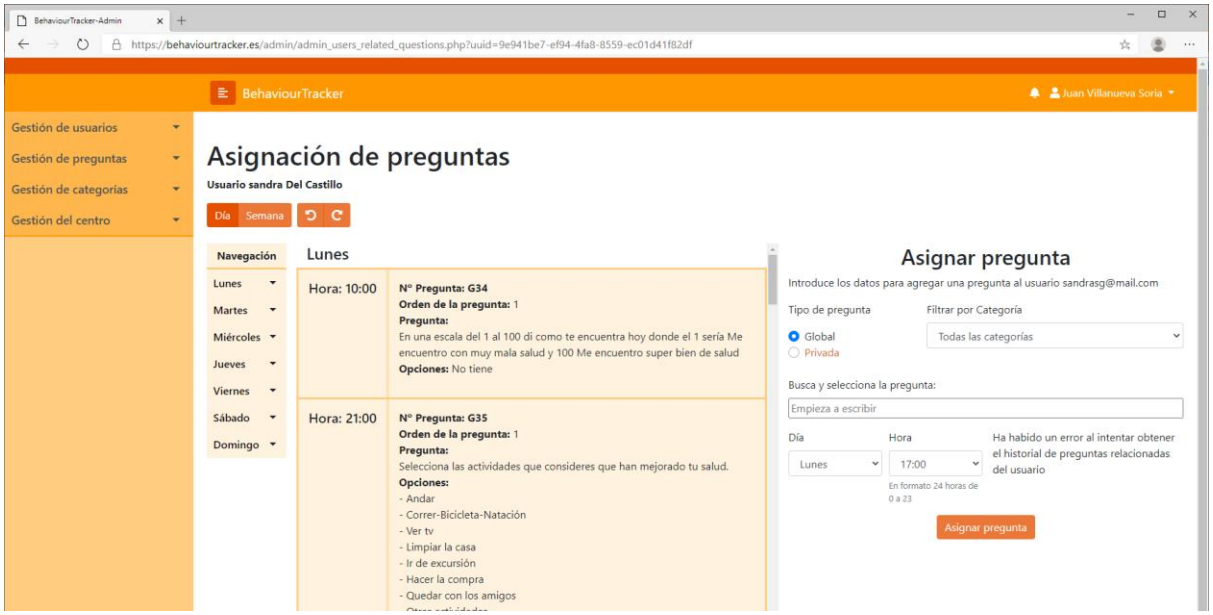


Figure 4.42: Day selector.

When you are in the day view, you can see that it shows a navigation bar on the one hand where you can navigate to the day and time you want in the table and on the other hand another table with the information of all the questions.

Figure 4.43: Day selector, navigation bar.

This table shows in more detail all the questions, and in what order they will be seen when the user of the mobile application asks them.

The day view is dynamically generated by a function when we press the selector.

In addition to this, the week view table is interactive. When we are in the weekly table (with the selector in week) you can delete questions with a single mouse click on the question, you can also drag and drop to change a question sequence as long as it is from the same day and at the same time, and you can delete entire rows by clicking on the time.

One identifier is used in each of the cells to make the table interactive when the table is created in *HTML*. The identifier is, on the one hand, the day-time for the cells containing questions, and on the other hand, the time for the cells of the time column. This allows you

to use the delete functions, assign a new question and change the sequence. In addition, when using the *DataTables* API, changes to the table must be saved using API functions so that the changes that have been made are effective.

On the other hand, if the changes that have been made do not convince us, they can be undone or re-done with the buttons at the top of the table.

Figure 4.44: Undo and redo button.

```
var toUndo = new Array();

function undo(){
    if(toUndo.length > 0){
        var data = toUndo.pop();
        switch (data[0]) {
            case "add-question":
               ...
            break;
            case "delete-question":
               ...
            break;
            case "delete-row":
                var timeQuestion = data[1];
                var row = data[2];
                addRow(row);
                toRedo.push(["delete-row", timeQuestion]);
            break;
            case "change-position":
               ...
            break;
        }
    }
} // function toUndo

var toRedo = new Array();

//
function redo(){
    if(toRedo.length > 0){
        var data = toRedo.pop();

        switch (data[0]) {
            case "add-question":
               ...
            break;
            case "delete-question":
```

```
                ...
            break;
            case "delete-row":
                var timeQuestion = data[1];
                deleteRow("undefined",timeQuestion);
            break;
            case "change-position":
                ...
            break;
        }
    }
}

function deleteRow(element, time){
    if(typeof time === 'undefined'){
        toUndo.push(["delete-row",
                     element.innerHTML,
                     related_question_table.row($(element)).data()]);
        related_question_table.row($(element)).remove().draw();
        toRedo = [];
    }else{
        var dayAndTime = "1-" + time;
        toUndo.push(["delete-row",
          time,
          related_question_table.row($("#" + dayAndTime).parents('tr')).data()]
        );
        related_question_table.row($("#" + dayAndTime).parents('tr'))
                              .remove().draw();
    }
}

function addRow(row){
  related_question_table.row.add(row).draw();
}
```

Code 4.17: Undo and Redo when a row is Deleted.

This code shows the undo and redo functions when a row is deleted.

When a row is deleted by clicking on a cell in the time column, the *deleteRow()* function is called, and only the element parameter is passed to it. First, the function inserts the delete-row action into the *toUndo* array along with the parameters it needs such as the time as the second parameter (element.innerHTML contains the time) and the entire row as the third parameter. Second, it removes the row using the API function, and finally the *toRedo* array is emptied because when you perform an action other than undo or redo you have to empty this array for it to work properly.

toUndo and *toRedo* work as a stack, so the *pop()* and *push()* functions are used.

If you want to undo the change, you just have to press the undo button. When pressing, the *undo()* function is executed, so first the last action performed is removed, and a new row is added with the *addRow()* function, this function only adds the row that is passed to it by parameter using the API. Finally, the action *delete-row* is added to the *toRedo* array with the time parameter to be able to redo that change.

If you want to redo it, press the redo button. When you press it, the redo() function is executed. First, the action is removed with the *pop()* function, which in this case is delete-row and then the *deleterow()* function is called but this time the time parameter is passed because the element is not saved. Therefore, when passing the time parameter, we form any *day-time* identifier, because as we are going to delete a row, we don't need to know the exact day, but the time. Then the action is saved in *toUndo* passing the time, and row parameters and finally the row is deleted using the API function. In this case, *toRedo* is not emptied as we have just re-done

The code above is an example of two of the functions used for the table. The others are longer and therefore, will not be included. Still, the basis of operation is the same, only taking into account more factors when removing, adding or rearranging the questions. The drag and drop will not be explained either, but the next section of the question management will explain and show your code of how it is done.

Continuing with the interface, when we want to add questions to the table, we have a question finder next to it.

Figure 4.45: Question finder.

First, you define what type of global or private question you want to search for. Then you can filter by category and select the day and time you want. This makes the search engine adapt with these selected filters and thus reduce the search of the questions. Then the search engine searches for the question. This search engine uses an auto-complete function that works so that when you start typing it shows the questions that match what you are typing. As shown in the figure below.

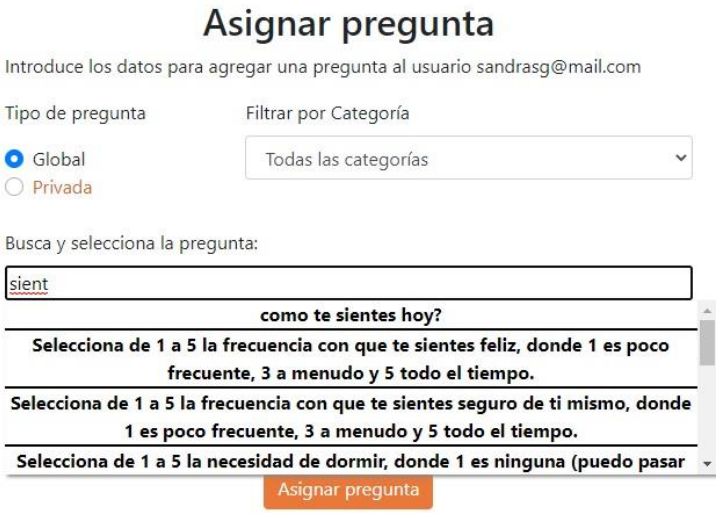


Figure 4.46: Finding a question.

```
jQuery(document).ready(function($) {
  var input_question = document.getElementById("inputQuestion");

  $( "#browser" ).autocomplete({
    source: all_global_questions[0],
    position: position,
    minLength: 0,
    select: function(event, ui) {
      if(ui.item){
        input_question.value = ui.item.content['id'] + "::" +
ui.item.content['secondary_id'] + "::" + ui.item.content['private'];
      }

           ...

    }
    }).data("ui-autocomplete")._renderItem = function (ul, item) {
      return $('<li></li>')
      .data("item.autocomplete", item)
      .append('<div style="text-align: center; font-weight: bold; color:
black;">' + item.label + '</div>')
      .appendTo(ul);
    };
});
```

Code 4.18: Autocomplete.

The code above shows the function that is executed to search the questions according to the letters and words that match. This function is passed to source in *JSON* format ({ label: "", content: ""},...). *label* is used to search for matches, and therefore the question is passed to it without the options, and then in *content* all the information of the question is passed to it (private, identifier, etc). The source changes depending on the filters used (private or global and the category). Once a question is selected it is added to an entry that is invisible with the

value of ***id::secondary_ir::private*** which will then be used when the add question function is executed by pressing the assign question button.

This way, you can quickly find the question you want to assign without having to search for it in a list that can be very big depending on the amount of questions that are generated.

Finally, when you want to save the questions in this table, you must press the save table button that appears above the table.

The table is saved through the **admin_users_related_questions_save_table.php** script.

Once it has been saved correctly, a warning will appear to say if it has been saved correctly.

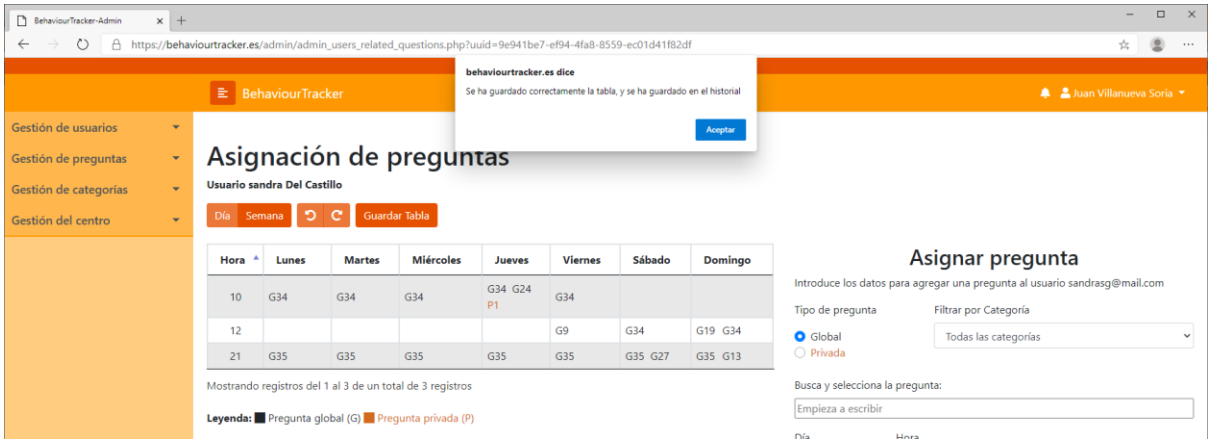


Figure 4.47: Save questions table.

When the table is saved, it automatically makes a copy in *JSON* format (discussed above in the database section) in the history which will be discussed below.

admin_users_related_questions_timeline.php

This is one of the actions that can be performed on users within the action menu.

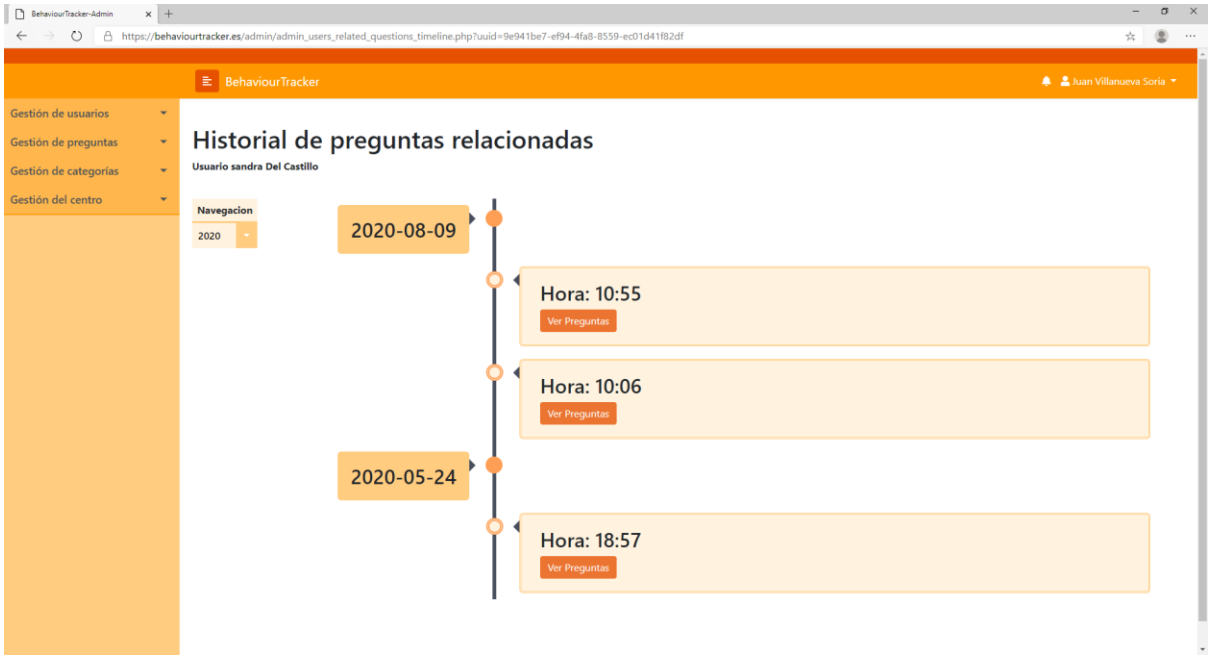


Figure 4.48: Related questions timeline.

This script is responsible for displaying a history of the tables that have been saved over time.

On the one hand, there is the navigation bar that we can select the year and month to take us to that point. On the other hand, there is the history where on the left appears the date where the changes were made, and on the right the hours at which they were made plus a button that takes us to the content of the tables. Each time you access to see the questions, the script admin_users_related_questions_content.php is called with an XMLHttpRequest [21] with which you can update the content of the page without having to load it. This last script is the one that generates the structure of the following figure dynamically.

Figure 4.49: Related questions timeline content.

The content is the same as in the question assignment table. Although you cannot perform any action on it, only the questions are shown by placing the mouse on the question. In addition, you can get a detailed view of the day.

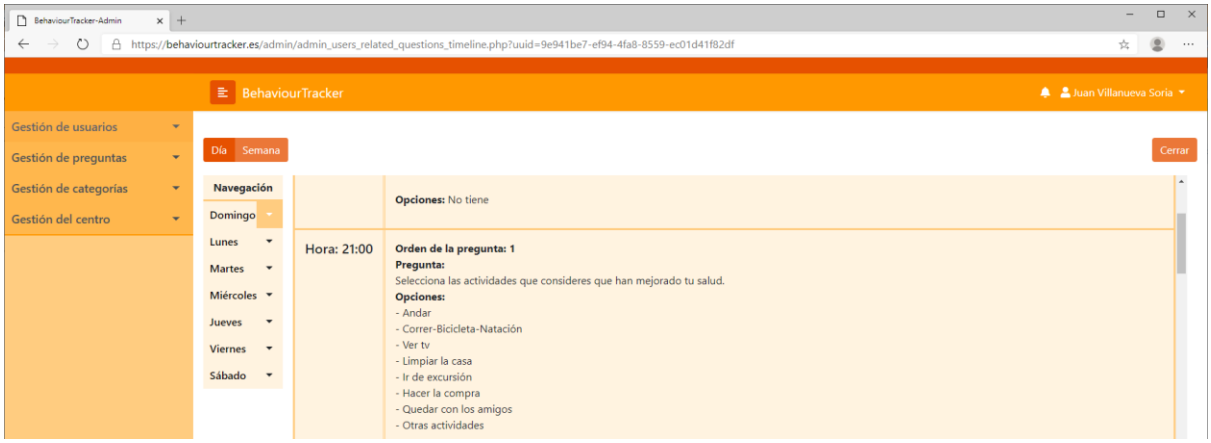


Figure 4.50: Related questions timeline content (day view).

The day view still has its functional navigation bar to go to the specific day and time.

admin_users_answer.php

This is one of the actions that can be performed on users within the action menu.

Figure 4.51: Answers timeline.

This script is in charge of showing a history as the previous one but of the answers that the user of the mobile application has been doing.

admin_users_notification.php

This script has already been discussed, it is used for notifications of unconfirmed users and returns the email, first and last names of unconfirmed users and the number of unconfirmed users.

4.3.5 Question management

Questions can only be handled by a specialist or an administrator. The secretary does not have access to this part of the web application.

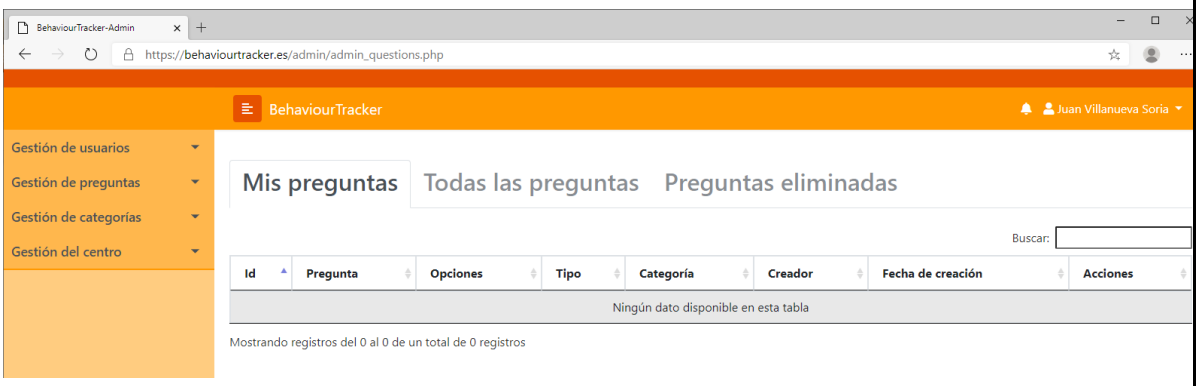
The questions have been designed so that they can be private or global to the centre. This means that a specialist or administrator can have their questions so that they can only see and use them themselves. On the other hand, if a specialist makes your question global, then you will not be able to put the question private again as another specialist might use it. Each specialist can only modify their question if it is private, and an administrator can modify any question from any specialist. Besides this, the questions use two IDs. One is the *ID* where it is stored in the database and another id (*secondary_id*) that is used to have an id of the questions depending on if they are private or global to facilitate their search.

Now we are going to describe each of the scripts that make up the question management.

admin_questions.php

This script can be accessed from the sidebar in question management and question list.

Administrator view



Specialist view

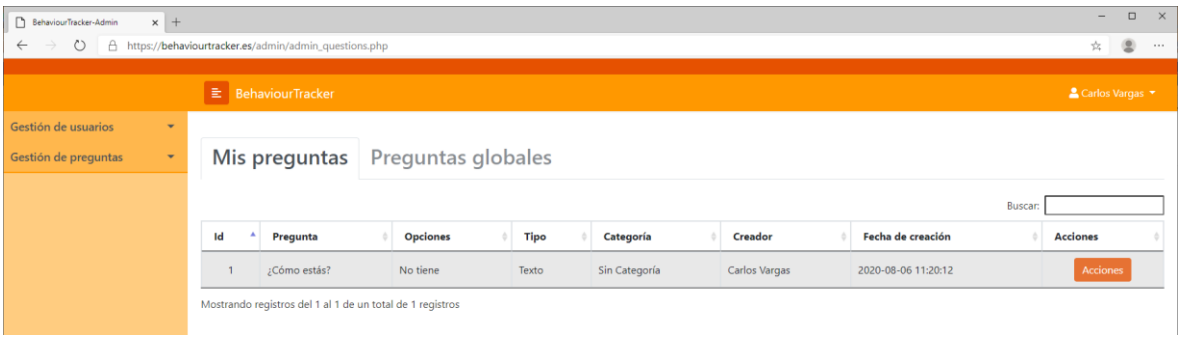


Figure 4.52: Questions list.

This script is the main view of the questions. You can see that there is a panel, when you access the list of questions from the sidebar the view shown in the figure above appears. We can see that there are a series of tabs that in the case of the administrator have three.

The first one is the one that appears in the figure above, and they are the private questions of each one.

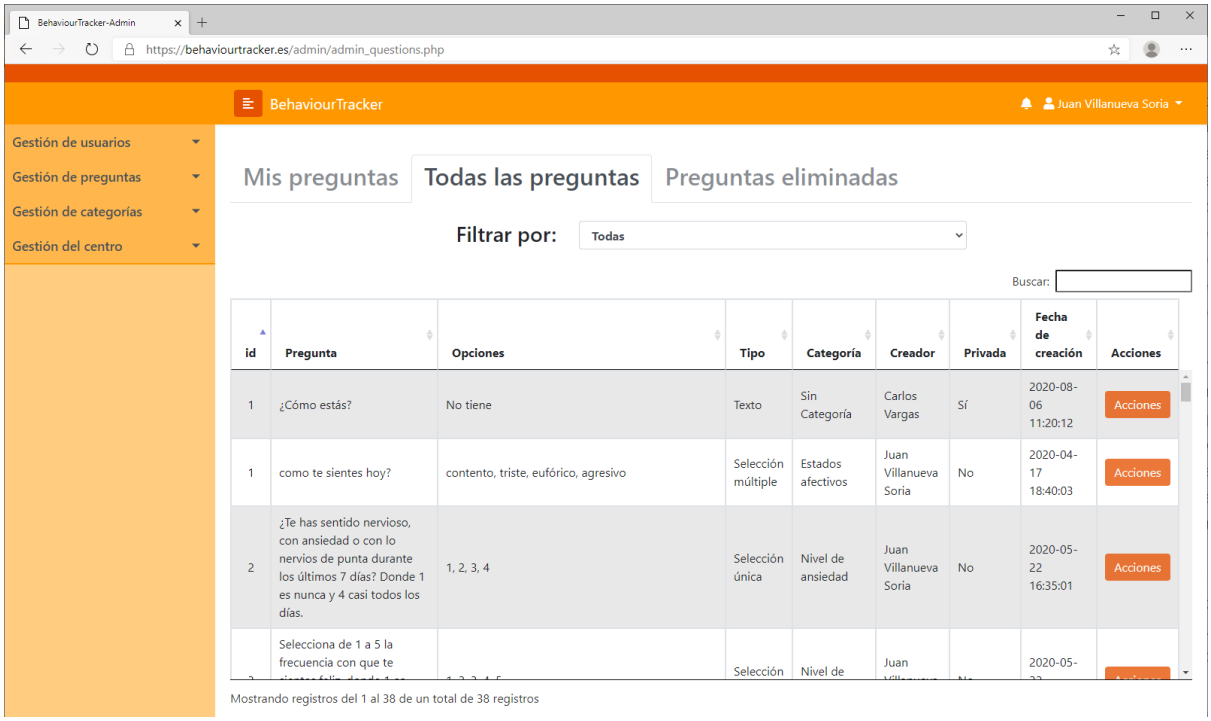


Figure 4.53: Global questions.

In the case of the administrator in the next tab, all the questions appear both private and global. If you were a specialist, only the global questions will appear. Unlike a specialist, the administrator can filter the questions so that either all the private and global questions appear or filter the questions by a specialist. A specialist only sees global questions on the global questions tab.

Figure 4.54: Deleted questions.

The last administrator tab shows the questions that have been deleted, but they do not disappear from the database because if they have been used previously, they would not be seen in the users' history of related questions.

Each table shows the fields that are needed, for example, in the tab “Mis preguntas” being private the private field is not shown.
In addition to this, each question has a field Actions that if pressed, shows the options you can make on that question as shown in the figure below.

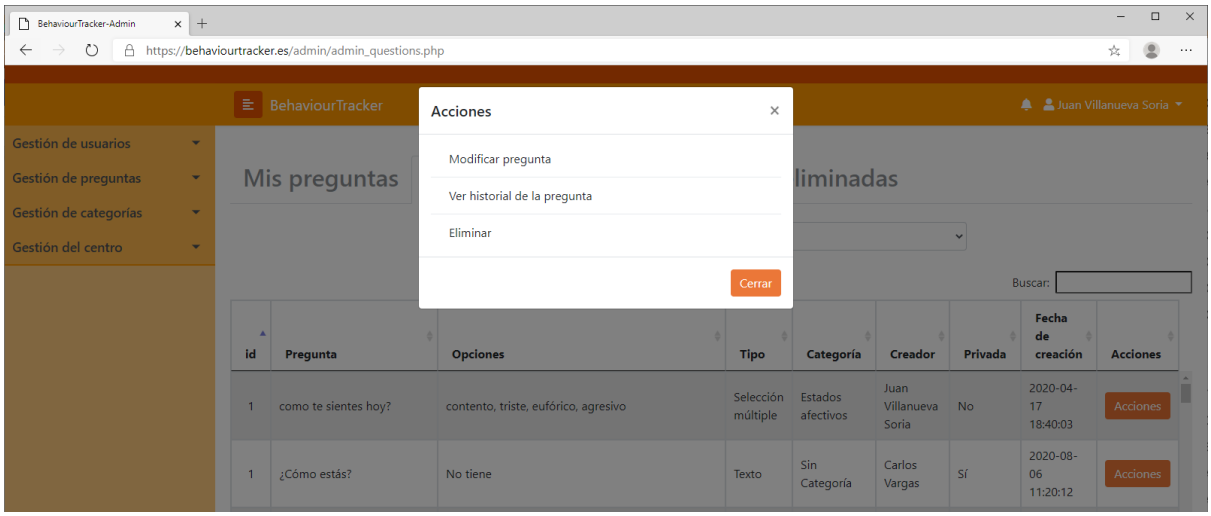


Figure 4.55: Question actions.

admin_questions_create.php

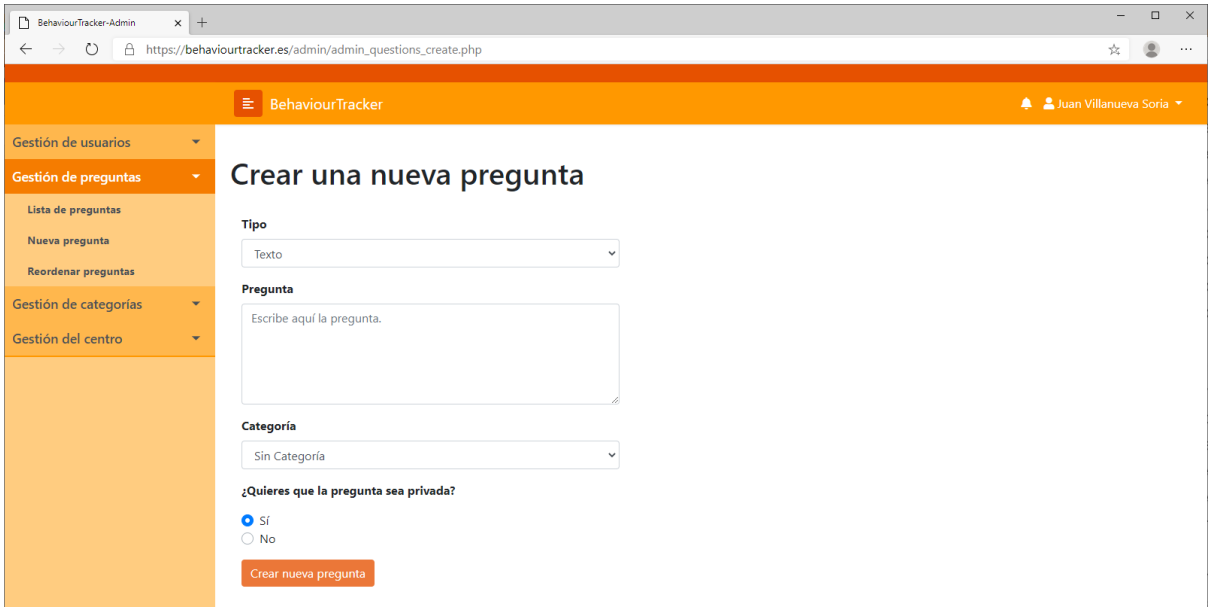


Figure 4.56: Create new question.

To access the creation of new questions, select New Question from the sidebar in Question Management.

In this script, new questions are created. You can select the type of question (which are text, single selection and multiple selection as explained in **subsection 4.2.6** in the mobile application's quiz screen), write the question, select the category to which the question will be associated, and select whether the question will be private or global which by default will always be private.

If the text type is changed to any of the selected types and therefore has options, it is shown as in the following figure.

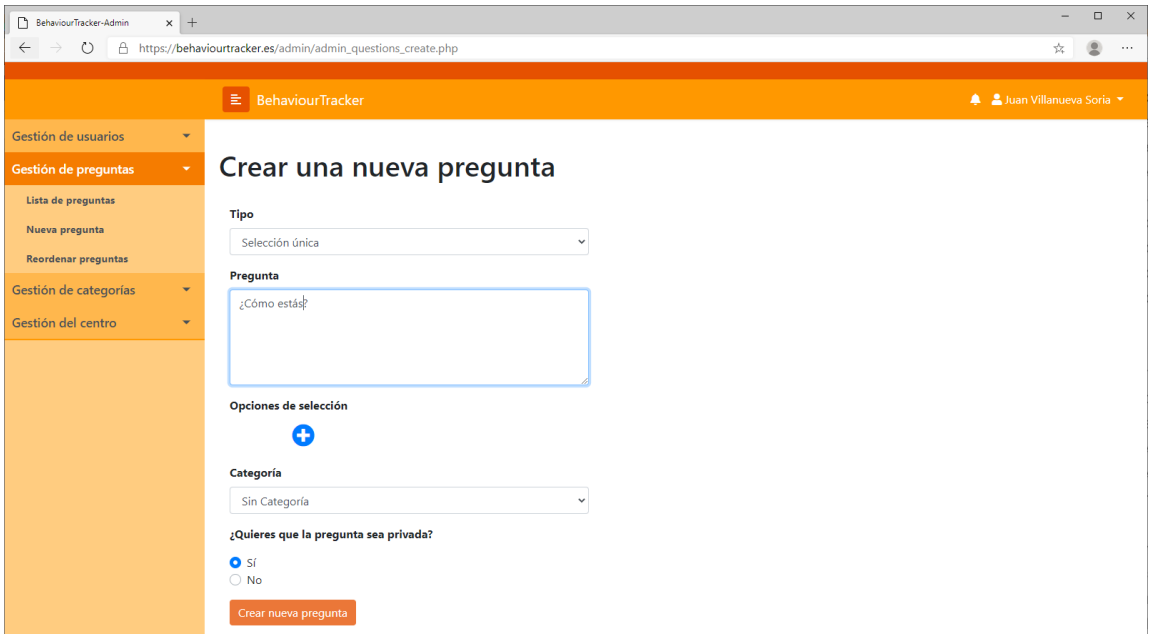


Figure 4.57: Changing question type.

In this case, the selection options appear, which you can create as many as you want. To create one, you should only add it by pressing the + button that appears in blue below.

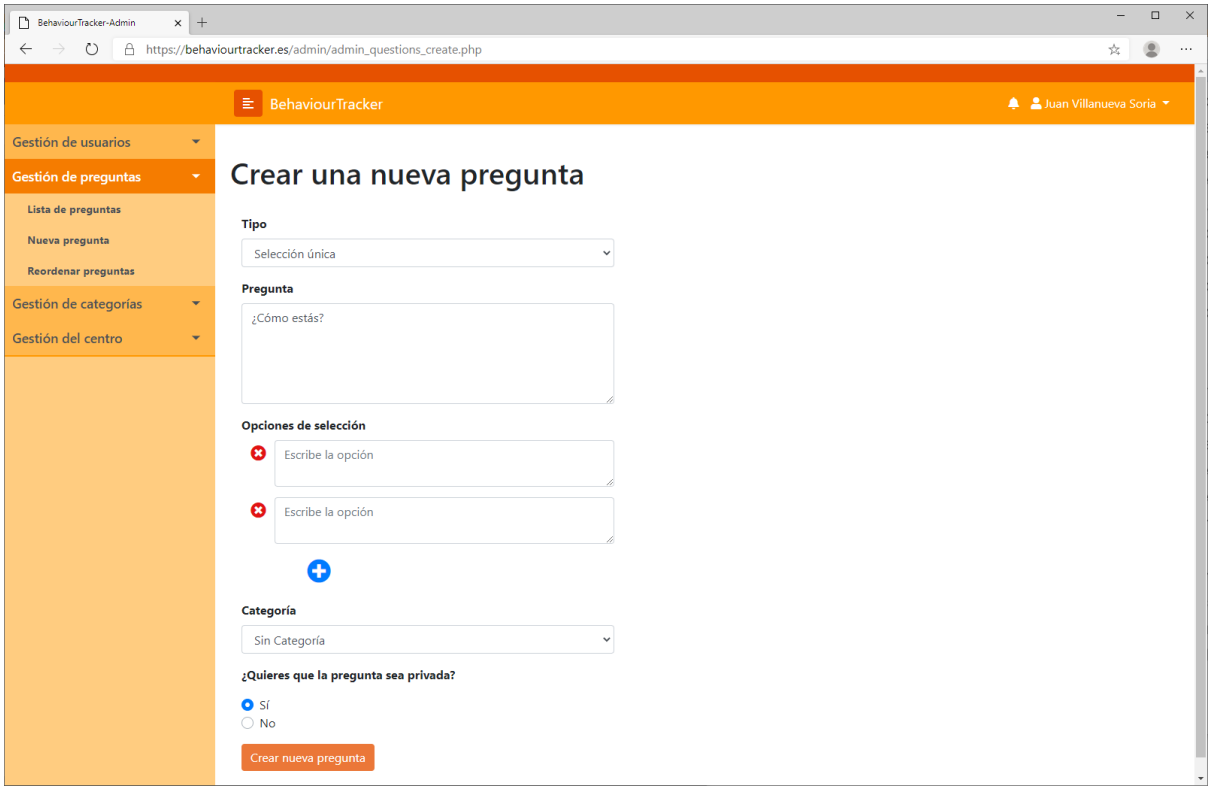


Figure 4.58: Creating question options.

They can be removed by clicking on the x button to the left of each option.

Another feature of option creation is the drag-and-drop option that allows you to alter the order of the options without having to delete and retype them. This action cannot be displayed in a figure, so it is shown in the following Javascript code used.

```
      function allowDrop(ev) {
        ev.preventDefault();
      }

      function drag(ev) {
        ev.dataTransfer.setData("text", ev.target.parentNode.parentNode.id);
      }

      function drop(ev) {
        ev.preventDefault();
        var data = ev.dataTransfer.getData("text");
        var valueToChange =
document.getElementById(data).children[1].children[0].value;
        console.log(valueToChange);
        var dinamicDiv = document.getElementById('dinamicDivOptions');
        var posupdate = -1, lastpos = -1;
        var newval = -1, lastval = -1;

        for(var i=0; i < dinamicDiv.childElementCount; i++){
          if(dinamicDiv.children[i].id == data){
```

```
            lastpos = i;
          }
          if(dinamicDiv.children[i].id == ev.target.parentNode.parentNode.id){
            posupdate = i;
          }
        }

        if(lastpos > posupdate){
          newval = dinamicDiv.children[posupdate].children[1].children[0].value;
          for(var i = posupdate+1; i <= lastpos; i++){
            lastval = dinamicDiv.children[i].children[1].children[0].value;
            dinamicDiv.children[i].children[1].children[0].value = newval;
            newval = lastval;
          }
          ev.target.value = valueToChange;
        }else if(lastpos < posupdate){
          newval = dinamicDiv.children[posupdate].children[1].children[0].value;
          for(var i = posupdate-1; i >= lastpos; i--){
            lastval = dinamicDiv.children[i].children[1].children[0].value;
            dinamicDiv.children[i].children[1].children[0].value = newval;
            newval = lastval;
          }
          ev.target.value = valueToChange;
        }
      }
```

Code 4.19: Drag and drop.

This code above allows you to change the order of the options. Basically what it does is that first you grab an element with the *drag()* function when you release the grabbed element the *drop()* function is executed which simply moves the other options one space up or down depending on where you released the element to be changed. To use the drag and drop functionality, you have to enable it in the *HTML* code of the elements that we want to have this functionality.

```
<textarea ondrop="drop(event)"
          ondragover="allowDrop(event)"
          draggable="true"
          ondragstart="drag(event)"
             ...
</textarea>
```

Code 4.20: Enable drag and drop.

The drag and drop code used for the questions is also used for when you want to change a question order in the user's question table but changing the behaviour when it is dropped and when it is grabbed.

admin_questions_update.php

The specialist can modify his private questions while the administrator can modify all questions. The modify action is one of the actions that can be performed when clicking on a question in the question list.

This script does exactly the same as the question creation script but modifies an already created question, i.e. it changes the SQL set, and instead of INSERT it uses UPDATE.

admin_questions_delete.php

Only the administrator can use this delete function. This can be done by clicking on a question in the question list.

This script takes care of removing the question from the question list, but not from the database as mentioned above.

admin_questions_recover.php

Only the administrator can use this function. This can be done by clicking on a question in the list of deleted questions.

This script is used to retrieve a deleted question.

admin_questions_reorder.php

Only the administrator can use this function. This action is located in the sidebar in the question management.

This script is used to reorder all the questions since removing questions creates gaps in the secondary id and with this action you can reorder to remove those gaps.

admin_questions_timeline.php

This script can be used by any specialist for both private and global questions. The administrator can use it for all questions. This can be done by clicking on a question in the question list.

This script is a question history that displays the changes to the selected question as a timeline. Also, if the question has been changed in different years or months, it will appear in the navigation bar, which can be used to go to a specific month and year in the question history.

Figure 4.59: Question timeline.

In addition, an open-source library [22] has been used to highlight the changes (in green and red) that have been made to the question text and question options. Not in categories, as it is easier to compare.

```
$from_text = mb_convert_encoding($old_question, 'HTML-ENTITIES', 'UTF-8');
$to_text = mb_convert_encoding($new_question, 'HTML-ENTITIES', 'UTF-8');
$diff_opcodes = FineDiff::getDiffOpcodes($from_text, $to_text,
FineDiff::$wordGranularity);

$rendered_diff_question = FineDiff::renderDiffToHTMLFromOpcodes($from_text,
$diff_opcodes);
$rendered_diff_question = htmlspecialchars_decode($rendered_diff_question,
ENT_NOQUOTES);
```

Code 4.21: FineDiff example.

4.3.6 Categories management

Like the questions, the categories can only be managed by the centre administrator who is responsible for creating, modifying or deleting them.

There is a category that will be the initial category of the centre which is called *initial_category*. This category does not appear in the list of categories and can not be modified or deleted and serves to have a category always available in the centre because, without it, the application could not work.

A category can be hidden. This means that it is not deleted, but it appears hidden as well as all the global questions that are associated with it, except the private questions of each

specialist that would change their category to no category. This is another utility of the categories to hide several questions.

admin_categories.php

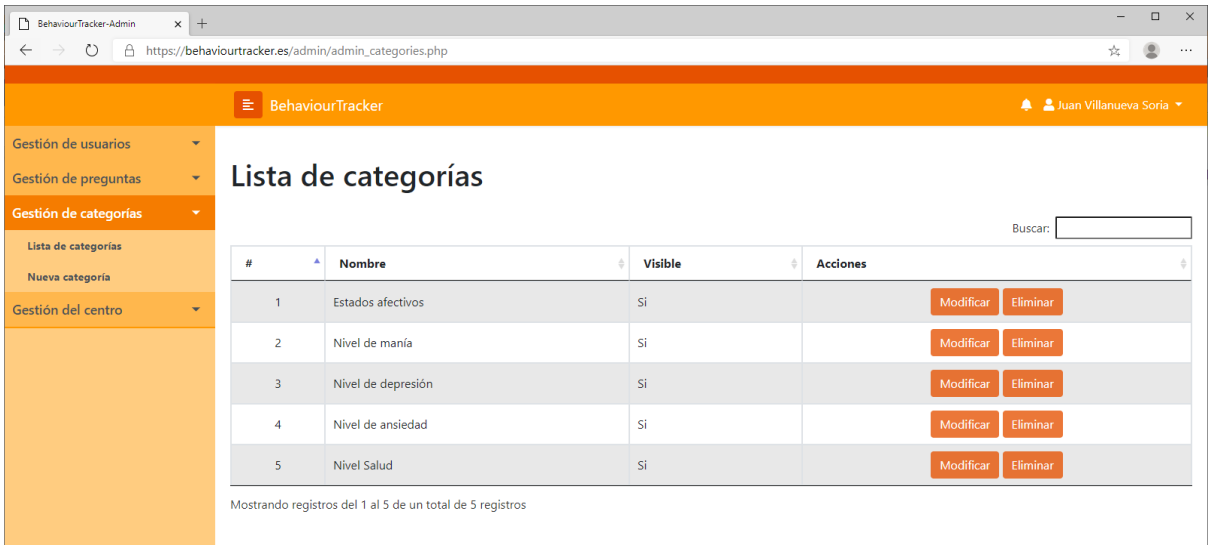


Figure 4.60: Category list.

You can access this list from the sidebar in Category Management and Category List.

This script shows a table with the categories that are currently available as well as the actions you can perform on them.

admin_categories_create.php

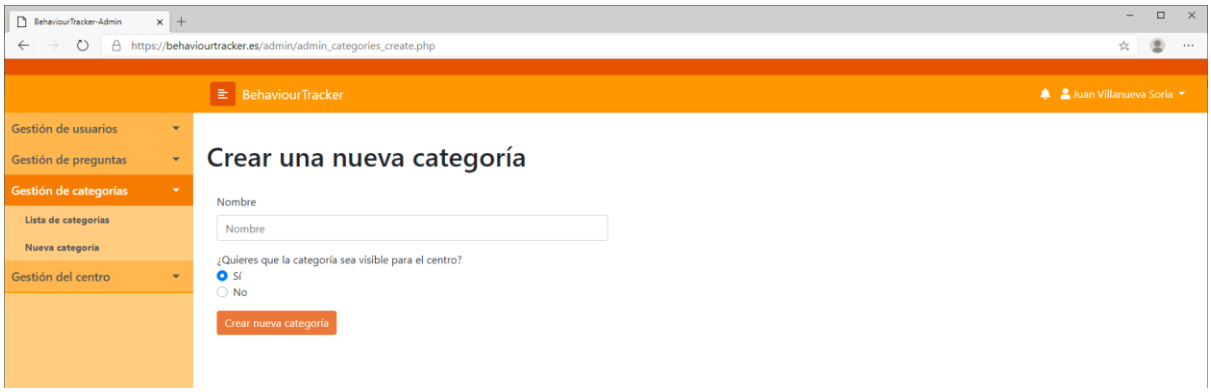


Figure 4.61: Create new category.

To create a category, you can access from the sidebar in category management and new category.

This script creates a new category by entering its name and whether or not you want it to be visible, by default the category is visible.

admin_categories_update.php

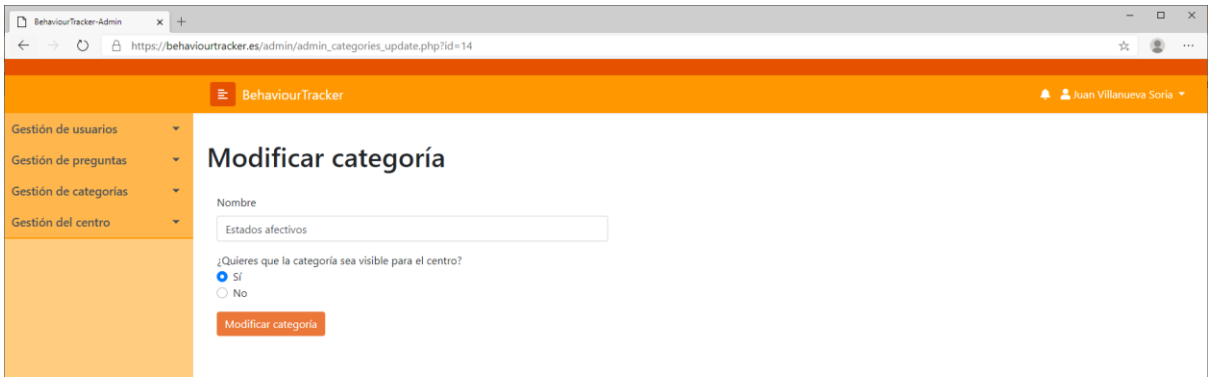


Figure 4.62: Update category.

Modifying a category is part of the actions that can be performed from the category list.

This script is exactly the same as the previous one with the difference in the SQL statement that is executed.

admin_categories_delete.php

This script is part of the actions that can be performed from the category list.

It serves to remove a category as long as that category is not in use for some question, if so it can not be removed.

4.3.7 Center users management

Like the questions and categories, the users of the site can only be managed by the site administrator.

Managing site users includes creating site users, modifying, deleting, and activating and inactivating site users.

admin_center_users.php

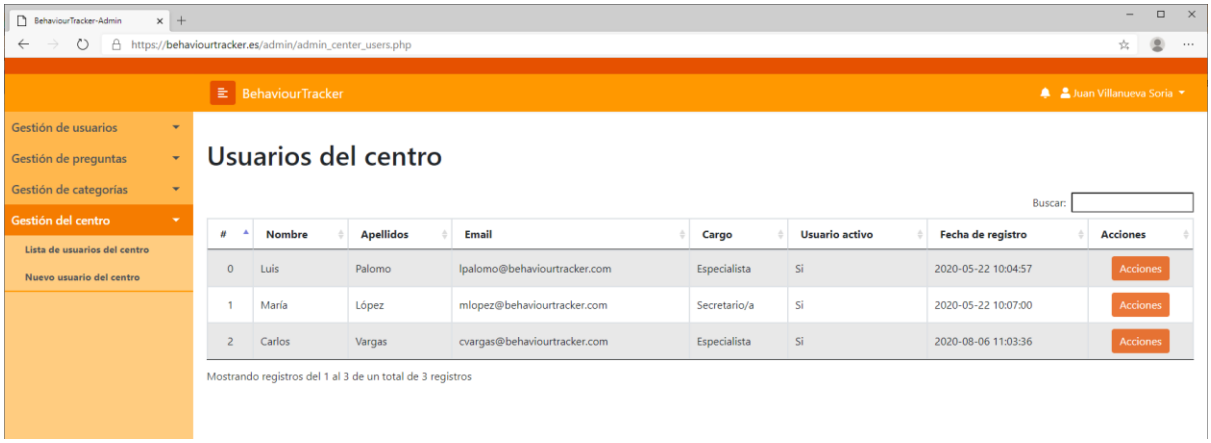


Figure 4.63: Center users list.

You can access it from the sidebar in site management and site user list. This script displays all the users of the centre in a list.

The actions that can be performed on each user are shown in the following figure.

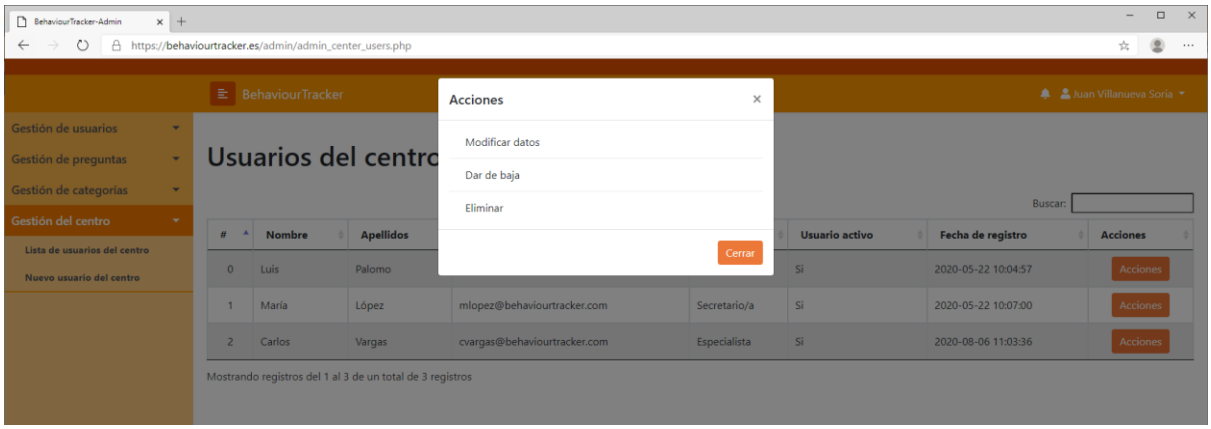


Figure 4.64: Center users actions.

You can change a user of the centre, disable a user, or activate in the case of a disabled user, and delete a specialist.

admin_center_users_create.php

You can access from the sidebar in plant management and new plant user.

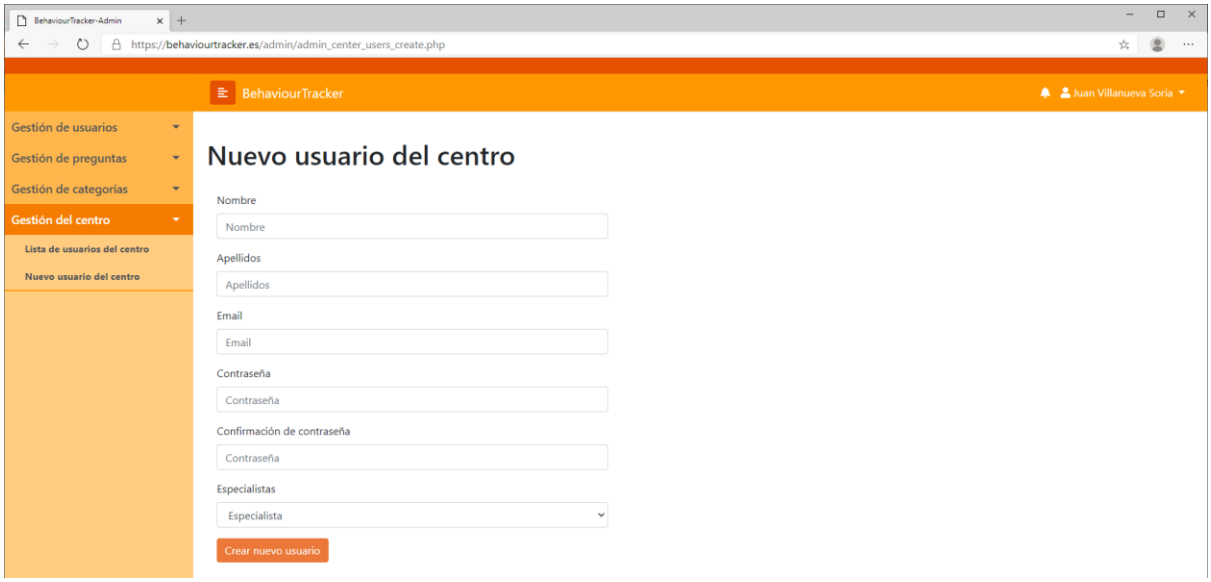


Figure 4.65: Create new centre user.

This script is used for the creation of new users of the centre, both specialists and secretaries.

admin_center_users_update.php

It is accessed from the plant's user list when you select a user's actions.

Same as the creation script but it is used to modify the data of a user of the plant.

admin_center_users_delete.php

It is accessed from the plant's user list when you select a user's actions.

This script handles the removal of users from the site. Before the user is deleted, the following is done:

- Update the users that you had active to inactive and assign them to the administrator of the centre.
- Assign all the questions you had to the site administrator.
- Assign all question history questions to the administrator.

Once all of the above is done, the site user is removed from the database.

admin_center_users_activate.php

It is accessed from the plant's user list when you select a user's actions.

This script is in charge of registering or activating a site user.

admin_center_users_disable.php

It is accessed from the plant's user list when you select a user's actions.

This script takes care of unsubscribing or deactivating a user from the site. Before deactivating the site user, users that have associates become inactive and are assigned to the site administrator.

4.3.8 Mobile application request management and password recovery

This section will briefly explain the REST API system designed for requests from the mobile application to the web and what data it returns. The password recovery system will also be explained. Since the website connection has the SSL security certificate, there is no problem sending the information in plain text without encryption since the connection already does.

Get data

In this section, there are three scripts that are:

- *gethealthcenters.php*: It returns in *JSON* format the centres that are available on the web.
- *getquestion.php*: It returns the questions associated to that user for that day.
- *getuserviewdata.php*: This script is in charge of providing the information of the User Home Activity screen. The information that is provided is: who is the specialist that follows that user, the centre in which he is and if he is active or not.

Update user data

The script *updateuserdata.php* is in charge of updating the server data when the user modifies it from the mobile application. This script is passed the email, password and type parameters, apart from the parameters needed to change the data that has been requested to be changed. With the email and the password, it is confirmed that the user exists and the type parameter is used to know which data must be updated.

```
if($can_update){
  switch ($_POST["type"]) {
      case "name":
         $code = "{'code' : '1'}";

         $query = "UPDATE users SET name = ? where email = ? ";
         if(mysqli_stmt_prepare($stmt,$query)){
            mysqli_stmt_bind_param($stmt, "ss", $_POST["name"], $_POST["email"]);
            if(mysqli_stmt_execute($stmt)){
               $code = "{'code' : '0'}";
```

```
            }
        }
        echo $code;
        break;
          ...
```

Code 4.22: Name update example.

Store answer

```
<?php
    if(isset($_POST["uuid"]) && !is_null($_POST["uuid"]) && $_POST["uuid"] !=
""){
        require "connect_db/connect_DB_USERS.php";
        require "connect_db/connect_DB_DATA.php";
        require "datastoretype.php";
        $stmt = mysqli_stmt_init($DB_USERS);
        $query = "SELECT uuid from users where uuid = ? ;";

        if(mysqli_stmt_prepare($stmt,$query)){
            mysqli_stmt_bind_param($stmt, "s", $_POST["uuid"]);
            mysqli_stmt_execute($stmt);
            if(mysqli_stmt_fetch($stmt)){
                mysqli_stmt_close($stmt);
                $data_store = new datastoretype($_POST["type"], $DB_DATA);

                if(($result = $data_store->insertData($_POST)) != ""){
                    echo $result;
                }else{
                    echo "{'code' : '2'}";
                }
            }else{
                echo "{'code' : '1'}";
            }
        }else{
            echo "{'code' : '3'}";
        }

        mysqli_close($DB_USERS);
        mysqli_close($DB_DATA);
    }
?>
```

Code 4.23: storeanswer.php

The script *storeanswer.php* is used to save an answers. This script uses the *UUID* of the user who, in case it exists, inserts the answer passed by POST parameter through another script called *datastoretype.php*.

datastoretype.php is the only script that is programmed as a class since it is easier this way since it is used to insert all the data from the web database.

```
class DataStoreType {
    private $data_type = '';
    protected $db;

    function __construct($type, $db) {
        $this->data_type = $type;
        $this->db = $db;
    }
    function insertanswer($uuid,$id,$answer,...){
      $query = 'INSERT into `answers` (`uuid`, `id`, `answer`, `content`,
`time_of_day`, `registration_date`) values (?,?,?,?,?,?)';
      $stmt = mysqli_stmt_init($this->db);
      if(mysqli_stmt_prepare($stmt,$query)){
        mysqli_stmt_bind_param($stmt, "sissis",
$uuid,$id,$answer,$content,$time_of_day,$resgistration_date);
        if(mysqli_stmt_execute($stmt)){
          mysqli_stmt_close($stmt);
          return TRUE;
        }
      }
      return FALSE;
    }
function insertData($values) {
      $result = "";
      switch ($this->data_type) {
          case 'answer':
     if($this-
>insertanswer($values['uuid'],$values['id'],$values['answer'],...)){
            $result = "{'code' : '0', 'id' : ".$values['id'].",
'registration_date' : '".$values['registrationDate']."'}";
          }
        break;
      return $result;
    }
}
```

Code 4.24: datastoretype.php, answer insert example.

Login and register

login.php checks that the email and password entered are correct and returns the corresponding error code.

register.php checks that the email is not already used and uses the class *datastoretype* to insert the new user, in case of error it returns the corresponding code.

Password recovery

A user can retrieve their password from the application on the login screen by using a password retrieval button. That button takes you to this page.

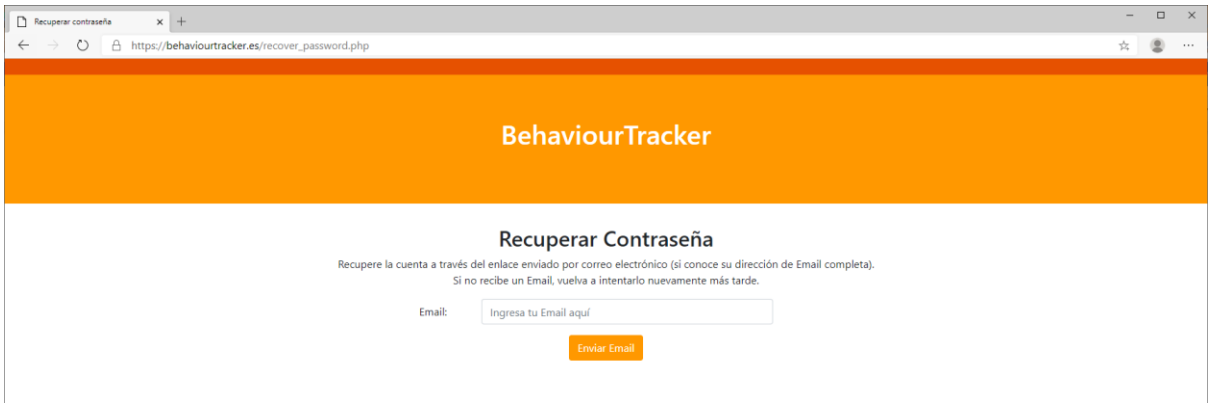


Figure 4.66: Recover password.

Once here the user enters his email, and if the email is registered, then an email is sent with a special link to retrieve the password.

```
if(isset($_POST['email']) && !is_null($_POST['email'])){
  require "connect_db/connect_DB_USERS.php";
  $stmt = mysqli_stmt_init($DB_USERS);
  $query = "SELECT email,uuid,name from users where email = ? ;";

  if(mysqli_stmt_prepare($stmt,$query)){
    mysqli_stmt_bind_param($stmt, "s", $_POST["email"]);
    mysqli_stmt_execute($stmt);

    mysqli_stmt_bind_result($stmt,$email,$uuid,$name);

    if(mysqli_stmt_fetch($stmt)){
      mysqli_stmt_close($stmt);

      $token = bin2hex(openssl_random_pseudo_bytes(64));

      $query = 'INSERT into password_reset (`email`,`token`) values (?,?)';
      $stmt = mysqli_stmt_init($DB_USERS);

      if(mysqli_stmt_prepare($stmt,$query)){
        mysqli_stmt_bind_param($stmt, "ss",$email,$token);
        if(mysqli_stmt_execute($stmt)){
          mysqli_stmt_close($stmt);

          $to = $email;
          $subject = "Recuperar contraseña en BehaviourTracker";

          $message = '<p>Hola '.$name.'</p>
```

```
                    <p>Ha solicitado que se le asigne una nueva contraseña en
behaviourtracker.es. Si no realizó esta solicitud, simplemente debe ignorar este
mensaje.</p>
                    <br>
                    <p>Para poder cambiar la contraseña, debes de hacer click
<a href="https://www.behaviourtracker.es/new_password.php?t='.$token.'">AQUÍ</a>
                      <br>Este es el link para el cambio de contraseña en caso
de que no pueda hacer click en el enlace de arriba.
                      <br>URL:
https://www.behaviourtracker.es/new_password.php?t='.$token.'
                    </p>
                    ';
          $headers  = 'MIME-Version: 1.0' . "\r\n";
          $headers .= 'Content-type: text/html; charset=utf-8' . "\r\n";
          $headers .= 'From: Soporte BehaviourTracker
<soporte@behaviourtracker.es>' . "\r\n";

          if(mail($to, $subject, $message, $headers)){
            $error_code = 0;
            $message_check = "Éxito al enviar un Email a ".$email.". Siga las
instrucciones del correo para recuperar su contraseña.";
          }else{
            $error_code = 1;
            $message_check = "No se ha podido enviar un Email a ".$email." por
favor intentelo de nuevo más tarde.";
          }

                              ...
```

Code 4.25: How to send an email.

In this code you can see that a token is created from the *bin2hex()* function, this token is used to link it to an email and serves as a *GET* parameter for the password recovery page.

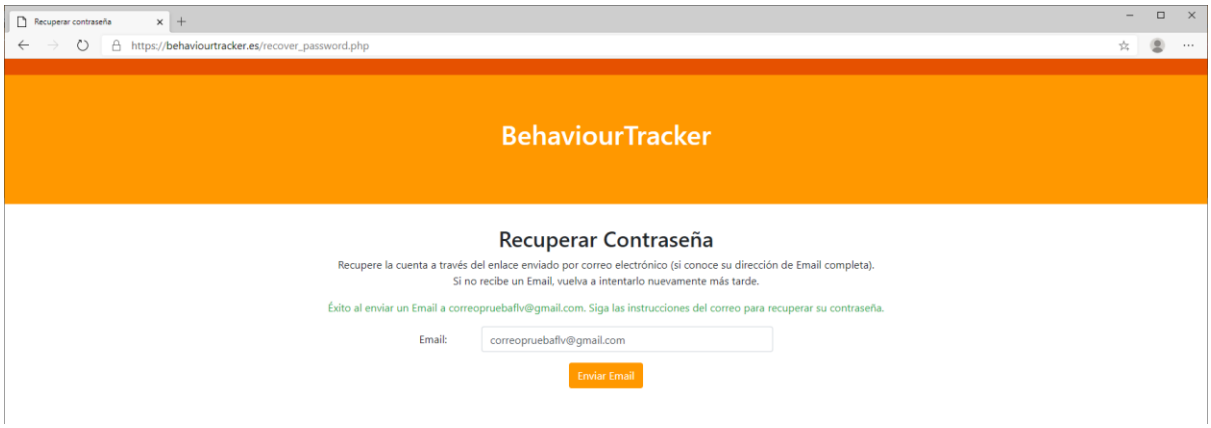


Figure 4.67: Successfully submitted email.

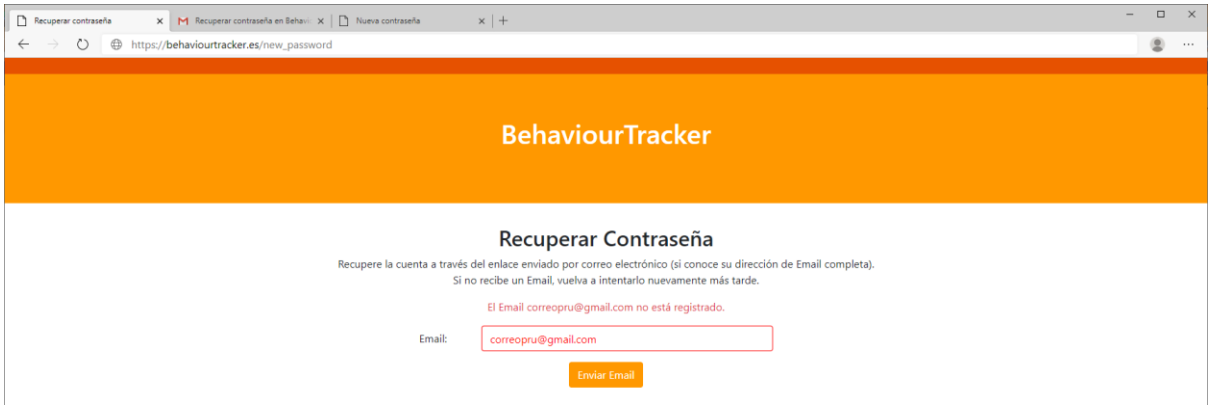


Figure 4.68: Unregistered email.

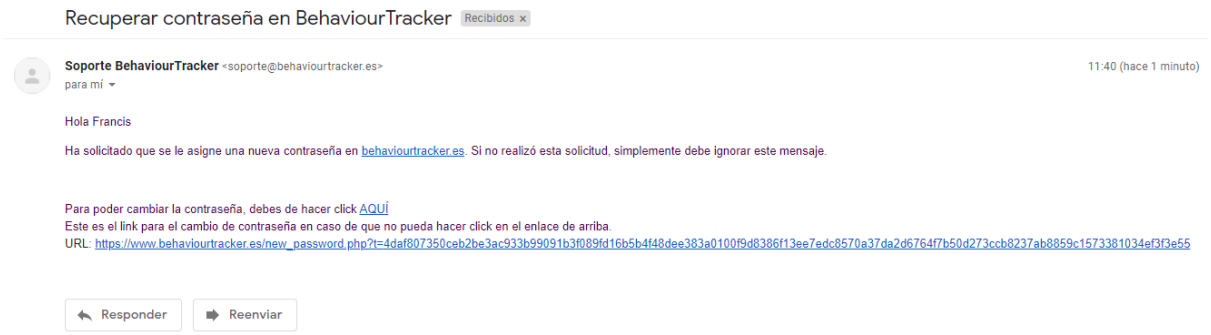


Figure 4.69: Email received.

The figure above shows the mail that is received. It shows the typical information of a recovery mail as well as the special link that you have to access to restore the password.

This link leads to the page to create a new password. The link is composed of the web page address and a random token that is generated as seen in the code above and serves to identify the user who wants to change the password because the token and the email are checked. If both are correct, then the password is changed, and the token is removed from the database so that it cannot be accessed again.

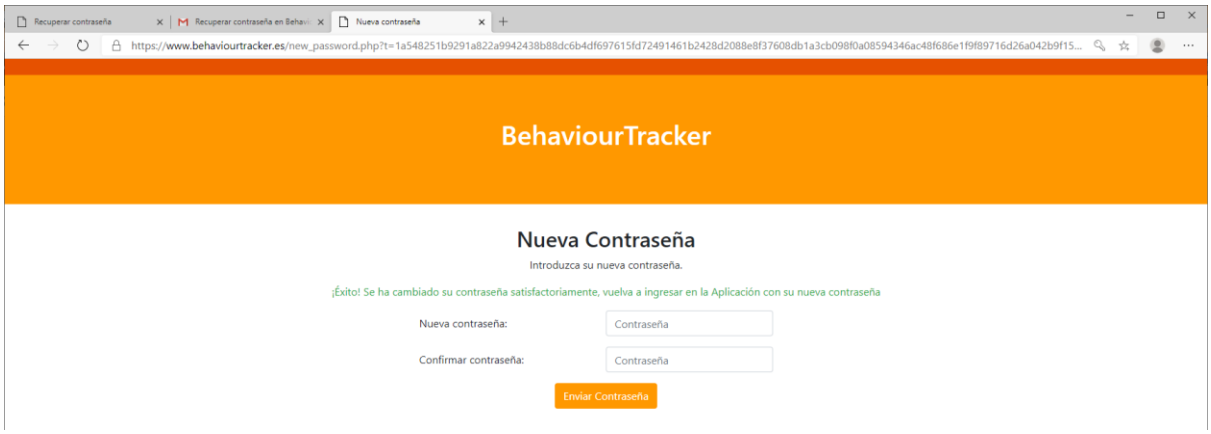


Figure 4.70: Successfully changed password.

5 Evaluation

The evaluation of the system has consisted of a preliminary study of 10 participants (4 female and 6 male; min age: 15 and max age: 59; median age: 40; standard deviation: 16,7) who have tested the mobile application. One of them has tested both the mobile application and the web application during a period of two weeks. The person who has tested the complete system is a expert in psychology in the field of depressive and anxiety disorders, and although the system is oriented to the study of people who suffer from bipolar disorder, it can be extrapolated to other pathologies such as depression in which the psychologist or psychiatrist has a great influence by talking to the person who suffers from the disorder and evaluating him or her through questionnaires. All the participants are asked to respond a usability questionnaire at the end of the testing process.

The usability assessment questionnaire used was the System Usability Scale (SUS) [23] because it provides a quick overview of the usability of the system.

The questions are as follows:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

The questions in this questionnaire are assessed as follows:

- If it is an even question, one point is subtracted.
- If it is an odd question, 5 is subtracted minus the value marked by the user.
- Finally, the 10 results are added together and multiplied by 2.5.

The final result of this questionnaire will be over 100 and is represented as follows:

- Below 50 is a failed system.
- A result of 68 is considered correct.
- Above 80 is outstanding.

5.1 Mobile application assessment

The answers to each question are given in the table below where P1 is participant 1 and Q1 is question 1:

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	**Result**
P1	4	1	5	1	4	2	5	1	4	1	90
P2	3	2	4	3	5	1	4	2	4	1	77,5
P3	5	1	5	1	5	1	5	5	5	1	90
P4	5	1	5	1	5	1	5	1	5	1	100
P5	4	1	5	1	4	2	5	1	5	1	92,5
P6	3	1	5	4	4	1	5	1	5	1	85
P7	4	1	5	1	4	1	5	1	5	1	95
P8	5	1	5	1	3	1	5	1	5	1	95
P9	5	1	5	1	4	1	5	1	4	1	95
P10	5	1	5	1	5	1	5	1	5	1	100
Average											**92**

Table 5.1: Mobile application assessment result.

With an average result of 92, we can say that the mobile application is outstanding and therefore the system meets the requirements of a system with a good design and an easy to use interface.

5.2 Web application assessment

As in the previous table, the assessment of each question will be represented, but for this test, only one person was able to use the system.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	**Result**
P1	5	2	5	1	5	3	4	3	4	4	75
Average											**75**

Table 5.2: Web application assessment result.

With a result of 75, we can say that the web application is correct, this means that the web application fullfils its function, but it must be improved. It is true that the web application is a

management application and not a simple web page, and therefore has a learning curve that requires time to understand how all the system's functionalities are used.

6 Conclusion

In this work, we present a system for monitoring people using questionnaires personalised by the specialists for their follow-up. The system consits of two applications. The first application is an Android application that allows to receive questionnaires display them and send the answers to a server. The second, is a website to host multiple clinical centres formed by an administrator, specialists and secretaries who according to their role, will be able to manage the centre by managing the users of the whole system, and the questions with their categories that are made to the users of the application as well as to visualize the answers of these questions. All this will allow a closer follow-up of the professionals with their respective users, helping the professionals to understand how these users carry out their day-to-day work.

In view of the evaluation results, this system has been satisfactory and therefore, I consider that it could help medical centres to improve the evaluation of their users. Moreover, thanks to its implementation, I have been able to learn new knowledge such as the use of Android and web programming, although I already knew a little about the latter.

It should be noted that a double effort has been required in carrying out this work due to the time invested in learning the new knowledge.

It is also worth mentioning that at the beginning of the research of this work, a smart band was used. This one used Bluetooth, and it was possible to access the data from this smart band obtaining all the data that it had in each minute (heart rate, steps, acceleration, type of activity, sleep). This was not included because the work was large enough, but will be included as future work of the system since the use of these bracelets can be very positive in the collection of data for later analysis.

6.1 Aims achieved

Thanks to the evaluation made in the previous chapter, we can say that we have managed to create a compact and easy to use system, even though the web is somewhat more difficult at first.

In addition, all the objectives, as well as the requirements raised in this work, have been achieved successfully to a greater or lesser extent. Namely, an analysis of the art has been made to understand the two aspects of monitoring a person. A system has been designed to monitor people through questionnaires. And finally, the prototype has been implemented and evaluated by several participants.

In the next section, we will talk about the future work we hope to achieve.

6.2 Future work

This system has been designed to be expanded with new functionalities in the future thanks to the potential it has along with the data obtained. Some of the most important features are the following:

- Extend the mobile application so that it can obtain information in a objective way as seen in the section on state of the art. This information will monitor the status of users through sensors, for example, sleep hours, distances, steps, etc.

- Together with the previous point, the mobile application could be extended to use wearables for monitoring the status tracking information.

- Carry out a real study in which people with mood problems can participate to see how this application influences those people.

- To detect the moods by applying data science with the data from the database and in the case of having extended the mobile application to use them also to be able to be analysed and improve the detection of the mood.

- Generate graphs of the questions according to their type to improve the visualisation of the users' answers.

- To implement a system of warnings with positive notifications for these people designed by the professional.

- Implement the relationship between users of the mobile application so that each user can have a person in charge and thus be able to compare information about what a user says and the person or persons in charge.

7 Bibliography

1. Diagnostic and statistical manual of mental disorders. American Psychiatric Publishing; 2013.

2. Altman, Edward G., et al. “The Altman Self-Rating Mania Scale.” *Biological Psychiatry*, Elsevier, 3 Feb. 1998, www.sciencedirect.com/science/article/pii/S0006322396005483.

3. Rush, A.John, et al. “The 16-Item Quick Inventory of Depressive Symptomatology (QIDS), Clinician Rating (QIDS-C), and Self-Report (QIDS-SR): a Psychometric Evaluation in Patients with Chronic Major Depression.” *Biological Psychiatry*, Elsevier, 11 Apr. 2003, www.sciencedirect.com/science/article/pii/S0006322302018668.

4. Robert L. Spitzer, MD. “A Brief Measure for Assessing Generalized Anxiety Disorder.” *Archives of Internal Medicine*, American Medical Association, 22 May 2006, jamanetwork.com/journals/jamainternalmedicine/article-abstract/410326.

5. “EQ-5D.” *Wikipedia*, Wikimedia Foundation, 13 Aug. 2020, en.wikipedia.org/wiki/EQ-5D.

6. Depp, Colin A, et al. “Augmenting Psychoeducation with a Mobile Intervention for Bipolar Disorder: A Randomized Controlled Trial.” *Journal of Affective Disorders*, Elsevier, 8 Nov. 2014, www.sciencedirect.com/science/article/pii/S0165032714006855.

7. Tsanas, A., et al. “Daily Longitudinal Self-Monitoring of Mood Variability in Bipolar Disorder and Borderline Personality Disorder.” *Journal of Affective Disorders*, Elsevier, 2 July 2016, www.sciencedirect.com/science/article/pii/S0165032716307819#f0010.

8. Cochran, Amy, et al. “Engagement Strategies for Self-Monitoring Symptoms of Bipolar Disorder With Mobile and Wearable Technology: Protocol for a Randomized Controlled Trial.” *JMIR Research Protocols*, JMIR Publications, 10 May 2018, www.ncbi.nlm.nih.gov/pmc/articles/PMC5968216/.

9. Abdullah, Saeed, et al. “Automatic Detection of Social Rhythms in Bipolar Disorder.” *OUP Academic*, Oxford University Press, 14 Mar. 2016, academic.oup.com/jamia/article/23/3/538/2909018.

10. Beiwinkel1, Till, et al. “Using Smartphones to Monitor Bipolar Disorder Symptoms: A Pilot Study.” *JMIR Mental Health*, JMIR Publications Inc., Toronto, Canada, mental.jmir.org/2016/1/e2/.

11. *Wearable Monitoring for Mood Recognition in Bipolar Disorder Based on History-Dependent Long-Term Heart Rate Variability Analysis - IEEE Journals & Magazine*, ieeexplore.ieee.org/document/6661378.

12. Faurholt-Jepsen, Maria, et al. “Smartphone Data as Objective Measures of Bipolar Disorder Symptoms.” *Psychiatry Research*, Elsevier, 13 Mar. 2014, www.sciencedirect.com/science/article/pii/S0165178114001875.

13. Mads Frost IT University of Copenhagen, et al. “Supporting Disease Insight through Data Analysis: Refinements of the Monarca Self-Assessment System.” *Supporting Disease Insight through Data Analysis | Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 1 Sept. 2013, dl.acm.org/doi/10.1145/2493432.2493507.

14. Forte, Fernando. “Smartphone OS Market Share in Spain 2013-2020.” *Statista*, 29 July 2020, www.statista.com/statistics/260419/market-share-held-by-smartphone-operating-systems-in-spain/.

15. “Save Data in a Local Database Using Room.” *Android Developers*, developer.android.com/training/data-storage/room.

16. “Save Key-Value Data.” *Android Developers*, developer.android.com/training/data-storage/shared-preferences.

17. “Services Overview.” *Android Developers*, developer.android.com/guide/components/services.

18. “Broadcasts Overview.” *Android Developers*, developer.android.com/guide/components/broadcasts.

19. “Optimize for Doze and App Standby.” *Android Developers*, developer.android.com/training/monitoring-device-state/doze-standby.

20. “Add Advanced Interaction Controls to Your HTML Tables the Free & Easy Way.” *DataTables*, datatables.net/.

21. “XML HttpRequest.” *XML HttpRequest*, www.w3schools.com/xml/xml_http.asp.

22. “PHP Fine Diff” *PHP Fine Diff*, www.raymondhill.net/finediff/viewdiff-ex.php.

23. Affairs, Assistant Secretary for Public. “System Usability Scale (SUS).” *Usability.gov*, Department of Health and Human Services, 6 Sept. 2013, www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

24. Iberley.es. “Convenio Colectivo Del Comercio. GRANADA.” Iberley, Iberley. El Valor De La Confianza., 4 July 2018, www.iberley.es/convenios/sector/convenio-colectivo-comercio-granada-4200050.

25. “W3SCHOOLS.” *w3schools*, www.w3schools.com.