COLLECTING AND PREPROCESSING DATA ON MOOD STATES: A FIREBASE-ANDROID SYSTEM

Darío Fernando Gallegos Quishpe
Mishell Janina Tigse Ortiz

UNIVERSIDAD COMPLUTENSE
MADRID

Bachelor's Degree Final Project in Computer Science and Engineering

Directora
María Victoria López López

Madrid, Enero 2020

Authorization of diffusion and use

Darío Fernando Gallegos Quishpe **Mishell Janina Tigse Ortiz**

09th January 2020

"Things do not go wrong and break your heart so you can become bitter and give up. They happen to break you down and build you up so you can be all that you were intended to be."

Charlie Jones

Acknowledgement

First of all, we would like to thank our director of the final project, Maria Victoria Lopez Lopez for her infinite patience, her help and suggestions, and her constant support.

We would also like to thank all the teachers who have accompanied us throughout this adventure. Thank you for your teachings, for your effort and for building the pillars with which in the future we will become great professionals in our field.

Finally, we would like to thank our family for joining us on this path, always giving us the strength to continue and never letting us give up. Thank you for making this possible.

Contents

List of Figures

List of Tables

Abstract

This study focuses on the field of medicine and aims to provide a solution to the difficulty of identifying mood disorders and the need to speed up this process through the use of technological tools.

To achieve this goal a system has been developed with a user-friendly interface that facilitates their interaction with it. As well as with the capacity to collect and store information of the users through a scalable database, capable of storing large amounts of information and multimedia files such as images and audios.

This system is focused to be managed by a psychiatrist and used by patients with mood disorders, allowing to track the activity and behavior of these patients in their daily lives. Making this system a useful tool to help the psychiatrist in his decision making and help future systems to make a better prediction of these disorders thanks to its stored information.

Resumen

Este estudio se centra en el campo de la medicina y pretende dar solución a la dificultad de identificar los trastornos del estado de ánimo y a la necesidad de acelerar este proceso mediante el uso de herramientas tecnológicas.

Para lograr este objetivo se ha desarrollado un sistema con una interfaz amigable que facilita su interacción con el mismo. Así como con la capacidad de recoger y almacenar información de los usuarios a través de una base de datos escalable, capaz de almacenar grandes cantidades de información y archivos multimedia como imágenes y audios.

Este sistema está enfocado para ser gestionado por un psiquiatra y utilizado por pacientes con trastornos del estado de ánimo, permitiendo realizar un seguimiento de la actividad y el comportamiento de estos pacientes en su vida diaria. Haciendo de este sistema una herramienta útil para ayudar al psiquiatra en su toma de decisiones y ayudar a futuros sistemas a realizar una mejor predicción de estos trastornos gracias a su información almacenada.

Keywords

English

Emotional disorders, Android, mobile applications, databases, Firebase, GitFlow, Trello, Scrum.

Spanish

Trastornos emocionales, Android, aplicaciones móviles, bases de datos, Firebase, GitFlow, Trello, Scrum.

Chapter 1

Introduction

In 2019, the Regional Office for Europe of the World Health Organization [1], reports that 25 percent of the population suffers from anxiety and depression, which are symptoms of mood disorders. As we know, there exist multiple mood disorders; however, in this study, we are going to focus on depression and bipolar disorders.

These disorders are chronic and severe mental health problems that have a high suicide index, and they are a significant problem in Europe.

Two of the biggest problems we find when identifying these disorders are the difficulty of the detection of those in itself and that traditional indicators and indexes like Hamilton and Yang are insufficient to predict and to avoid crises. In one hand, it is difficult to detect the disorder since people who suffer from them tend to lie about their feelings, stop with the medication or hide their situation as much as possible and on the other hand, traditional indicators have crises prediction bad because they do not take into account enough variables to make a good prediction.

To solve this situation and help as much as possible to patients, it is important to find an acceptable method to collect data being objective and without being intrusive. So after analyzing the different symptoms, it has been found that much of these symptoms can be collected with the help of the sensors of smartphones, smart bands, and smartwatches.

For example, into the depression, low energy and oversleeping are relevant. Bipolar disorders have lows known as depression and highs known as mania. Often depression symptoms are social isolation, low energy, oversleeping or insomnia and suicidal thoughts, and often mania symptoms are elevated mood, extreme irritability and the decreased need to sleep.

All these signs have in common that they can be detected through monitoring the activity and behavior of the patients in daily life. This is why a system that collects and processes this data helped by the sensors of smartphones could be a useful tool to manage enough variables to do better detection and to help future systems to improve

crisis prediction of these disorders.

1.1 Research context

This final study springs from the Bit4cast project [2, 3, 4], which studies the biological, psychological and physical signals of patients [5, 6, 7] to know which variables are correlated and which features are important to build a crisis prediction model for patients with bipolar disorder.

The main aim of the Bit4cast project is the development of a system that helps the psychiatrist to predict a crisis of a patient in advance so that the crisis can be stopped or in case it occurs, the consequences are the minimum possible.

There is a wide range of professionals working on this project, including professors, master and degree students, two doctoral students, a psychiatrist and patients with Bipolar Disorder. The project is developed by Clínica Nuestra Señora de la Paz[8], a non-profit center dedicated to Mental Health and supported by 3EGA [9], a startup from Amsterdam which is working on a revolutionary new method that could help diagnose and treat patients more efficiently than ever.

Various other research projects have been very relevant for this study but we want to emphasize two projects, the Bip4cast project since thanks to this project, this study has been possible and the Introduction to Big Data and First Steps in a Big Data Project [10], which has given us the necessary guidelines to start this study.

The goal of the specific study described herein has been to try to collect as much patient data as we can through the developing of a firebase-android system. To define its direction and scope, we plan the following specific objectives:

- To develop a system that allows users to collect and store information through a scalable database capable of storing large amounts of information and multimedia files such as images and audios. Besides, the users could be able to observe and carry out control of the information uploaded.
- To extract the information from the database in the form of a CSV file to make it easier to analyze since the data collected through this system will be used to be analysed by another system later to guess the evolution of the distribution of the real state of the patient.
- To design a friendly interface that attracts users and facilitates interaction with the system.

During the development of this study, the knowledge obtained during the Computer Science Engineering degree has been very present.

Before starting this study, we had barely had contact with the development of mobile applications. However, subjects such as *Object-Oriented Programming* and *Databases* helped us a lot in the development of the system with Android and Firebase technologies. Besides, *Software Engineering* helped to structure and to divide the study into phases, applying software design patterns and finally, *Interactive Systems Design* supported the creation of an intuitive and easy-to-use system with a great design.

1.2 Work plan

The software development process can be a complex and far-reaching task. To make life easier, we can use software development models, each of which describes a unique approach to different activities in the development life cycle. Using the cascade model [12] as a guide, we have adapted the procedure according to our needs, obtaining the following stages: compilation and analysis of similar systems, specification of requirements, generation of a prototype or mockup, the definition of the technologies used, construction and implementation of the software and usability testing (see Figure 1.1).

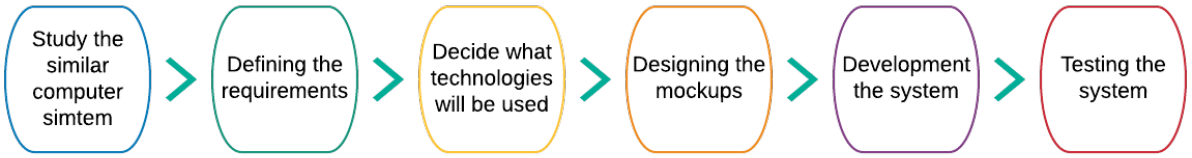

Figure 1.1: Work plan diagram

- The first part of the study consists of studying the computer systems in the area of disorders moods to define the collective characteristics and what differential value we can provide.
- The next part of the study consists of defining the requirements, which will help us to establish the basis of the system. We start from the Bip4cast project, which will serve as a guide when it comes to profiling the software, and from there, we will provide our vision. To this end, we have organized several meetings with the tutor to establish all the functions. Once the functionalities of the system have been defined, it will be possible to determine the requirements of the application and with it, begin to design the prototype.
- Before starting development, it is necessary to decide what technologies will be used. To do this, we will conduct research comparing the pros and cons of each one. It should be clarified that the system is focused Android operating system so that tools that are not compatible will be discarded. Finally, the language chosen to project is Java but oriented to programming with Android. For the development, we used the free IDE Android Studio, provided by Google. Most debugging will do with our mobile phones, OPPO RX17 Neo and Xiaomi Mi A2, and also in the Android Studio which has several virtual devices with various distinct API version and screen sizes.

- Once we have an overview of the technologies, we will focus on designing the first mockups of the system. A mockup is a representative model of a final software view. It allows us to show an approximation of the final result without having to go through the entire development process. Each team member will create four to six mockups taking into account the defined requirements. Each member can contribute their vision to the design of the views. Once this is done, we will choose the best solutions or a mix of several.
- Designing efficient software, making the most of the development time is a complicated task. To structure, plan and control the development process, we will use a software development model, adapted to our needs and the Scrumbat work methodology. While it is true that this task may be the most extended, thanks to the prototype previously designed, we can streamline the implementation of the system.
- The last part of the study is to evaluate the system. We will look for people to test the system to corroborate the functionality and detect possible failures, bugs and the own acceptance of the application. The evaluation will be carried out using printed forms and Google forms, where they will be able to indicate the user's experience, possible changes or existing errors.

1.3 Structure of memory

This work includes a detailed description of the whole process followed in this study. It has been structured in different chapters to make reading easier:

Chapters 1 and 2 contain the Introduction, both in English and Spanish. The background and the goals of the study are explained in those chapters.

Chapter 3 is the State of the Art, explains the similar studies that have been carried out in this field, the pros and cons of the existing software and the general comparison of these applications with our system.

Chapter 4 is Used Technologies, where the tools and technologies used are explained as well as their use during the application.

Chapter 5 is the Uses Cases, which specifies the use cases of the system grouped according to the module to which they belong, user module or data module.

Chapter 6 is the Design, which explains the whole design process of the system from the initial creation of the mockups to the final prototyping.

Chapter 7 is the Firebase, which describes what Firebase, the different services which provide, the basic concepts, the services used by our system, the access rule and the

scaling of the database is.

Chapter 8 is the Development with Android, which includes a brief introduction to Android, its basic concepts, the specification, the implementation and the future changes of our system.

Chapters 9 and 10 include the Conclusions and the Future Work, in English and Spanish, which contains a summary of the conclusions of the study as well as the future work that can be applied.

This work is completed with a Bibliography where this study can be validated and consists of some web pages whose last access date is January 2020.

Chapter 2

Introducción

En 2019, la Oficina Regional para Europa de la Organización Mundial de la Salud (OMS) [1] informa que el 25 por ciento de la población sufre de ansiedad y depresión, que son síntomas de trastornos del estado de ánimo. Como sabemos, existen múltiples trastornos del estado de ánimo; sin embargo, en este estudio nos vamos a centrar en la depresión y los trastornos bipolares.

Estos trastornos son problemas de salud mental crónicos y severos que tienen un alto índice de suicidio, y son un problema significativo en Europa.

Dos de los mayores problemas que encontramos a la hora de identificar estos trastornos son la dificultad de la detección de los mismos y que los indicadores e índices tradicionales como Hamilton y Yang son insuficientes para predecir y evitar las crisis. Por un lado, es difícil detectar el trastorno ya que las personas que lo padecen suelen mentir sobre sus sentimientos, dejar la medicación o esconder su situación lo máximo posible y por otro lado, los indicadores tradicionales tienen una mala predicción de las crisis porque no tienen en cuenta las suficientes variables para hacer una buena predicción.

Para resolver esta situación y ayudar en la medida de lo posible a los pacientes, es importante encontrar un método aceptable de recogida de datos que sea objetivo y no intrusivo. Así que después de analizar los diferentes síntomas, se ha encontrado que muchos de estos síntomas pueden ser recogidos con la ayuda de los sensores de los teléfonos inteligentes, las bandas inteligentes y los relojes inteligentes.

Por ejemplo, en la depresión, la baja energía y el exceso de sueño son relevantes. Los trastornos bipolares tienen bajas conocidas como depresión y altas conocidas como manía. A menudo los síntomas de la depresión son el aislamiento social, la baja energía, el dormir demasiado o el insomnio y los pensamientos suicidas, y a menudo los síntomas de la manía son el estado de ánimo elevado, la irritabilidad extrema y la disminución de la necesidad de dormir.

Todos estos signos tienen en común que pueden ser detectados a través de la monitor-

ización de la actividad y el comportamiento de los pacientes en la vida diaria. Por ello, un sistema que recoja y procese estos datos ayudado por los sensores de los teléfonos inteligentes podría ser una herramienta útil para gestionar suficientes variables para hacer una mejor detección y ayudar a futuros sistemas a mejorar la predicción de crisis de estos trastornos.

2.1 Contexto de investigación

Este estudio final surge del proyecto Bit4cast [2, 3, 4], que estudia las señales biológicas, psicológicas y físicas de los pacientes [5, 6, 7] para saber qué variables están correlacionadas y qué características son importantes para construir un modelo de predicción de crisis para pacientes con trastorno bipolar.

El objetivo principal del proyecto Bip4cast es el desarrollo de un sistema que ayude al psiquiatra a predecir una crisis de un paciente de antemano para que la crisis pueda ser detenida o, en caso de que ocurra, las consecuencias sean las mínimas posibles.

Hay una amplia gama de profesionales trabajando en este proyecto, incluyendo profesores, estudiantes de máster y grado, dos estudiantes de doctorado, un psiquiatra y pacientes con Trastorno Bipolar. El proyecto es desarrollado por la Clínica Nuestra Señora de la Paz [8], un centro sin fines de lucro dedicado a la Salud Mental y apoyado por 3EGA [9], una startup de Ámsterdam que está trabajando en un nuevo y revolucionario método que podría ayudar a diagnosticar y tratar a los pacientes más eficientemente que nunca.

Varios otros proyectos de investigación han sido muy relevantes para este estudio pero queremos destacar dos proyectos, el proyecto Bip4cast ya que gracias a este proyecto, este estudio ha sido posible y el proyecto Introduction to Big Data and First Steps in a Big Data Project [10], que nos ha dado las pautas necesarias para iniciar este estudio.

El objetivo del estudio específico aquí descrito ha sido intentar recopilar la mayor cantidad posible de datos de pacientes mediante el desarrollo de un sistema firebase-android. Para definir su dirección y alcance, planificamos los siguientes objetivos específicos:

- Desarrollar un sistema que permita a los usuarios recopilar y almacenar información a través de una base de datos escalable capaz de almacenar grandes cantidades de información y archivos multimedia como imágenes y audios. Además, los usuarios podrán observar y controlar la información cargada.
- Extraer la información de la base de datos en forma de archivo CSV para facilitar su análisis ya que los datos recogidos a través de este sistema se utilizarán para ser analizados por otro sistema más adelante para adivinar la evolución de la distribución del estado real del paciente.

- Diseñar una interfaz amigable que atraiga a los usuarios y facilite la interacción con el sistema.

Durante el desarrollo de este estudio, los conocimientos obtenidos durante la carrera de Ingeniería Informática han estado muy presentes.

Antes de iniciar este estudio, apenas habíamos tenido contacto con el desarrollo de aplicaciones móviles. Sin embargo, asignaturas como *Programación orientada a objetos* y *Bases de datos* nos ayudaron mucho en el desarrollo del sistema con tecnologías Android y Firebase. Además, *Ingeniería del Software* ayudó a estructurar y dividir el estudio en fases, aplicando patrones de diseño de software y finalmente, *Diseño de sistemas interactivos* apoyó la creación de un sistema intuitivo y fácil de usar con un gran diseño.

2.2 Plan de trabajo

El proceso de desarrollo de software puede ser una tarea compleja y de gran alcance. Para hacer nuestra vida más fácil, podemos utilizar modelos de desarrollo de software, cada uno de los cuales describe un enfoque único de las diferentes actividades en el ciclo de vida del desarrollo. Utilizando el modelo de cascada [12] como guía, hemos adaptado el procedimiento de acuerdo a nuestras necesidades, se obtienen así las siguientes etapas: compilación y análisis de sistemas similares, especificación de requisitos, generación de un prototipo o maqueta, definición de las tecnologías utilizadas, construcción e implementación del software y pruebas de usabilidad (ver Figura 2.1).

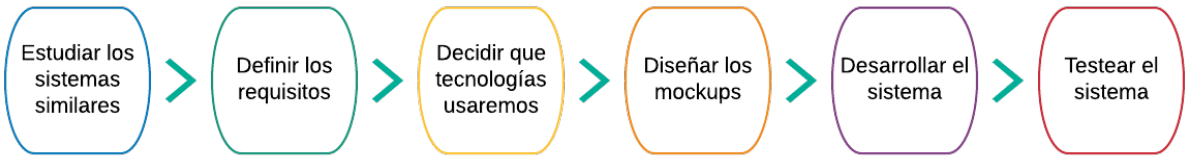

Figure 2.1: Diagrama plan de trabajo

- La primera parte del estudio consiste en el estudio de los sistemas informáticos en el ámbito de los estados de ánimo para definir las características colectivas y el valor diferencial que podemos aportar.
- La siguiente parte del estudio consiste en definir los requisitos, lo que nos ayudará a establecer las bases del sistema. Partimos del proyecto Bip4cast, que servirá de guía a la hora de perfilar el software, y a partir de ahí, daremos nuestra visión. Para ello, hemos organizado varias reuniones con el tutor para establecer todas las funcionalidades. Una vez definidas, será posible determinar los requisitos de la aplicación y con ello comenzar a diseñar el prototipo.
- Antes de iniciar el desarrollo, es necesario decidir qué tecnologías se utilizarán. Para ello, llevaremos a cabo una investigación comparando los pros y los contras de cada una de las tecnologías. Debe aclararse que el sistema está enfocado en Android por

lo que las herramientas que no sean compatibles serán descartadas. Finalmente, el lenguaje elegido para proyectar es Java pero orientado a la programación con Android. Para el desarrollo, utilizamos el IDE gratuito Android Studio, proporcionado por Google. La mayoría de las depuraciones se realizarán con nuestros teléfonos móviles, OPPO RX17 Neo y Xiaomi Mi A2, y también en el Android Studio, que tiene varios dispositivos virtuales con diferentes versiones de API y tamaños de pantalla.

- Una vez que tengamos una visión general de las tecnologías, nos centraremos en el diseño de los primeros mockups del sistema. Una mockup o maqueta es un modelo representativo de una vista final del software. Nos permite mostrar una aproximación del resultado final sin tener que pasar por todo el proceso de desarrollo. Cada miembro del equipo creará de cuatro a seis mockups teniendo en cuenta los requisitos definidos y aportando su visión al diseño de las vistas. Una vez hecho esto, elegiremos las mejores soluciones o una mezcla de varias.
- Diseñar un software eficiente, aprovechando al máximo el tiempo de desarrollo es una tarea complicada. Para estructurar, planificar y controlar el proceso de desarrollo, utilizaremos un modelo de desarrollo de software, adaptado a nuestras necesidades y a la metodología de trabajo de Scrumbat. Si bien es cierto que esta tarea puede ser la más extensa, gracias al prototipo previamente diseñado, podemos agilizar la implementación del sistema.
- La última parte del estudio consiste en evaluar el sistema. Buscaremos personas que prueben el sistema para corroborar la funcionalidad y detectar posibles fallos, errores y la propia aceptación de la aplicación. La evaluación se realizará mediante formularios impresos y de Google, donde se podrá indicar la experiencia del usuario, posibles cambios o errores existentes.

2.3 Estructura de la memoria

Este trabajo incluye una descripción detallada de todo el proceso seguido en este estudio. Se ha estructurado en diferentes capítulos para facilitar la lectura:

Los capítulos 1 y 2 contienen la Introducción, en inglés y español, en la que se explican los antecedentes del estudio así como los objetivos a alcanzar.

El capítulo 3 es el estado del arte, explica los estudios similares que se han llevado a cabo en este campo, los pros y contras del software existente y la comparación general de estas aplicaciones con nuestro sistema.

El Capítulo 4 es Tecnologías Usadas, donde se explican las herramientas y tecnologías utilizadas, así como su uso durante la aplicación.

El Capítulo 5 es el de Casos de Uso, que especifica los casos de uso del sistema agrupados según el módulo al que pertenezcan, módulo de usuario o módulo de datos.

El Capítulo 6 es el Diseño, que explica todo el proceso de diseño del sistema desde la creación inicial de las maquetas hasta el prototipo final.

El capítulo 7 es el Firebase, que describe lo que es Firebase, los diferentes servicios que proporciona, los conceptos básicos, los servicios utilizados por nuestro sistema, las reglas de acceso y el escalado de la base de datos.

El Capítulo 8 es el Desarrollo con Android, que incluye una breve introducción a Android, sus conceptos básicos, la especificación, la implementación y los cambios futuros de nuestro sistema.

Los Capítulos 9 y 10 incluyen las Conclusiones y el Trabajo Futuro, en inglés y español, que contienen un resumen de las conclusiones del estudio, así como el trabajo futuro que se puede aplicar.

Este trabajo se completa con una Bibliografía donde se puede validar este estudio y consta de unas páginas web cuya última fecha de acceso es Enero de 2020.

Chapter 3

State of art

In this chapter, we will explain the current state of the art regarding the areas that will be dealing with in our project, as is the case of data collection and processing systems on mood states.

The main problem with this project is to make an early and accurate prediction of mood disorders; it is necessary to have a reliable and up-to-date data source. A repository of reliable information is essential for the development of studies and tools focused on the prevention and prediction of mood disorders. An example is the Machine Learning algorithms that allow the design of classification and regression models which help to elaborate on the diagnosis of different diseases. The algorithms receive and analyze input data to predict output values within an acceptable range. As new data are introduced, the algorithms learn and optimize their operations to improve performance, developing some form its intelligence.

Different projects and studies are currently being carried out in the area of mood disorders. To describe the current picture, we will focus on two areas: studies related to the prediction of mental disorders and mood, and similar software of our system. The most relevant for the project are presented below.

3.1 Similar studies

Mood disorders have been a very relevant topic in recent years. Various studies [3, 5, 6, 7] at present being conducted in the field of mental health, which is supported by technology to collect data and health indicators from patients. Among the most outstanding studies are: studies related to the prediction of mental disorders [10, 13], studies that focus on the extraction of patient data smartphone data for symptom measuring and studies[11] that focus on the analysis and design of data collections for later use [12].

3.1.1 Smartphone-based state change recognition and patient monitoring

Some studies [11, 7] are trying to detect the state and the changes of a patient using technologies such as mobile sensors which are integrated into smartphones. These studies are similar to this one since in our project we have developed an application that collects data of the patient through the interaction of him/her with the application which uses smartphone sensors such as the microphone and the camera.

3.1.2 Smartphone data for symptom measuring

These studies aim to correlate Bipolar Disorder symptoms with objective data gathered from smartphones [10]. Includes both social and physical activities that patients have to write down every evening for a specified period.

However, in our study, we provide to the patient with an application where they have to fill some easy tasks such as recording his/her voice or take a selfie. They do not have to write down anything but the application knows the state of the patient because it collects a lot of the information of the patient daily, information such as his/her mood, energy, concentration, etc.

3.1.3 Design and Implementation of the Data Collection

Some studies [6, 12] focus on how information about a set of actors involved in the implementation of a project has to be collected, treated, analyzed and disseminated. Knowing what data is essential for the design and implementation of prediction models is one of the main tasks. According to the types of data used in the studies, we select the most relevant for the investigation and provide new sources of information. With this, we seek that our data collection system can be implemented in the existing system and differentiate ourselves from other similar projects. The new data entered are voice audios and daily user selfies.

3.2 Similar software

Mental health is a field of continuous development for mobile applications. WHO [1] suggests that prevention is the key to helping people with their mental health, and with the use of new technologies, this task is more comfortable to carry out.

Until now, therapy was the standard way of treating people with mental health problems, but there are barriers to accessing [14] them: therapies can be costly, sessions that must be adapted to the schedule of the patient and the specialist, and the stigma of attending the treatment itself. The mental health applications [15] seek to close the gap between the patient and the treatment, allowing anyone with a mobile device to

self-manage their condition. For this, we will have tools such as mindfulness, cognitive-behavioural therapy (CBT) [17] and peer support.

Mental health applications [16] have a low cost or are free and with them, you can work at any time and anywhere. These applications do not seek to replace individual therapy but to be a tool to improve treatments. Below are some examples of mobile applications, all of them compatible with Android and iOS.

3.2.1 MindShift

It's a mobile application [18] designed in collaboration with the Anxiety Canada Association [19] to help young people cope with anxiety. It acts as a personal assistant that guides users through difficult situations. It uses scientifically proven strategies based on cognitive-behavioural therapy (CBT) [17] to help learn to relax, be aware of the environment, and develop more effective ways of approaching the problem. Among its most important features are:

- Thought journal where write thoughts or concerns.
- Cards to re-adjusting the thinking with helpful coping statements.
- Overcome your fears by gradually facing them with small tasks and objectives.
- Keep track of anxiety and mood over time with graphs and journal entries.
- Healthy habits and tips to set the stage to better manage anxiety.

3.2.2 Moodpath

It is an interactive system for detecting anxiety and depression. This application [20] analyzes the psychology, emotional and physical health over two weeks to generate an evaluation of the user's mental health. The app also includes an educational section to teach users about the psychology that exists behind their mood, symptoms of depression and anxiety.

For a person who has some emotional disorder, it can be challenging to get a general picture of their mental health. Moodpath provides a structured report that could help users know what treatments to find the shortest path to their sickness. To elaborate the report Moodpath set up:

- Daily questions about your emotional and physical well-being.
- Get a weekly mental health assessment.
- Share your evaluation with therapists, psychologists, and other healthcare specialists.
- One intelligent mood tracker and journal.

- A swift overview of your emotional states during the day.
- Receive helpful insights to understand behaviour patterns and triggers.
- Exercises based on cognitive-behavioural therapy [17] to learn to let go of negative thoughts and overwhelming emotions.

3.2.3 MoodTools

It is an application designed to help users fight depression and alleviate negative moods. Mootools [21, 22] has a simple and intuitive interface, which can adapt according to the needs of users. It offers a set of tools based on cognitive-behavioural therapy and behavioural activation therapy (BAT) [24] that includes:

- Thought Diary: Improve your state by analyzing the thoughts and identifying negative distorted based on principles from Cognitive Therapy.
- Safety Plan develops a suicide safety plan to keep you safe and utilize in suicidal crisis.
- A PHQ-9 [23] depression survey to monitory your symptom severity.
- Help videos to improve your mood as meditation guides.
- Regain energy by performing energizing activities based on behavioral activation therapy [24].

3.2.4 TalkLife

It is a support tool where people can talk with others who have lived through similar situations. This mobile application [25] connects users in a community so they can share their struggles with depression and anxiety. It is flexible enough to work at different levels of comfort for users such as anonymous posts, private chat with other users, public announcements and all to share and receive support. In this way, thousands of users can share their experiences as if it were a social network without feeling embarrassed. It also has an integrated blog that provides content to read and discuss with others.

3.2.5 Sanvello

Sanvello [26] is a subscription application that seeks to break the cycle of negative thoughts that lead to stress, anxiety and depression. It consists of tools designed by psychologists such as online care, self-help guided lessons, humour and health monitoring lessons, CBT techniques, and a supportive community. This application offers a balance between the tools and the user interface to achieve an excellent user experience that keeps users committed with the app.

3.3 Conclusions

Following the research work, it has been determined that market applications mostly use mood indicators to identify the user's diagnosis. Although those indicators of state can indeed generate elaborate diagnoses, the lack of variety of data types can be a disadvantage in determining health assessment. Therefore, our system will focus on collecting and store data that is not usually used, such as voice recordings and daily selfies, also including mood indicators.

Chapter 4

Used Technologies

Technology is evolving at high speed, new ideas and frameworks are appearing every day, and it is not very easy to decide which tools to use. The following is a description of the technological resources used in the development of this project.

4.1 React Native

At the very beginning, we decided to develop the application with React Native [27] since it is an emerging technology which is being more and more important nowadays. However, the fact of being a new and developing technology, far from being an aid, is sometimes a problem, because many of its updates bring bugs with it that took a long time to fix and did not allow backward compatibility with previous versions, which made what functionalities that previously worked, they stopped doing it. Another consequence of this was the lack of documentation that sometimes we found regarding problems that we found in the development of the application.

As a result, we decided to develop the mobile application with Android Studio which uses native Java, has a stable environment and plenty of documentation about it.

4.2 GitFlow

Git-flow [28] is a set of extensions made for Git [29] to facilitate the work with repositories. The followed model was first designed and made popular for Vincent Driessen. This set of extensions defines a strict branching that will be defined later, around the project release. As a consequence, this provides a robust framework for managing larger products.

We have chosen Git-Flow because it is ideally suited for big projects that have a scheduled release cycle and as we had different features scheduled at different times and developed at the same time, we wanted to have a stable version of the mobile application that only would be updated in case the developed features were stable and worked perfectly. Besides, this workflow does not add any new concepts or commands beyond what

is required for the Feature Branch Workflow. Instead, Git-Flow assigns very specific roles to different branches and defines how and when they should interact. Also, it allowed us to use individual branches for preparing, maintaining and recording the new functionalities. Of course, you also have to leverage all the benefits of the Feature Branch Workflow such as pull requests, isolated experiments and more efficient collaboration which is very important when you are developing a project in pairs.

As we have mentioned before, Git-Flow assigns very specific roles to its branches and defines how and when they should interact. Below, we explain the different branches and how we have taken advantage of them:

- Master: Master branch is the one that contains the last stable version of the project and it is the one that is going to be updated each time the new features of the application have been tested and works perfectly in the develop branch.
- Develop: Develop branch is the one where all the developed features coded in the feature branches are proved once they have been finished and in case of discovering any bug, is the one where the bugfix branches are applied. Once all the features are proved and everything works well, everything inside this branch is copied to a release branch which later will be merged to master giving it the last stable version of the application.
- Feature: A feature branch is used to develop and test new functionalities, one for each branch. This branch is created of the develop branch and once the new functionality is done and tested, it is merged again with develop. The name of the branch describes the functionality which is being developed inside it and the first commit includes the Trello card url which describes the functionality to be developed.
- Bugfix: Bugfix branch is used to fix errors in the developed software. This branch is created from the develop branch once an error has been identified and once the error has been solved, it is merged with the develop branch. Besides, the first commit of this branch is the Trello card url associated to the error that has to be solved in the branch, we do not have to forget that the name of the bugfix has to describe the error found.
- Release: Release branch is used to launch a new version of the application. This branch contains all the new developed features in the version, together with a JSON file called changelog which contains the new features which are included. Besides, another file with the version of the release called version is included also. Release branch is created from the develop branch once everything planned for that version works properly and is merged with the master branch.
- Hotfix: Hotfix branch has the same objective that bugfix branch, solve errors of the developed software. However, the hotfix branch is created from the master branch and once the error is solved, it is merged with the master branch. Besides, it includes a JSON file called changelog with the errors that have been solved and its first commit includes the Trello card url which described the found error. As the name of the bugfix branch, the hotfix branch name also has to be very descriptive.

- Support: Support branch is a branch created to solve errors in the application that are found in previews versions, not in the actual version. This branch has to be named as the version where the error has been found and in production, after the error is solved, a new release has to be done however as our application is not really big and we do not have many versions, we merge the support branch in the master branch after the error is solved. For sure, this branch is created from the master branch.

Finally, we would like to say that Git-flow helped us to have a good organization and management of the code that follows more of the companies nowadays.

4.3 Scrumbat

Scrumban is the methodology of work that we have followed to develop the application. Scrumban comes from scrumbut plus kanban.

- Scrum: Scrum is a framework which helps people to approach complex problems while they deliver efficient and creative software with the maximum value. Scrum has to be: light, understandable and difficult to dominate.
- Scrumbut: Scrumbut, as its name indicates, it is a methodology based on Scrum but with some buts. So that in the team we work with scrum with some buts. The buts come when we have a plan for two weeks and then, any unforeseen happens such as underestimate the difficulty of the task or resolve some bug that we have not contemplated, then a review and a readjustment of the planning is done.
- Kanban: Kanban is an information system which controls the necessary product fabrication in quantity and time. The word comes from the Japanese and wants to say something similar to "visual cards".

  To manage these visual cards, we use Trello which is the most famous kanbas digital platform. Besides, we use this platform by default to the management of Scrumban. In order to do this, we had to learn the basic functioning of Trello through a tutorial that it is offered through the own platform, be registered in the platform, create a new board for our project called Psymood, know the procedure to fill a card and the use of each of the columns which is going to be explained later in deep.

The previews paragraphs define Scrumban as the composition of scrum plus scrumbut plus kanban trying to do use of the virtues of each one and correcting the defects in a methodology a bit more complex but more powerful.

While this methodology is not perfect, it is the most used for the software development teams in the world because it has more points in favor that against.

- Cons:
  - It requires that every member of the team knows the methodology and follow it.

  - It also requires daily meetings for its management which can be ponderous but they are made to avoid subsequent interruptions and reduce the documentation or emails. In these meetings, each member of the team explains the progress of his/her tasks and any problem which he/she has found so the team can get a solution.
  - Its implementation is difficult at the beginning because each member of the team has to habituate to the methodology, learn it and accept it. Besides, it requires adjustments and continuous improvements to optimize team performance.

- Pros:
  - Improve the efficiency of the team since each person of the team knows what task has to do and it is not necessary to reorganize ones and again the planning.
  - It is flexible in case of a change in the objectives and requirements of the project.
  - It is transparent to the rest of the team, every member of the team know what the other members are doing.
  - Every member of the team works together to get the planning done.
  - Every member of the team is responsible for x tasks and can organize the tasks as he/she wants.
  - It allows seeing the advance of the tasks and any problem that the developer has found.
  - It does not apply to the big teams, more than ten members but our team is composed of two members so it works perfectly.
  - Not only it is a simple, strong and flexible methodology.
  - Moreover, it is a methodology widely use, documented and in continuous progress with a great community that supports it.

As for our point of view, following Srumbat was not as difficult as it could be, we had read a lot about it and we caught the concept immediately. In the beginning, the daily meetings were ponderous since everything was right and each of us knew what to do, however, we understood the importance of realizing these meetings when we started to have problems with some developments since it helped us a lot and we saved a lot of time. Another, thing to highlight is that until you habituate to all the methodology and do everything as the methodology determines to elapse a time. Besides, we highlight the flexibility that gives to you and the great documentation which you can find on the internet. For sure, we recommend it a lot since it helps us to be organized, to be more efficient, to work better and to have the application at the time and it is not difficult at all.

4.4 Trello

As we have said, Trello [32] is the most famous kanban [33] digital platform. We have used this platform to manage the visual cards of our board called Psymood that has helped us to follow our methodology scrumbat. In this section, we are going to explain the columns of our board:

- To Do: It contains the tasks to develop during two weeks which is the period that lasts each planning. These tasks are ordered by priority and have assigned one developer. Each task should be correctly specified and be easy to understand. Besides, they should contain tags which specify the type of the task (feature, bug or maintenance), the involved areas (backend or frontend) and the date of the finalization of the task.
- Doing: It contains the tasks in developing in the actual planning. Each task has one developer and to move the task to the done column, the task has to be completed, documented and tested. For sure, the task has to be completed at the time except any unforeseen such as an unexpected bug or an unexpected problem during the development.
- Blocked: It contains the tasks in which the developer can not work for some reason. Each task has a commentary that explains why it is in this section.
- Done: It contains the tasks done in this planning. Each task in this section has to be done, documented and tested by the developer in charge of it. Besides, the branch where the task is developed had to be merged to the develop branch.

These are the columns that we have used to develop our app. Besides, once planning it is done, all the tasks in the section done are archived and a new release with a new version is created that later will be merged with the master branch.

Finally, we would like to say that we learn how to use this platform was exciting. It helped us a lot to achieve our objectives and followed the methodology chosen. It is versatile and flexible, allowing you to create more or fewer columns and have an entire perspective of how the planning is going. Besides, it is straightforward to use; the only disadvantage that we can highlight it is not having met this platform before.

4.5 Github repository

Github [30] is a cloud service that helps developers with project management and code version control. It works as a social network platform designed for programmers, allowing them to work collaboratively with people around the world, plan projects and track work.

Projects are organized in repositories. A repository is a directory where files such as code, images, audios and everythin'g related to app development are stored. It can be configured publicly or privately, depending on the needs of the user, and include a distri-

bution license.

A Github public repository [31] has been created for this project. To access the repository, click on the following link: Github repository of Psymood. It contains the following information:

- The source code of the mobile application
- A detailed description of the prerequisites and installation of the project
- The visual resources of the app, such as images and animations
- Distribution license.

User data is private and cannot be published. The user is the only one who can distribute his information under his responsibility.

Chapter 5

Use cases

In this chapter, we will deal with the use cases of the application. A use case is a description of the actions of a system from the user's point of view. It is a handy tool because it allows you to check if the system requirements are correct or incorrect from the user's point of view.

Use cases can be modelled using use diagrams, which represent the functionality and interaction of the actors, any user, with the system.

Suppose we have a medical system to treat mood disorders. If we made a use case diagram, the result would be similar to the following (see Figure 5.1):

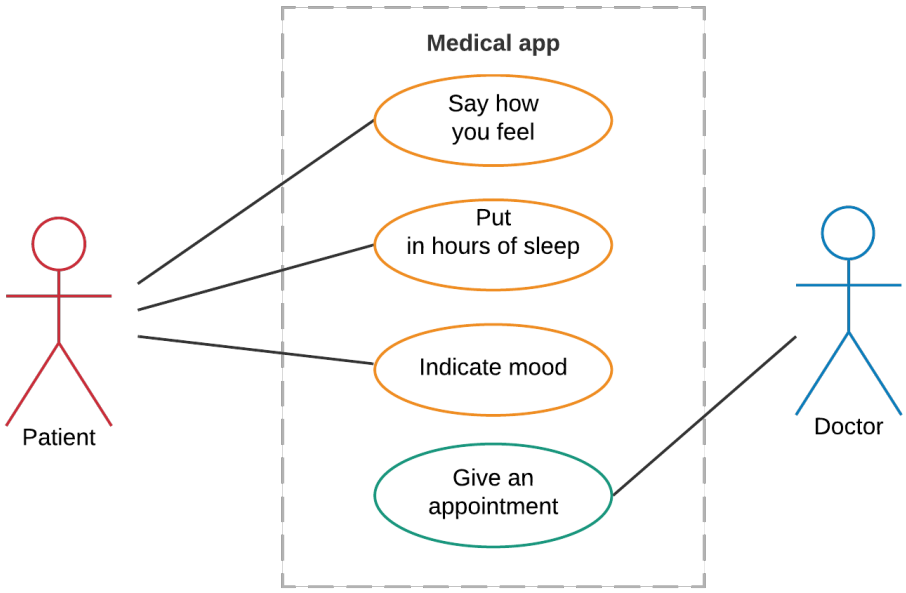

Figure 5.1: General diagram

Use cases are presented with the oval. The label in the figure indicates the function as giving an appointment. In this example, we have two actors, the patient and the doctor. Depending on the system, we will have one, two or more actors. Finally, the system is

indicated by the large square. To organize the use cases, let's group them in functional modules: user module, session module and data module.

5.1 User module

The user module is all the functionalities related to the management of the user account. To be able to access the system, it is necessary to log in with an account. Only registered users can access Psymood. As illustrated in Figure 5.2, a patient can create a new account, update user information, view changes, and delete the account. In the following tables, we will explain in more detail the possible actions of the patients.

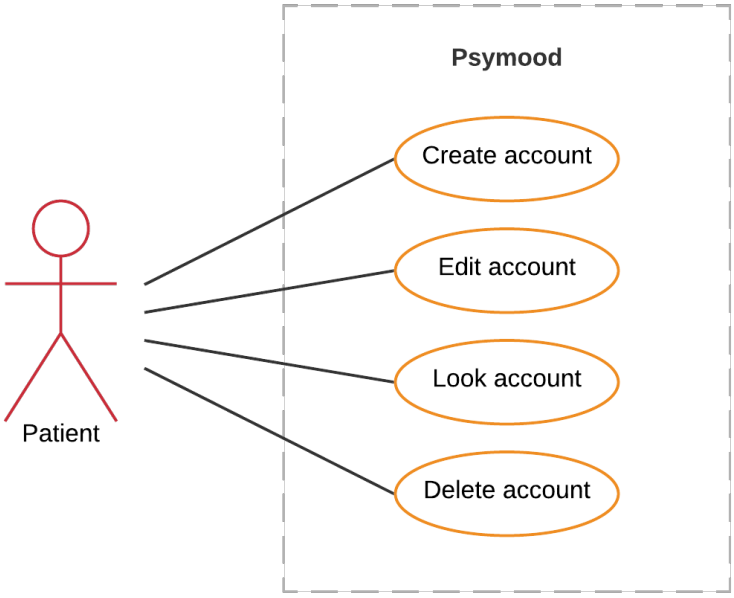

Figure 5.2: Module 1: user account

Create account	
Function	Add a new user account of system.
Priority	High
Input data	Required: - Name, email, password, password confirmation. Optional: - Profile image.
Output data	A message confirming the registration.
Precondition	The email is not already registered.
Post condition	The user is added to the user database.
Description	The registration view shows a form with four fields: name, email, password, and confirmation password. The user enters the data and presses the registration button. It is checked that the data does not contain strange characters and that the password has a minimum of 6 characters, the email is not registered. If all conditions are met, the user is created and automatically logs into the application, displaying the main screen. If these conditions are not met, an issue message is displayed.

Table 5.1: Create user account

Edit account	
Function	Update the user account.
Priority	Medium
Input data	- User profile photo - Username - User gender - The birthday
Output data	Show updated user data.
Precondition	Existing user in the database and with session initiated. The user has to be in the section of his profile.
Post condition	The user is updated to the user database.
Description	The user in the profile view enters the new information in the corresponding fields and confirms the changes by clicking on a save button. If the data entered is correct, the changes are saved in the database, and if they are not correct, this is indicated by a message warning of the problem. All the fields can be edited except for the email. The email is the user's identifier in the system, an identifier unique, for this reason, is not editable. The system accepts letters, numbers and accents, including the ñ, but no other special characters or blank spaces. Changes can be viewed after saving.

Table 5.2: Edit user account

Look account	
Function	Look the user data
Priority	High
Input data	Access the profile
Output data	The user data
Precondition	The user exists in the database and is logged in.
Post condition	The system displays all user data on the screen.
Description	When accessing the profile, the data is displayed. For example, profile picture, name or email. If any field is empty, not shown.

Table 5.3: Look user account

Delete account	
Function	Delete the user account from the system.
Priority	Low
Input data	Password
Output data	A confirmation message that user has been deleted.
Precondition	The user has logged in to the application.
Post condition	The user is removed from the user database.
Description	In the user profile, click the delete account button. The system asks for the password to confirm the action. If it is correct, the user account is deleted; otherwise, the password is requested again. When an account is deleted, the user session is closed. The system takes the user from the login screen.

Table 5.4: Delete user account

5.2 Session module

To be able to use the system, it is necessary to log in. When the user does it, he is identifying himself and allowing the system to configure the environment. In this way, we can ensure that all information collected belongs to him and that he is aware of this fact.

Data security is one of Pysmood's priorities. This is an aspect which deals with protecting data against unauthorised access and possible corruption throughout its life cycle. According to the RGPD [34]:

> *"Personal information is considered to be any information about a natural person whose identity can be determined, directly or indirectly, in particular by means of an identifier, such as a name, an identification number, location data, an online identifier or one or several elements of the physical, physiological, genetic, psychic, economic, cultural or social identity of said person."*

Implementing security procedures such as data encryption and essential management practices that help protect data will help us to safeguard information. Through the validated procedures and sessions, we can ensure compliance with the above.

Finally, if the user wants to end the interaction with the system, he can log out from the profile settings view.

The session module houses these functionalities (see Figure 5.3).

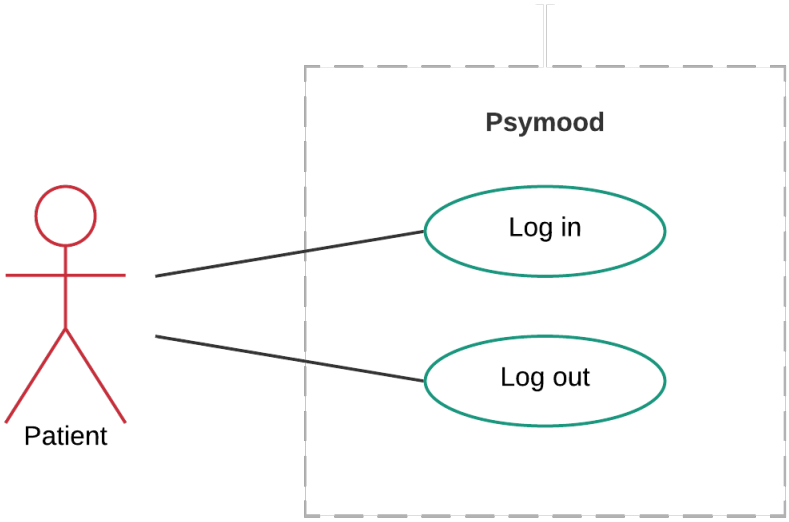

Figure 5.3: Module 2: session management

The use cases associated with this module are listed below:

Log in	
Actor	User
Function	Login system.
Priority	High
Input data	Email and password
Output data	The message of correct login.
Precondition	The user email and password exist in the database and are corrects.
Post condition	Save the session.
Description	In login screen, input the username/email and password fields. If are correct, will be redirected at the main view. If the data is not correct, the error message is displayed, specifying the error in the field or fields.

Table 5.5: Log in system

Log out	
Actor	User
Function	Log out system.
Priority	High
Input data	Click in exit button.
Output data	The message of the session has closed.
Precondition	The user exists in the database and is logged in.
Post condition	Close the session.
Description	In the profile screen, click on the logout button. The system automatically logs out the user and displays the log in screen.

Table 5.6: Log out system

5.3 Data module

The data module covers all functions for collecting mood information. Taking into account the types of data to be collected: daily selections, voice audios and status indicators, the way they will be recorded, stored and uploaded to the system's database must be defined.

Taking a picture is nothing like recording audio or saying how happy you are today. The data we work with is very different from each other so that a unique upload method will be developed for each type of data. Consequently, we will direct much of our efforts to promote the specifics methods of uploading data and the rest of the functions to view or delete data (see Figure 5.4).

From the user's point of view, the only perceivable action is the uploading of his information; the rest go unnoticed because they are performed in the background. Although our goal indeed is to collect the most significant number of daily indicators, we will not allow false nulls, in other words, information uploaded by mistake. We seek to make our information base very reliable. To fix the problem, we are going to implement a system to confirm the sending of data so that the user can discard the data or not.

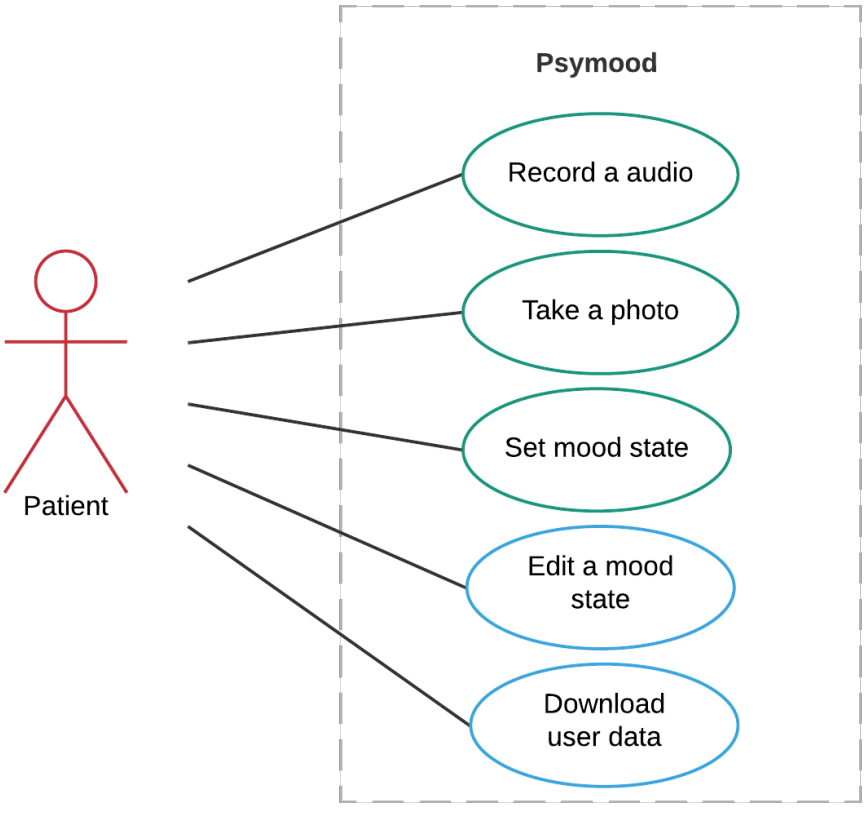

Figure 5.4: Module 3: Module data

The use cases associated with this module are listed below:

Take photo	
Actor	User
Function	Take a photo.
Priority	High
Input data	Click on the camera button.
Output data	A photo.
Precondition	The user exists in the database and is logged in.
Post condition	The photo is added to the respective database and the camer button is enabled again.
Description	To take a photo or selfie, access the camera view. The device's native camera opens automatically, and if the user wants to take a picture, he must click on the button shown. The user can repeat the photo as many times as necessary. If he is satisfied with the result, a modal will be displayed to confirm the file upload. Just click on 'accept' to make an upload. After a brief moment, a message will be presented with the result if everything went well a confirmation, otherwise an error message.

Table 5.7: Take photo

Record audio	
Actor	User
Function	Record a audio.
Priority	High
Input data	Click on the record audio.
Output data	A voice audio.
Precondition	The user exists in the database and is logged in.
Post condition	The audio is added to the respective database and the record audio button is enabled again
Description	Recording audio allows the user to store their voice in a file. To record, it is necessary to access the recording screen and pressed the audio button; then the device will start recording. The user can repeat the recording as many times as needed and listen to the recorded audio in each iteration. If he is satisfied with the result, a modal will be displayed to confirm the load. Before sending the amount, the user is asked once if he wants to act and when he clicks on 'accept'. After a short time, if everything went well, a message is displayed with success and if not, an error message.

Table 5.8: Record audio

Set mood state	
Actor	User
Function	Set all mood states.
Priority	High
Input data	Click on the mood buttons.
Output data	Status indicators.
Precondition	The user exists in the database and is logged in.
Post condition	The indicators are added to the corresponding database and the status buttons are displayed again.
Description	The status indicators are displayed in the form of a form. Each row in the structure corresponds to a status that can be evaluated from low to high. A state can only have one selected value at a time. For example, today's level of fatigue can be low, medium, high or very high, but not all four options. Status levels must be able to be changed quickly, so a fast and efficient status recording system must be developed so that they can be edited at any time.

Table 5.9: Set mood state

Update mood state	
Actor	User
Function	Update a mood state.
Priority	Medium
Input data	Click on the mood button.
Output data	New status indicators.
Precondition	The user exists in the database and is logged in.
Post condition	The indicators are updated to the corresponding database and the status buttons are displayed again.
Description	Editing a state should be a simple task. The user should be able to change the state quickly and without any more effort than pressing another button.

Table 5.10: Update mood state

Download data	
Actor	User
Function	Download the recorded data.
Priority	Low
Input data	Click on the download button.
Output data	A CSV file with structured and organized user data.
Precondition	The user exists in the database and is logged in.
Post condition	The csv file stored on the phone in the system folder on the device.
Description	In the settings view, when the user clicks on the download button, the system generates a CSV file with the user's database records.

Table 5.11: Download user data

Chapter 6

Design

The interface has been designed to be friendly and intuitive. In this way, it allows good interaction with the system making easy the collection and the store of the user data.

This chapter aims to illustrate the process followed to design the interface of the system. First of all, after having clear the objectives, we did some sketches of lower fidelity and then, we did eleven mockups before starting the development of the system.

A mockup is a previous design of a web or application. It is used in the initial design phase to visualize ideas or concepts. Besides, it allows giving a fairly close and real idea of the final system.

In spite of there exists multiple tools to generate mockups such as Marvel App, Wireframe, Mockflow or Cacoo. Each tool offers several possibilities and to decide which one was more suitable for us, we have taken into account several aspects such as learning time, the difficulty of tasks, communication between members and possible results. Taking into account these aspects, we opted for the Marvel App.

Marvel App [35] is a tool to create interactive prototypes from a web platform. This means that it allows working from any device with an internet connection. Besides, it is collaborative and allows members to modify the project at the same time. Last but not least, the learning curve is minimal, obtaining results quickly and it also allows to import designs created by Photoshop and Sketch using plugins, to synchronize with Dropbox or Google Drive, and to upload image files.

Finally, after finishing the mockups, we did the prototype. Prototypes can be adapted to all types of devices and operating systems, just by selecting the function. Within a page you can add areas of interaction, transitions, and gestures, allowing to see the complete functionality as well as the interaction between the different screens of the prototype. Besides, the work can be exposed directly from the application, without the need to make a separate presentation.

In this chapter, you can see the different screens of the prototype and they may make easy to understand the prototyping process.

6.0.1 Login screen

Figure 6.1 is the login screen, the first view of the system. It offers three different ways to access the system. A common login, a login with Facebook and a login with Google. The common login is composed of two editable fields, email and password, and a button to validate the introduced information and access to the system. Besides, at the end of the screen, we can find a sign-up button to access the registration form.

6.0.2 Sign-up screen

Figure 6.2 is the sign-up screen. This screen includes a button with a user image that allows uploading a profile picture, four editable fields, and a button to register into the system. The four editable fields are the email, the full name, the password, and the confirmed password. Besides, we can find at the end of the screen a login button to log into the system.

6.0.3 Home view screen

Figure 6.3 is the home view screen that is shown after the user logs in the system. On this screen, a daily, weekly and monthly bar show the user progress. Besides, at the end of the screen, we can find a menu button that allows navigating through the different functionalities of the system. This menu button is always present in the different views of the system logging into the system.

6.0.4 Settings screen

Figure 6.4 is the settings screen that is shown after the user button of the menu button is pressed. This view offers the user the possibility to change his/her password and to log out of the system.

6.0.5 Camera screen

Figure 6.5 is the camera screen that is shown after the camera button is pressed on the menu button. This screen displays the camera so the user can take a picture. The picture can be a selfie or a normal picture. After that, as it is shown in Figure 6.6, the user can decide between send the picture to the database and store it or take another, pressing the confirmation or the cancel button.

6.0.6 Audio screen

Figure 6.7 is the audio screen that is shown after the button with the paper icon is pressed. This screen includes a random text and a button to record the voice of the user reading the text. After the recording, as it is shown in Figure 6.8, the user can listen to the record and decide between record another or send the record to the database and store it, pressing the confirmation or the cancel button.

6.0.7 Mood states screen

Figure 6.9 is the mood states screen that is shown after the user presses on the circle button which contains a plus symbol of the menu button. This screen allows recording the emotions, the energy and the mental state of the user among others. Each of them has different levels that are going to help to collect information about the daily life of the user.

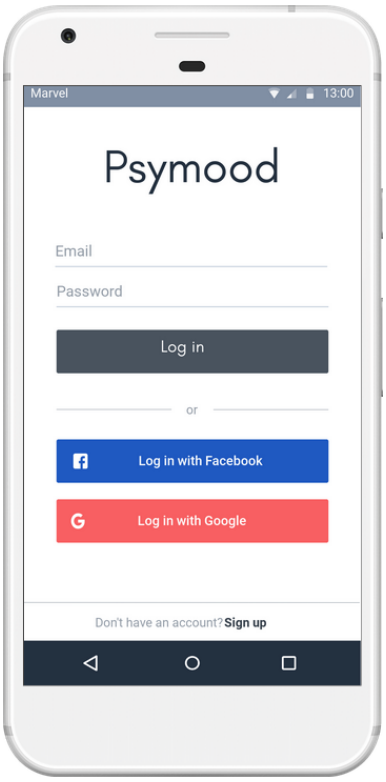

Figure 6.1: Prototype login screen

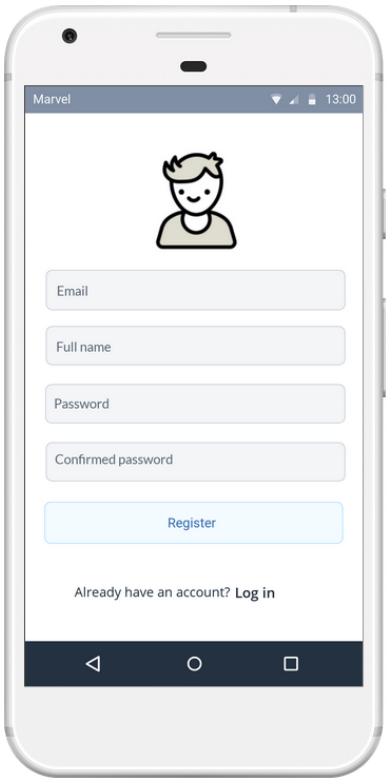

Figure 6.2: Prototype sign-up screen

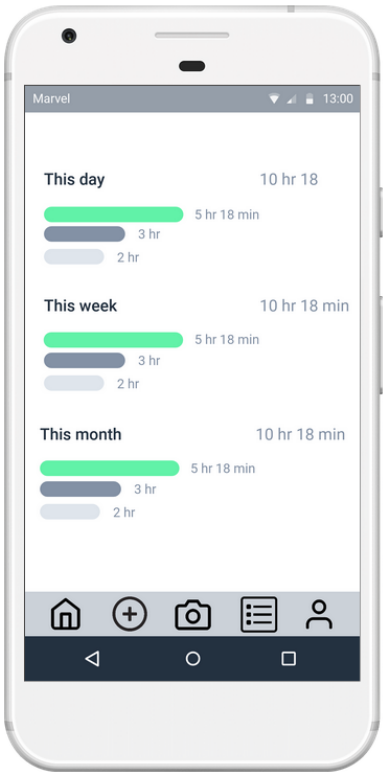

Figure 6.3: Prototype home view screen

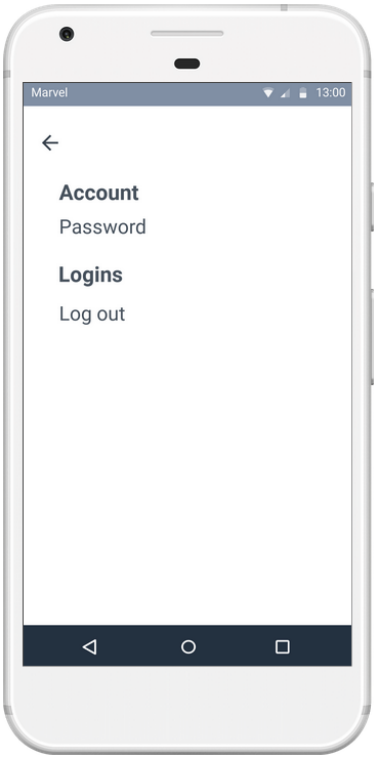

Figure 6.4: Prototype settings screen

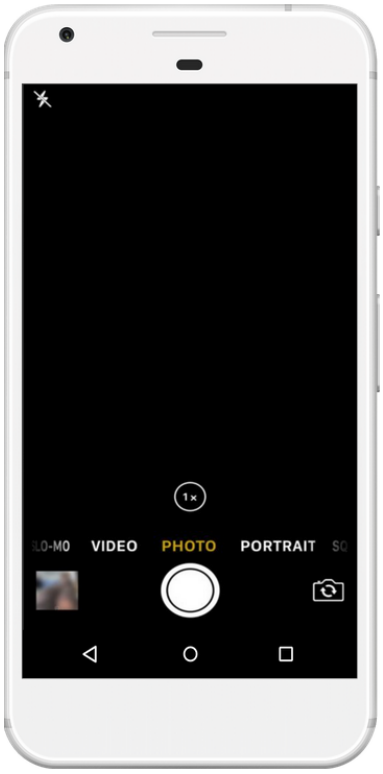

Figure 6.5: Prototype camera screen

Figure 6.6: Prototype picture validation screen

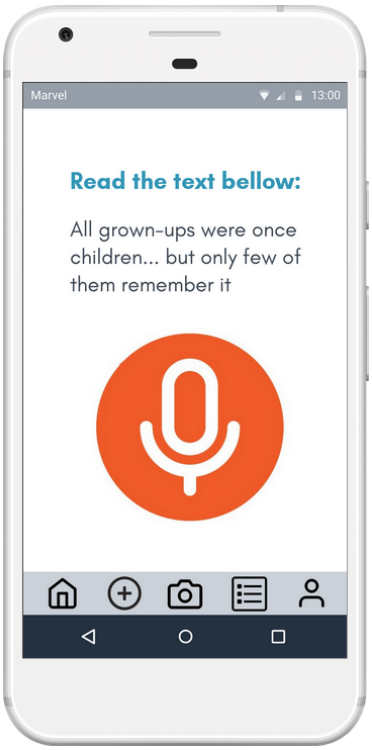

Figure 6.7: Prototype audio screen

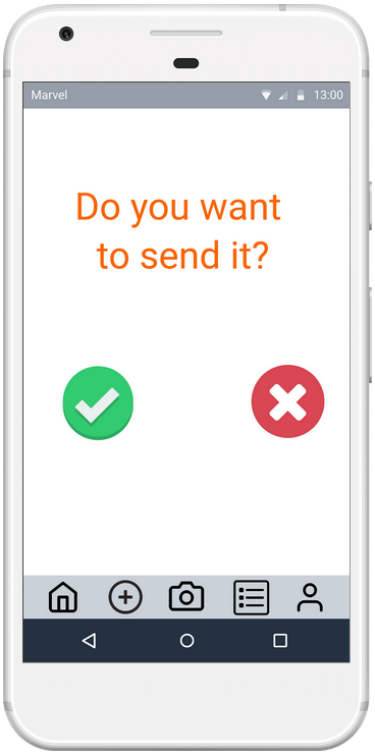

Figure 6.8: Prototype audio validation screen

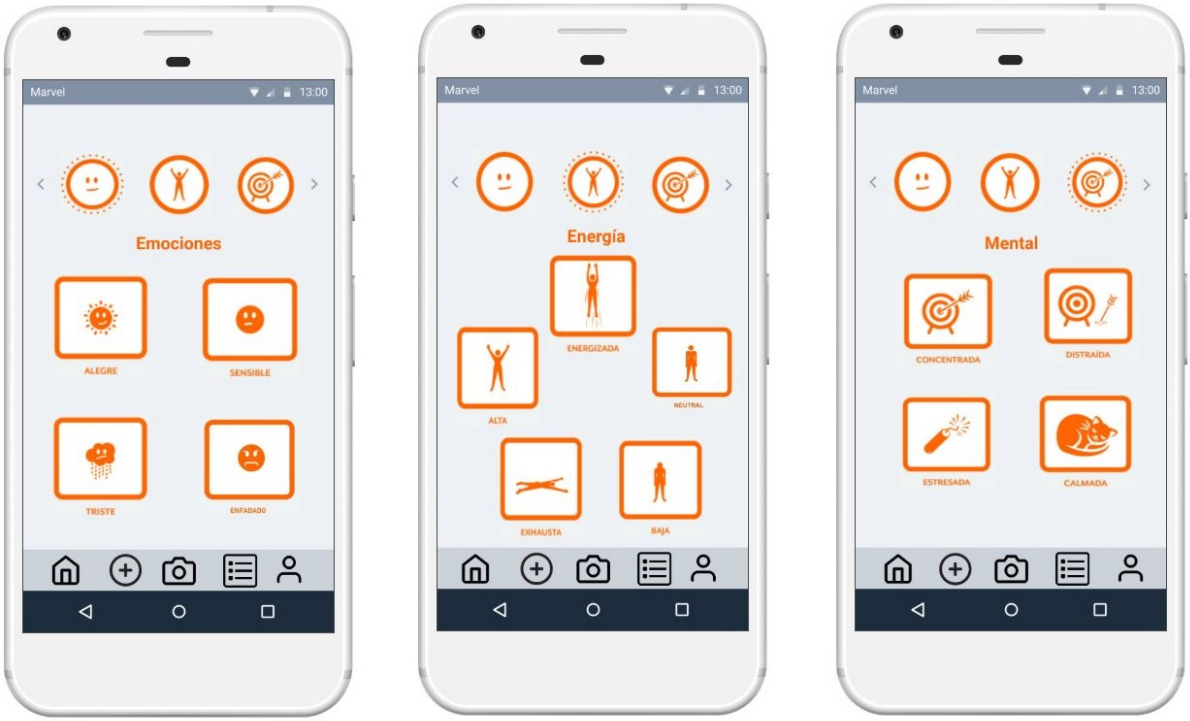

Figure 6.9: Prototype mood states screen

Chapter 7

Firebase

Firebase is a web and mobile development platform in the Google cloud. The platform provides an API to save and synchronize data users in the cloud in real-time.

To better understand what Firebase [36] is, I will explain its three principles. The developer's experience is essential. Ease of use, proper documentation, and intuitive APIs make developers more productive and happy. It can work on diverse platforms. They'll support you in the build to iOS, Web, or Android. Integrate your app where possible. Firebase has one SDK, one console and excellent documentation.

7.1 Features

In this section, we are going to comment on the characteristics of Firebase. For it, we are going to take as an example the Firebase newsletter blog.

7.1.1 Firebase Analytics

Firebase Analytics [37] is a free and unlimited analytics solution for mobile apps. Thanks to Google's experience with Google Analytics [38], it features some capabilities exciting. Focuses on the user and events and gives you information about what your users are doing in your application. You can also see the performance of advertising campaigns on networks, where users come from, and the benefits through a single panel.

It was integrated with other services to obtain a source of information called Audiences. The Aundiencias [39] allows to define groups of users with common attributes and to apply changes, measures or functionalities to them jointly. To understand it better, as we explain the rest of the services, we will indicate which are the Audiences.

7.1.2 Development fast with Messaging, Storage, Config

Google Cloud Messaging [40], cloud-to-device push messaging service is integrated with Firebase. Available for free and for unlimited usage and supports messaging on IOS, Android, and the Web.

Firebase Storage [41] is another essential feature. It is allowing you to store images, videos, and other large files and, upload and download it quickly and safely. It works

with Google Cloud Storage , which gives you massive scalability.

Firebase Remote Config [42] offers immediately upgradeable variables that we can use to adapt and customize your application in execution to offer the best experience to the users. You can enable or disable features or change the appearance without having to publish a new version. It can also target configurations to specific Firebase Analytics audiences [39] so that each of the users has a personalized experience.

7.1.3 Back-end

As for the back-end exists numerous possibilities. We will center in Firebase Realtime Database [44] and Firebase Authentication [43]. Realtime Database stores and synchronizes data with our cloud-based NoSQL database. The data is stored in JSON format and synchronized with all clients in real-time. When the connection is lost, the Firebase Realtime Database SDK causes the data to persist on the disk. When the connection is reestablished, the client device receives the missing changes and synchronizes them with the current state of the server.

Firebase Authentication provides backend and SDK services to authenticate users in your app. It supports authentication through passwords, phone numbers, federated identity providers, such as Google, Facebook and Twitter, and much more.

7.2 Learning Firebase

When we start with the project, we must choose which database we use to store the data. Many ideas came up, SQL-Lite [45], Mongo DB [46], but finally we stayed with Firebase. We didn't know exactly what Firebase was, so the first step was to find information.

On its website, can access great documentation. It is classified by categories, including each of the platforms, so everything is very intuitive, easy to use with lots of examples with real code. It also has a YouTube channel, where we can find tutorials and simple explanations.

To complement the documentation we turn to Stack overflow. Many people with the same uncertainties and different solutions that can be applied.

7.2.1 Prerequisites

Before we start we must have Android Studio installed. Preferably the most recent version. To be integrated into the project, the app must have an API level of 16 (Jelly Bean) or higher. The Gradle [47], compilation system, must have a version greater than 4.1. Finally, a device or emulator must be configured to run the app.

7.2.2 Install

Following the steps in the *"Add Firebase to your Android project guide"* [48], we begin the installation process. There are two options for installation. Adding to your application involves performing tasks both in the Firebase console and in your Android project.

The first step is to create a project that connects to the application. Select Android to start the configuration work and enter the application id. Each Android application has a unique ID, such as com.example.psymood. This ID identifies your application on the device and in the Google Play Store [49]. It is defined in the applicationId property in the module´s app/build.gradle file, as shown here:

```
android{
    defaultConfig {
        applicationId "com.example.psymood"
        mindSdkVersion 23
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
    }
}
```

Certain Google Play services [50] (such as Google Sign-in) require you to provide the SHA-1 of your signing certificate so we can generate with OAuth2 client and API key for your app. Then, we open the terminal in Android Studio and run the keytool utility equipped with Java to get SHA-1 fingerprint.

```
keytool -exportcert -list -v -alias <your-key-name> -keystore
  <path-to-production-keystore>
```

The keytool service prompts you to enter a password for the keystore. The default password for the debug is android. The keytool then prints the fingerprint to the terminal. For example:

```
fingerprint: SHA1: DA:39:A3:EE:5E:6B:4B:0D:32:55:BF:EF
  :95:60:18:90:AF:D8:07:09
```

Then, we copy our fingerprint and paste it into the project we had created in Firebase. This generates the google-services.json file. It is downloaded and added to the android project. For the Firebase platform to be synchronized, the build.gradle must include Google Services plugin.

In the root-level Gradle file (build.gradle), add rules to include the Google Services snap-in. Verify that you also have the Google Maven repository [51]. The result is something like this:

```
buildscript{
    repositories{
        google() //Google's Maven repository
    }
    dependencies{
         //Google Services plugin
        classpath 'com.google.gms:google-services:4.3.1'
    }
}
allprojects{
    //...
    repositories{
        google() //Google's Maven repository
    }
}
```

In the module's Gradle file (application level) (usually app / build.gradle), we must add a line to the end of the file.

```
apply plugin: 'com.android.application'
android{
    //...
}
//Google Play Services
apply plugin:'com.google.gms:google-services'
```

Finally, the only thing left is to add the SDK to the app. We can add any Firebase product to the Android project. We can add Add SDKs for the Firebase products that you want to use in your application. In our application we will use Firebase Realtime Database [44], Firebase Authentication [43] and Cloud Store [40].

Next, we will explain in more detail the SDKs selected and that contribute to our project.

7.3 Firebase Realtime Database

It is a database hosted in the cloud. The data is stored as a JSON file and synchronized in real-time with each connected user. All of your clients share an instance of Realtime Database and automatically receive updates with the latest data.

7.3.1 Implementation

To add a Realtime Database SDK, we must add the corresponding dependency to the application level build.gradle file:

```
implementation 'com.google.firebase:firebase-database:19.0.0'
```

The real-time database provides a language of declarative rules that allows you to define how we should structure the data, how they should be indexed and when their data can be read and written.

7.3.2 Rules

By default, read and write access to the database is restricted, so only authenticated users can read or write data. To begin and be able to perform the first tests, it is necessary to change the configuration of the rules for public access. This makes its database open to anyone, even people who do not use the application, so once the development of the application is finished, the configuration rules must be changed again to restricted access. With a few simple examples, I will illustrate each of the configuration options.

When we create the database, the default rules disable the read and write options in the database. Because of this, the database can only be accessed through the console in Firebase.

```
//These rules do not allow anyone read or write access to your
    database
{
    "rules":{
        "read": false,
        "write": false
    }
}
```

In the development stage, we will use the public rules instead of the default ones. Thus, any user can perform write or read operations depending on the situation. This is very useful because it allows us to quickly unravel the prototype, before starting without configuring Authentication [43]. Once the development is finished we must reconfigure security rules. This step is very important if we are going to publish the app in the Play Store [49].

```
//These rules give anyone even people who are not users of
   your app.
//read and write access to your database
{
    "rules":{
        ".read": true,
        "write": true
```

```
    }
}
```

When we implement the Firebase Authentication [43], we can develop more elaborate rules. For example, a rule that gives each user a node (memory branch) from which they can read and write. Through Authentication, we can obtain a unique ID that we will use to general the node, as seen in the example.

```
//These rules grant access to a node matching th authenticated
//user's ID from the Firebase auth token
{
    "rules": {
        "users": {
            "$uid": {
                ".read": "$uid === auth.uid",
                ".write": "$uid === auth.uid"
            }
        }
    }
}
```

7.3.3 Development

Once the security rules are configured, it is time to create an instance of the database and see how it works. To verify is that the service has been established correctly, we will try to write to the database.

In the main application, we must instantiate the database. An instance is an executable that allows you to manage the database. The database is structured in the form of a JSON object tree. Unlike SQL databases, there are no tables or records. When we add information to JSON objects, they become nodes. In order to edit it, it is necessary to create a reference. We can consider that a reference is a pointer to a node in the cloud.

Needless to say, let's write Hello, World! We call the message reference and assign the desired value. If the reference does not exist, the node will be created automatically, and we can work with it. Let us look at an example.

```
//Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");

myRef.setValue("Eii, hello World!!!!");
```

This way, we can save different types of data including Java objects. When you save an object, the attributes of this object become new nodes.

Now, let's imagine that the app data must be updated in real-time so that the user always has the latest news, we must add a ValueEventListener to our reference. The event consists of two methods, the most important OnDataChange, which will be activated whenever there is a change in the reference. In this way, we can show the new information to the user.

```
//Read from the database
myRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        //This method is called once with the initial value
           and again whenever data at this location is updated
        String value = dataSnapshot.getValue(String.class);
        Log.d(TAG, "Value is:" + value);
    }
}
```

7.4 Firebase Authentication

In most applications, identifying the user is one of the most important tasks. It allows you to provide a personalized experience on all the devices you use.

For that, Firebase Authentication [43] provides us with backend services and an SDK. To define the access method, access the confirmation console, authentication section, as can be seen in Figure 7.1. Authentication supports email and password, through Gmail, Facebook, Github [30] and many more. The method we use in the application is that of accounts based on email and password, as it is one of the most common and at the same time simple to use. In the authentication sale, we will select the " Email / password " method and save the configuration. After a few moments, the system updates the configuration.

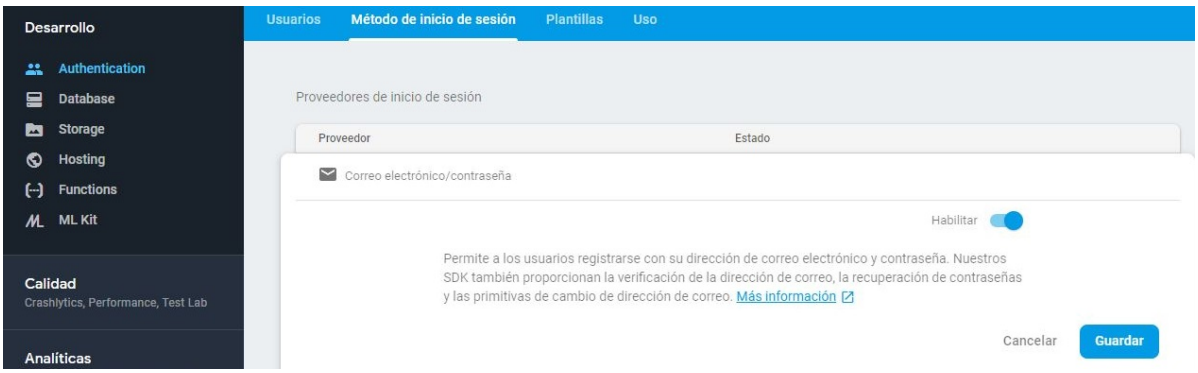

Figure 7.1: Hability view email from Firebase console

7.4.1 Implementation

In the same way as Database Realtime [44], we must add the dependency on the build.gradle file. Next, we synchronize the project and wait for the dependencies to be updated to use the Authentication SDK. All the dependencies must use the same version because it can generate incompatibility errors when synchronizing the application with Firebase. If a library has a different version, just change the version number and synchronize again.

```
implementation 'com.google.firebase:firebase-auth:18.1.0'
```

7.4.2 Development

For a user to access the application, they must have a registered account with which to log in. Using the authentication SDK we can develop a method for the user to create an account. The first thing is to create an instance of a FirebaseAuth object in the corresponding activity. In our case, they are LoginActivity and RegisterActivity.

```
private FirebaseAuth mAuth;
//Initialize Firebase Auth
mAuth = FirebaseAuth.getInstance();

public void onStart() {
    super.onStart();
    //Check if user is signed in (non-null) and update UI
       accordingly.
    FirebaseUser currentUser = mAuth.getCurrentUser();
    updateUI(currentUser);
}
```

When the user enters the application, it is verified and has previously accessed. If this is the case, enter directly. For a new user, a registration form is displayed in the application. The necessary checks must be carried out to create a new account, such as verifying that there is no empty field, that foreign characters have not been entered or that the passwords match. If all the data is correct, we will use the instance to create a new account. We call the function createUserWithEmailAndPassword, which returns a task object, which contains information, to identify whether everything went well or could not be created.

```
mAuth.createUserWithEmailAndPassword(email, password).
    addOnCompleteListener(this, new OnCompleteListener<
    AuthResult>() {
     @Override
     public void onComplete(@NonNull Task<AuthResult> task) {
         if (task.isSuccessful()) {
             //Sign in success, update UI with the signed-in
                user's information
```

```
6                 Log.d(TAG, "createUserWithEmail:success");
7                 FirebaseUser user = mAuth.getCurrentUser();
8                 updateUI(user);
9             }else {
10                /If sign in fails, display a message to the
                      developer.
11                Log.d(TAG, "createUserWithEmail:failure", task.
                      getException());
12            }
13        });
14    }
```

As we can see, the implementation is simple and easy to understand. With a simple method, we were able to create a new account in the database. If all went well, the user enters the application and a call is made with the user's information. To get the data, simply call the getCurrentUser method. In this way we can use all user information in any part of the application, simplifying the logic of our software.

When creating a new user, a node is generated in the authentication bucket. From the Firebase console, in the Authentication window, the new node can be displayed. If the user's email is "dariogal@ucm.es" and the password "dario1234", in the console we will see the registration of this information (see Figure 7.2). The password is not visible for security reasons, which makes the system much more robust. An ID is also generated, unique within the Firebase platform that identifies the user. We can use the ID to search for the user in any other service if the need for your data. Also as an identifier in Database Realtime, we ensure that each user has a unique identifier.

Figure 7.2: Authentication assistance at Firebase

Similarly, if the user already has an account and wants to access the app, it is necessary to check the status of the session. If the session is open, you will enter, otherwise, you must fill in the login form. Similarly, we will call the singInWithEmailAndPassword method to check if the user is registered in the database, returning a Task object if everything went correctly.

Finally, if we want to close the app session, we will call the signOut() method.

7.5 Cloud Storage

It is a free object storage service. Powerful, simple and compact, designed to load and download heavy files from apps to Firebase, regardless of connection quality.It is robust, this means that if there is a network crash while loading or unloading, when the connection returns, the service continues from the point of interruption, saving the user's time and bandwidth. It can be used to store images, videos, audios and other types of content.

Great scalability If our application goes viral and we need more space, Firebase cloud is designed to scale to exabytes. This way it is easy for the prototype to produce our app.

Before you can use Cloud Storage SDK, it is necessary to create a default repository using the Firebase console. In it, the files will be stored. As shown in Figure 7.3, access rules are also requested and, as Database Realtime, and depending on the situation, one or the other will apply. For the development of the application, we define a rule that only allows access to authenticated users. And this is because in the previous chapter we already manage the Authentication and it would not be necessary to declare additional rules.

Figure 7.3: Access rules in Cloud Storage

7.5.1 Implementation

Adding Cloud Storage [40] is simple. Similar to the other Firebase services, we must synchronize the dependency in the Gradle file.

```
implementation 'com.google.firebase:firebase-storage:18.0.0'
```

The first step in accessing the bucket is to create an instance of FirebaseStorage. To access the files, we must create a reference. Calling the FirebaseStorage method with its getReference method we will obtain the bucket reference. We can create references to a lower level of the object tree, similar to the directory structure of any operating system. Let's see an example:

We are going to create an image directory where we save the users' photos. Calling the child method from the reference we look for the folder if it does not exist it creates it and

returns a reference. With it, we can access the directory, either to upload or download files. If we want the reference of a specific file, we can specify the exact path in the child method and it will be returned.

```
1 //Create a child reference imagesRef now points to "images"
2 StorageReference imagesRef = storageRef.child("images);
3
4 //Child references can also take paths spaceRef now points to
    "image/s.jpg"
5 StorageReference spaceRef = storageRef.child("images/user1.jpg
    ");
```

We can also navigate through references. Using the getParent() and getRoot() methods we can go to the highest levels of the file hierarchy. getParent ascends one level and getRoot ascends to the upper level. To fall, we use the child() methods.

7.6 Design

In the previous sections, we have explained each of the services that we will use in the project. From its implementation to a small development. But our database does not work as three separate services, but as a set. Each service is interconnected with the rest from the main axis, Database Realtime.

I had no experience with NoSQL databases. I had heard about MongoDB [46], Firebase sounded like a blog, but nothing more. My experience in databases has been doing projects in the faculty with MYSQL [52] or Oracle [53]. The first idea that came to mind was: I will draw up a diagram with the attributes of the users and the possible relationships to obtain the tables in the database. Bad idea, NoSQL databases are organized differently. Then I decided to look in the Firebase documentation, to better understand how it works. With simple examples and some videos I was able to capture the NoSQL concept applied to Database Realtime [44].

Creating the structure of the database correctly requires a lot of foresight. The most important thing is to plan how the data will be saved and recovered so that the process is as simple as possible.

7.7 Data types

The first step is to define the types of data to identify. For example, a user will have a name, email, a profile picture. With this exercise, we can begin to sketch the final database. Let's look it.

ID a unique string that identifies the user. It is generated when an account is registered.
emailUser String for the user's email.
nameUser String for the user´s name.

profileUser String containing a pointer of the user profile photo. The photo is stored in Cloud Storage.

stateUser Strings array with user states. The possible states are activity, concentration, emotions and energy. Each can provide a range of values from very little to much.

emotions A string that indicates daily user emotions. The possible values are sad, dissatisfied, normal, satisfied and happy.

energy A string that indicates daily user emotions. The possible values are exhausted, low, normal, high, energized.

concentration A string that indicates daily user concentration. The possible values that it saves are insufficient, low, normal, sufficient and much.

hoursSleep A int that indicates number of number of hours of sleep. The possible values that it saves 0 at 12 hours or more.

qualitySleep A string that indicates the quality of the user's sleep. The possible values are bad, insufficient, normal, good, excellent.

cigarette It's an int that indicates the number of cigarettes consumed during the day.

alcohol It's an int that indicates the amount of alcohol consumed. The possible values are from 1 to 5, 5 is a large amount of alcohol.

drug It's an int that indicates the amount of drug consumed. The possible values are 1 to 5, 1 is a mild amount, and 5 is a large amount of the drug.

audioUser A strings array.

audiosUser Item A string with the audio pointer that the user has recorded in the application. Its associated key is the hour and day of the year in which it was recorded.

dailyPhotosUser A strings array.

dailyPhotosUser Item A string with the daily photo pointer that the user has recorded in the application. Its associated key is the hour and day of the year in which it was recorded.

daily_user_photos Image with extension jpg, which the user uploads daily. The name of the image is generated with the date and time it was taken.

profile_user_photos User profile image. Its extension is jpg. The name of the image is generated with the date and time it was taken.

daily_user_audios Audio with extension 3gp. The name of the audio is generated with the date and time it was taken.

Once defined, we will build the structure of the Database and cloud Storage.

7.7.1 Data structure: JSON tree

To start working with the tree we have to define the root node of the tree (InfoUser). From this node, we will define new nodes. Each node has an associated key, which is the node name, and the data it saves. To distinguish each user's node, we will use the authentication ID as the associated key. In this way, we ensure that the nodes are unique and not overwritten.It is recommended that we avoid data nesting for the structure. Let us look:

- **Avoid nesting:** Firebase supports a maximum of 32 levels so we can think that this should be our structure. However, when we extract data from a location, we also recover the secondary nodes. Also, when you grant someone access to read or

write to a node, you also grant access to all children. Therefore, the idea is that the structure is as simple as possible.
- **Compact structures:** If the data is divided into independent routes, they can be downloaded into separate calls as needed. In this way, we will only download the necessary bytes for each user. This allows you to quickly obtain the data.
- **Scalable data:** When working with apps, it is better to download a subset of a list, if the list contains thousands of records. If we need a method to obtain a list of users with a common characteristic, we can define user indexes on a new node. This is likely to duplicate some data since the relationships must be stored both in the user record, and in the index record, but thanks to redundancy, the relationships become bi-directional, allowing users to obtain information quickly and efficiently.

Next, we will see the first sketch of the database. The root of the tree is InfoUser, with three branches. Each branch is a node with the user ID as the associated key. The data is two Strings, one contains the email of the other and the other the name of the user (see Figure 7.4).

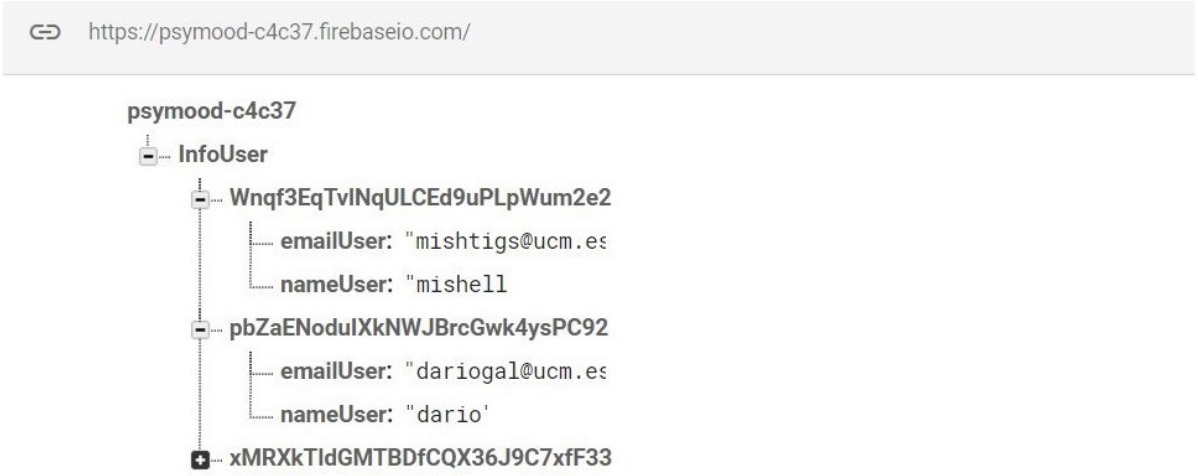

Figure 7.4: Database Infouser structure

7.7.2 Storage structure: Directories

In the app, we can perform various tasks. From uploading a photo, recording audio or register a mood. All of this is valid information that we will use to determine user status. To facilitate this task, we will organize the multimedia files into three groups. Profile images, images were taken daily and voice audios recorded. Each group has a folder, within which are each of the multimedia files of the users. To organize the content, we will form subfolders with the user ID. In this way, we have organized the multimedia content by type and by the user. To extract user information, just know the ID and the type of file you want to extract. Besides, each multimedia file has a URL, which is saved in the Database Realtime node [54]. This redundancy facilitates access to specific files, improving access times, loading and downloading data. The names of the files are generated from the date and time because it has unique names. It also helps us to evaluate user activity in a day, a week, a month or a year. The possibilities are enormous. For example, a sample with the user's photos in the last 5 days that have

been uploaded between 7 p.m. and 11 p.m.

Below is the structure of the daily photo folder that users upload. This structure is similar to the one we can find in the profile user photo directory (see Figure 7.5).

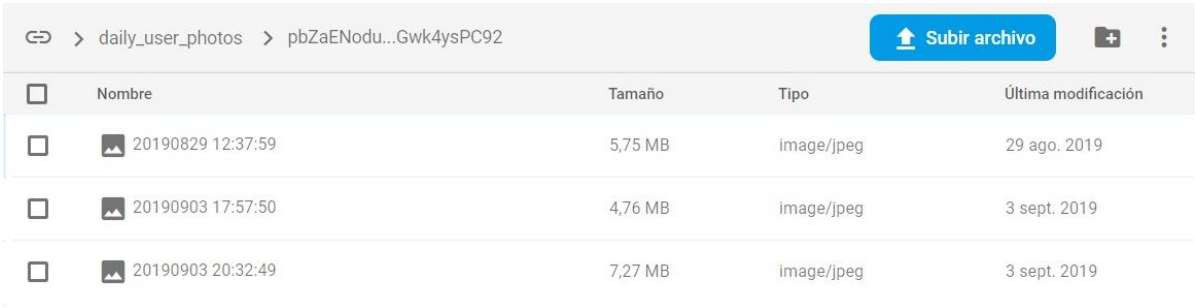

Figure 7.5: Structure of the daily photo folder

As for voice audios, we keep the structure similar to that of the photos, but with a file format is different (see Figure 7.6).

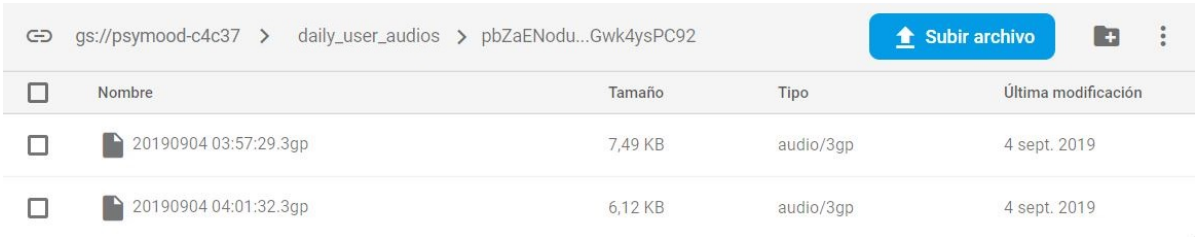

Figure 7.6: Structure of the voice audio folder

Chapter 8

Development with Android

This chapter deals with the implementation of the most relevant Psymood functionalities. But first of all, let's know a little more about Android and its general concepts.

Android is a very versatile operating system mainly used in mobile phones. It is based on the Linux kernel, a free, free of charge and multi-platform operating system with a high capacity of adaptability to every type of devices which give it a high development potential.

Android was initially developed by Android Inc., a company which Google gave financial support and bought in 2005. As founders in Android Inc., we can highlight to Andy Rubin, Richer Miner, Nick Sears, and Chris White.

In 2007, Android was presented together with the foundation of the Open Handset Alliance (hardware, software, and telecommunications consortium of companies). Still, it was really in 2008 when releasing version 1.5 when Android started to expand to the general public, and nowadays, we can consider Android as the most used operating system in the smartphones market as well as the one with the highest growth.

8.1 Learning Android

We had not had any contact with Android before this project, so we spent a considerable amount of time learning about it; however, our experience with java language helped us. Besides, one member of this team took a mobile programming course where he learned a lot about Android and iOS, which helped us a lot to develop this application. We took a lot of advantage of the slides and the examples of the course of which he was provided and the complete official guide of Android. Nevertheless, for more specific topics or doubts, we used other guides, blogs, and forums, primarily.

8.1.1 Prerequisites

To develop applications in Android, it is essential to install Android Studio, which is an IDE based on IntelliJ IDEA, from the company JetBrains and distributed by Google for free. Among its main tools are a Build System Gradle, which allows the production of several versions of the same application, a tool for building user interfaces and a set of

templates that integrate the main code structures used in Android development.

The technical requirements that our computer must meet to support Android Studio are:

- Windows Vista (32 or 64 bit), Windows 7 (32 or 64 bit) or higher.
- Mac OS X 10.8.5 or higher.
- Linux (GNU C (Glibc) 2.1 or higher); 64-bit distributions must be capable of running 32-bit applications.

A minimum of 500 MB of disk space is required for Android Studio installation plus 1.5 GB for the SDK and emulators although it is recommended to have 4GB of free space. Also requires the Java 8 JDK (*Java Development Kit*). For more details, see the following link: Download and system requirement's Android Studio.

8.1.2 Install

The first thing we have to do is download the package with the version of Android Studio we want. Google recommends that we download the latest version, as it has the latest updates. We are going to work with the 3.5 release, which is the most current in 2019.

Once the download is done, we will run the assistant that helps us in the configuration of the IDE. Just run the file android-studio-ide-XXX-mac.dmg or exe (for windows). The assistant opens, and we click on next.
q The android tools are grouped into packages. For the development of Psymood we have installed the following packages: Android SDK Tools which contains most of the development tools, Documentation for Android SDK [58], is the official documentation from Android integrates with the IDE, SDK Platform Android [55] which are the components of the user interface, Google Play Services [50], is a module that provides complementary functionalities such as geolocation sensors, google maps, gmail account management, and Android X Library [56], is a project that is used to implement, model and layout libraries.

8.2 Android General Concepts

We consider that to discuss the implementation of the mobile application it is necessary to profoundly understand the elementary components of an Android project since, without knowledge, even the simplest errors could lead to a tremendous amount of time wasted unsuccessfully looking for mentioned mistakes in the code.

8.2.1 Android Manifest

One of the most important files of every Android project is a file called AndroidManifest.xml. This file, which is in XML format, describes the essential component information, the data entry point, the necessary security permissions, the required libraries, the system environments and the supported hardware.

Any element defined by the application as a library or service must be declared in the

manifest; otherwise, the Android system will not recognize it and will not allow its use. Among the most important ones are:

- Package name and ID of the application: Android compilation tools use the application package name to determine the location of the entities of the code when the project is compiled. When the application is packaged, compilation tools substitute this value for the ID of the application of the Gradle compilation files which is used as the unique identifier of the application in the system and in Google Play.
- Application components: The application components include all the activities, services, emissions receptors, and content providers. Each component should define basic properties such as the name of its Java or Kotlin class.
- Permissions: The permissions needed for the application to access the protected parts of the system or to other applications. Besides, it declares any permission which the other applications need to access to the content of this application.
- Device compatibility: The hardware and software functions that require the application affect the devices which can install the application from Google Play.

The result is somewhat similar to the following code. As we can see, it is built from labels and attributes, and each pair of tags and attributes indicates an element.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android" package="com.example.psymood">
    <uses-permission android:name="android.permission.INTERNET
        " />
    <uses-permission android:name="android.permission.
        RECORD_AUDIO" />
    <uses-permission android:name="android.permission.
        ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.CAMERA"
        />
    <application
        android:testOnly="false"
        android:theme="@style/AppTheme">
        <activity android:name=".Activities.LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.
                    MAIN" />
                <category android:name="android.intent.
                    category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Activities.RegisterActivity"
            />
    </application>
</manifest>
```

8.2.2 Gradle

Gradle file known as build.gradle is another of the most important files in an Android project. Gradle is a compilation system based on Java Virtual Machine that takes all the needed source files to create one compressed APK file. This means that it is able to copy files from a directory to another one before the actual build of the application thus enabling the programmer to have structured topology of the project without the need of manually writing a dedicated script to link these files together. The best of Gradle is:

- It is a plugin that makes easier its updating and its exportation from one project to another.
- It allows to easily reuse code.
- It makes easier the configuration task and personalization of the compilation.
- It allows the easy distribution of code and supports teamwork.
- It manages dependencies in a comfortable and powerful way.
- It allows the compilation from the console which facilitates the compilation task in systems without the development workspace sets.
- It facilitates in an enormous way the creation of different versions of the application.

Finally, it is important to mention that in each Android Studio project, there is lo lo actually two build.gradle files, one is app-dedicated and one is project-dedicated. The content of these two files is very different and it is important to bear in mind and be sure which build.gradle is being modified.

8.2.3 Layout XML files

In Android, an XML layout is a file that defines the different widgets to be used in the user interface and the relations between those widgets and their containers. The benefits of XML layout are:

- XML is a very popular and widely-used format. Hence, a lot of developers are quite comfortable with it.
- It helps to separate the user interface from the logic code. This provides flexibility to change one without affecting much the other.
- Generating XML output is easier than writing direct code, making it easier to have drag-and-drop user interface tools to generate interfaces for android apps.

The user interface in Android is a hierarchy of view groups and views. View groups will be intermediate nodes in the hierarchy and the views will be the terminal nodes. Android provides the following standard layouts: AbsoluteLayout, FrameLayout, LinearLayout, RelativeLayout, TableLayout.

Absolute layout

In absolute layout, we specify the exact coordinates of each widget that we want to place in the user interface, this means that we give the exact X and Y coordinates of each widget. However, we have to bear in mind that absolute layouts are less flexible and

harder to maintain than other layouts without absolute positioning, this is why we have not used it. The following is an example of an absoulte layout:

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent">

     <TextView
          android:layout_x="10px"
          android:layout_y="110px"
          android:text="@string/username"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content" />
     <EditText
          android:layout_x="150px"
          android:layout_y="100px"
          android:width="150px"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content" />
     <TextView
          android:layout_x="10px"
          android:layout_y="160px"
          android:text="@string/password"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content" />
         <EditText
          android:layout_x="150px"
          android:layout_y="150px"
          android:width="150px"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content" />

      <Button
          android:id="@+id/login"
          android:text="@string/login"
          android:layout_x="75px"
          android:layout_y="200px"
          android:width="200px"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content" />
</AbsoluteLayout>
```

The Absolute Layout class was depreciated to API Level 3, so its use is not recommended. Instead, we will use Frame Layout or Relative Layout.

Frame layout

Frame layout is used when you want to show one item on each screen. Using a frame layout, we can have multiple items, but they will be overlapping and only displaying themselves one at a time. We have used this layout to develop our camera interface since we use only one activity where a fragment or another is displayed depends on the option selected. The following is the code of our camera layout:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Fragments.CameraFragment">
        <ImageView
            android:id="@+id/imgFoto"
            android:layout_width="match_parent"
            android:layout_height="480dp"
            android:layout_gravity="center"
            android:background="@color/GreyBackground" />
        <Button
            android:id="@+id/cameraButton"
            android:layout_width="match_parent"
            android:layout_height="480dp"
            android:layout_gravity="center"
            android:text="Camara"
            android:background="@color/redBackground"/>
</FrameLayout>
```

This organization is based on the weight of each of the components. With the previous code, we get a layout that has two widgets inside, a text (TextView) and a camera button (Button), whose proportions are defined to occupy the screen in a 50 per cent.

Linear layout

The Linear layout is a container in which the child elements are placed according to the value of an orientation attribute: android: orientation = "vertical" or android: orientation = "horizontal". Vertically, the widgets are placed in line, one below the other. This is a widely used layout for creating forms. As a result, it is often used in the register, logging or settings view. Horizontally, the elements are placed next to each other, from right to left. This layout is often used to show several items on a slide so that as the view is moved, the user can see new details.

Each linear layout organizes the elements according to the orientation attribute. However, we can also assign a weight to each widget so that a component with a higher weight can take up more space in the row or column of the linear arrangement.

In Psymood, we have used the vertical view to develop the logging activity, and the code is shown below:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/VioletTask"
    android:orientation="vertical"
    android:weightSum="5"
    tools:context=".Activities.RegisterActivity">

    <LinearLayout
        android:id="@+id/linearLayout4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="20dp"
        android:layout_marginEnd="8dp"
        android:weightSum="4">
    </LinearLayout>
    <TextView
        android:layout_marginTop="10dp"
        android:layout_marginBottom="10dp"
        android:id="@+id/textView13"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:fontFamily="@font/muli_bold"
        android:text="@string/psymood"
        android:textAlignment="center"
        android:textColor="@color/White"
        android:textSize="30sp"
    android:textStyle="bold"/>
</LinearLayout>
```

As we can see in the previous code, we have a linear root layout with a vertical orientation that occupies the entire screen. It contains a child linear layout and a text view, which are organized in a division of 80 per cent of the screen for the linear layout and 20 per cent for the text view. This organization is based on the weight of each of the components.

Relative layout

The relative layout is used when we want to specify the position of an element about another element, or about the parent container.

To specify the position with relation to its parent container, we use android:layout_alignParentTop="true" and android:layout_alignParentLeft="true" to align elements to the top-left of the parent container. Then, to align concerning another element we can use the properties android:layout_alignLeft="@+id/otherelement" and android:layout_below="@+id/otherelement".

We have used this layout in our terms and conditions interface. Here, you can find the code:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Activities.TermsAndConditions">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginHorizontal="12sp"
            android:layout_marginTop="4sp">

            <TextView
                android:id="@+id/titlePrivacyPolicy"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textSize="22sp"
                android:fontFamily="@font/muli_bold"
                android:textAlignment="center"
                android:text="Términos y condiciones de
                    servicio"
                />

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:id="@+id/textPrivacyPolicy"
                android:layout_below = "@+id/
```

```
                titlePrivacyPolicy"
34              android:textSize="16sp"
35              android:fontFamily="@font/muli_bold"
36              android:lineSpacingExtra="5sp"
37              android:text="@string/terms_conditions"/>
38          </RelativeLayout>
39      </ScrollView>
40  </RelativeLayout>
```

8.2.4 Activity

An activity is a component of the application that provides a view the client can interact with. A usual application which is only not running in the background if not it is composed of an interface has to have an activity as its main element where all its buttons, text views, image views, and others components are found. In our case, Psymood is composed of two main activities, one is the login where the user access to the application and the other is the navigation activity where the user can navigate through all the possibilities that offer the application such as take a selfie, recording a voice record, etc.

Each activity can init another activity to do other different actions. Each time a new activity start, the previous activity is stopped but the system keeps it in a stack. Then when the user finishes with the actual activity and presses the back button, this activity is pushed from the stack, destroyed and the previous activity is resumed.

All the activities of the application have to be declared in the AndroidManifest.xml so the system can access them, on the contrary, the application will be blocked. To program an Android mobile application, it is necessary to understand the life cycle of the activity at least the basic concepts. An activity has four states:

- **Resumed**: The activity is visible and has user focus.
- **Paused**: Another activity is visible and has the user focus, but this activity is still visible. Paused activities are completely alive however they can be killed by the system in low memory situations.
- **Stopped**: This activity is running in the background and it is not visible. It is alive but it can be killed by the system.
- **Desroyed**: When an activity ends when the finish method is invoked or is destroyed due to lack of memory.

To manage the life cycle of our activity, it is necessary to implement callback methods. Every time an activity changes state, an event is generated that can be captured by these methods and depending on the result we are looking for, we can either deal with a specific action or let the system act by default. The most important callback methods are:

- **onCreate()**: This method is called when the activity is created for the first time. This is where all the set up needs to be done such as create views, bind data to lists, and so on.

- **onStart()**: Indicates that the activity is about to be shown to the user.
- **onResume()**: This method is called just before the activity starts interacting with the user.
- **onPause()**: This method is called when the system is about to start resuming another activity.
- **onStop()**: Indicates that the activity is no longer visible to the user. If the device is has low memory, it is possible that the activity is destroyed.
- **onDestroy()**: This method is called before the activity is destroyed.

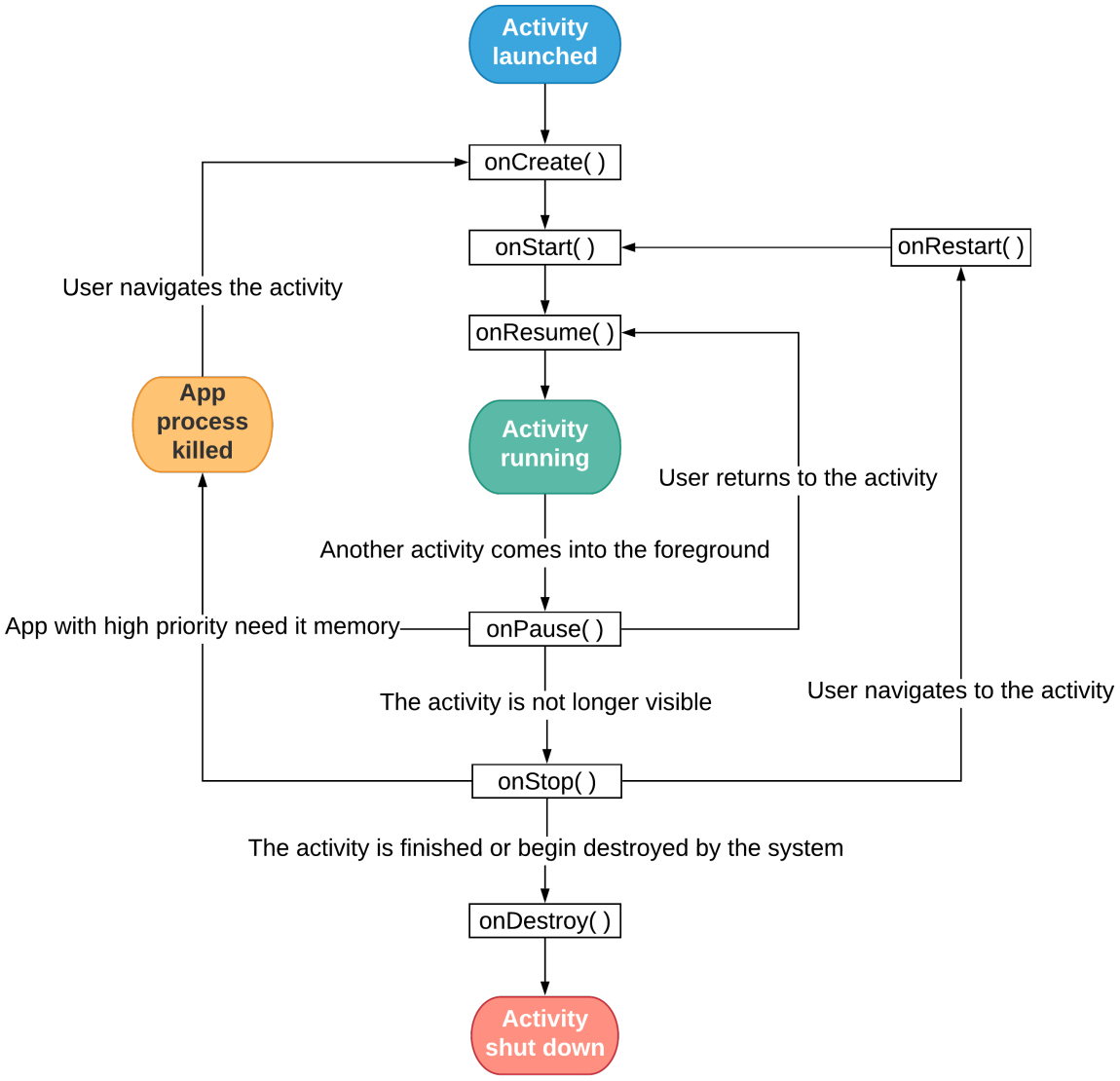

Figure 8.1: Activity life cycle

Our application is composed of the two main activities mentioned before and five more which are the register, the settings, the terms and conditions, the version and the privacy policy activities, which will be explained later in detail.

8.2.5 Fragments

A fragment [?] represents a portion of the user interface in an activity. It is convenient to use fragments when multiple activities shared too much functionality and seemed rebundant since it is more efficient creating a fragment within the first activity than creating a second activity. Besides, the creation of a fragment requires fewer resources than an activity creation and it has a lifecycle on its own within the parent activity. In addition, multiple fragments can be combined in a single activity and they can be added or removed while the activity is running. In our case, Psymood uses fragments because it is more efficient and they allow more dynamic and flexible designs.

To add the fragment to the activity layout, there are two ways:

- Declaring the file inside the activity's layout file: In this case, we can specify layout properties for the fragment as if it were a view.

```
<fragment
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.MapFragment"/>
```

- Programmatically add the fragment to an existing ViewGroup: We can add fragments to our activity layout at any time the activity is running. We simply need to specify a ViewGroup in which to place the fragment. In our case, the ViewGroup is a FrameLayout declared in the activity's layout file and we use the id as a reference. This is how we added the microphone to Psymood.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk
    /res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Fragments.AudioFragment">
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    <TextView
        android:id="@+id/ramdomSentence"
        android:layout_width="301dp"
        android:layout_height="77dp"
        android:layout_marginStart="15dp"
        android:layout_marginTop="4dp"
        android:layout_marginEnd="15dp"
        android:fontFamily="@font/muli_regular"
        android:text="@string/textRecord"
        android:textColor="@color/GreyTitle"
```

```
20          android:textSize="18sp"
21          app:layout_constraintEnd_toEndOf="parent"
22          app:layout_constraintHorizontal_bias="0.521"
23          app:layout_constraintStart_toStartOf="parent" />
24      <com.airbnb.lottie.LottieAnimationView
25          android:id="@+id/animation_view"
26          android:layout_width="350dp"
27          android:layout_height="350dp"
28          android:layout_marginStart="8dp"
29          android:layout_marginTop="8dp"
30          android:layout_marginEnd="8dp"
31          android:layout_marginBottom="8dp"
32          android:scaleType="centerCrop"
33          app:layout_constraintBottom_toBottomOf="parent"
34          app:layout_constraintEnd_toEndOf="parent"
35          app:layout_constraintHorizontal_bias="0.488"
36          app:layout_constraintStart_toStartOf="parent"
37          app:layout_constraintTop_toBottomOf="@+id/
                ramdomSentence"
38          app:layout_constraintVertical_bias="0.268"
39          app:lottie_autoPlay="false"
40          app:lottie_loop="true"
41          app:lottie_rawRes="@raw/record" />
42      </androidx.constraintlayout.widget.ConstraintLayout>
43  </FrameLayout>
```

The fragment can access to its activity instance and use activity methods using getActivity().

As soon as its lifecycle, the most important callback methods are the same as with activity plus one more, onCreateView() method is called when it is time for the fragment to draw its user interface for the first time. Below, you can see a diagram of the lifecycle of a fragment:

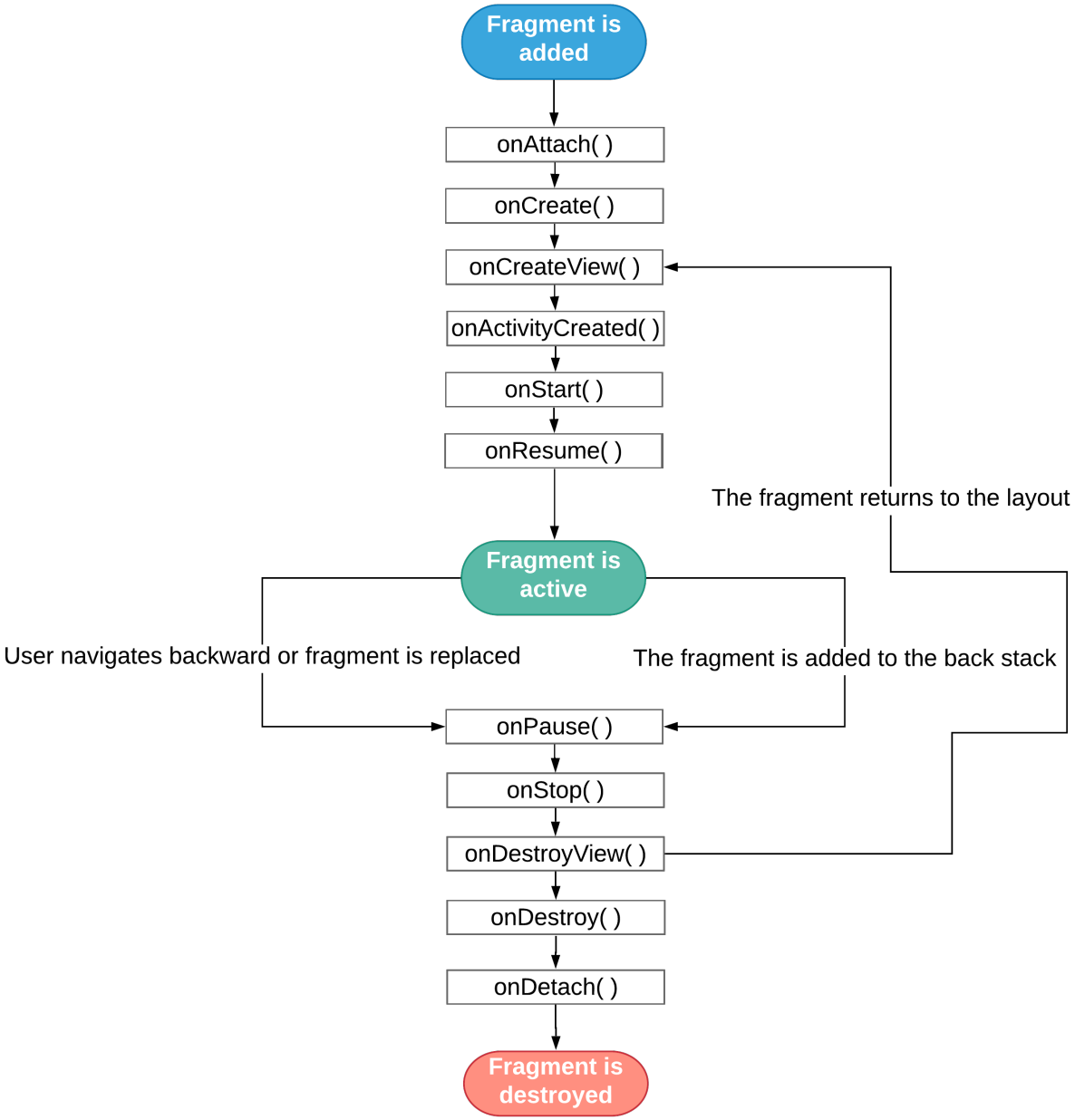

Figure 8.2: Fragment life cycle

8.3 Implementation

This section describes the implemented functionalities and how they are associated with each screen. For the development of each one, we have followed the design elaborated in the chapter of design and mock-up, applying specific changes to improve the experience and the usability of the system. To make the explanation as clear and straightforward as possible, we will only describe the most relevant elements.

8.3.1 Log in view

When a user accesses the Psymood system for the first time, the first screen that is painted is the Log in. In this view, a login form is shown with two email and password

fields and a login button. It also has a link to go to the registration screen, in case you do not have a registered account (see Figure 8.3).

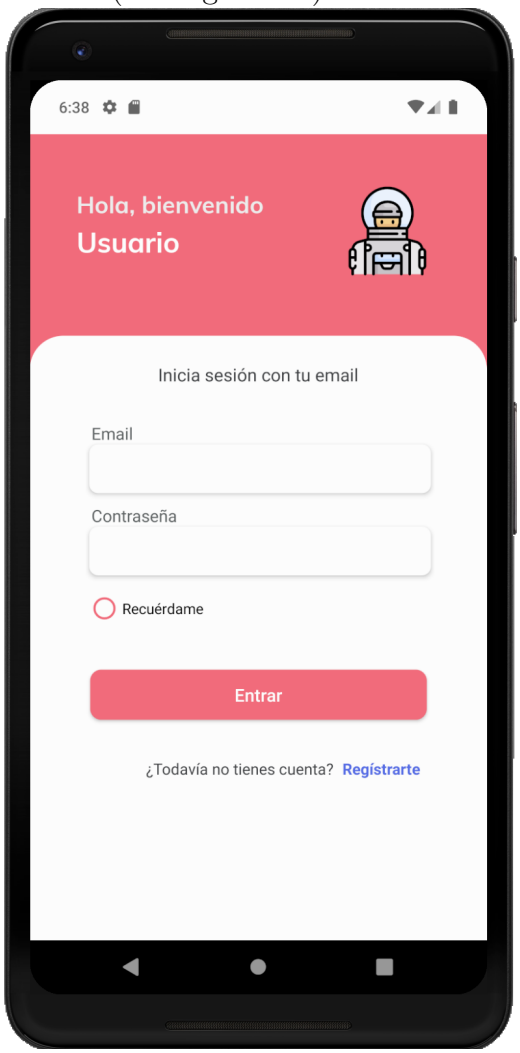

Figure 8.3: Screen of Login view

To be able to log in, you need to enter an email and a password in the respective fields. Both values belong to the patient account, which you have previously registered. The following code shows the implementation of the form logic.

```
1  //Form login code
2  userMail = findViewById(R.id.userMail);
3  userPassword = findViewById(R.id.userPassword);
4  buttonLogin = findViewById(R.id.buttonLogin);
5  ImageView loginPhoto = findViewById(R.id.loginPhoto);
6  linkRegister = findViewById(R.id.linkRegister);
7  rememberMe = findViewById(R.id.buttonRememberMe);
8
9  linkRegister.setOnClickListener(new View.OnClickListener() {
10 @Override
11     public void onClick(View v) {
```

```
12          Intent intent = new Intent(getApplicationContext(),
                RegisterActivity.class);
13          startActivity(intent);
14      }
15  });
16
17  buttonLogin.setOnClickListener(new View.OnClickListener() {
18      @Override
19      public void onClick(View v) {
20          final String mail = userMail.getText().toString();
21          final String password = userPassword.getText().
                toString();
22          if (mail.isEmpty() || password.isEmpty()) {
23                      buttonLogin.setVisibility(View.VISIBLE);
24                      showMessage("Por favor, verifica los
                            campos vacios");
25          } else { signIn(mail, password);}
26      }
27  });
```

Pressing the login button triggers an event that generates an object with the parameters entered and executes a call to the Firebase authentication service: **signIn(email, password)**. The authentication service receives the request, collates and validates the information and if it is correct, creates a session. After a brief moment, the response from the service is received, and if everything has gone well, the session is started. It's possible that Firebase returns an error and depending on the type, we'll execute one action or another. The most common ones are: the email is incorrect, or the password doesn't match. The code for receiving the response is shown below.

```
1  //comprobacion del emial and password
2  private void signIn(final String mail, final String password)
       {
3  mAuth.signInWithEmailAndPassword(mail, password).
       addOnCompleteListener(new OnCompleteListener<AuthResult>() {
4      @Override
5      public void onComplete(@NonNull Task<AuthResult> task) {
6      if (task.isSuccessful()) {
7          updateUI();
8      } else {
9          try {
10             throw Objects.requireNonNull(task.getException());
11         } catch (FirebaseAuthWeakPasswordException e) {
12             userPassword.setError(getString(R.string.
                   error_weak_password));
13         } catch (FirebaseAuthInvalidUserException e) {
14             userMail.setError(getString(R.string.
                   error_invalid_email));
```

```
15            } catch (FirebaseAuthInvalidCredentialsException e) {
16                userPassword.setError(getString(R.string.
                      error_invalid_password));
17            } catch (Exception e) {
18                showMessage(e.getMessage());
19            }
20        }});
21    }
```

8.3.2 Register view

The registration view is used to register an account. It consists of a form with four text fields: name, email, password, password confirmation, a profile picture and a login button, as shown in Figure 8.4. All fields are mandatory except for the image. If the user does not select a profile picture, one is assigned by default.

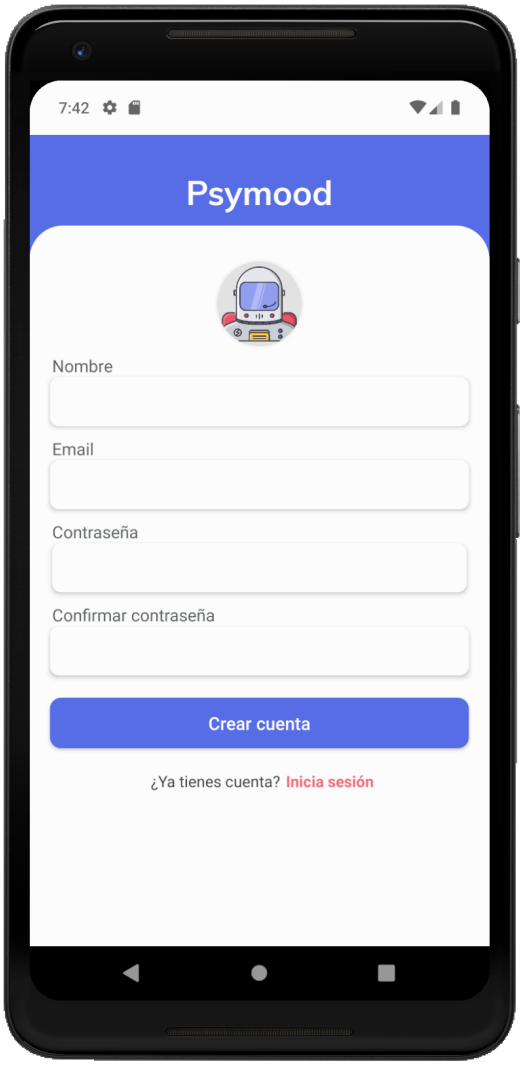

Figure 8.4: Screen of Register view

The data entered must meet some conditions: the email must be in the format of

xxx@xxx.xxx, the password must be at least six characters long and the password confirmation must be the same. In the following code, you can see the data behaviour.

```
if(email.isEmpty() || name.isEmpty() || password.isEmpty() ||
   confirm.isEmpty()){
//Something goes wrong, then we show a message with the error
    showMessage("Por favor, verifica todos los campos");
}else if(!password.equals(confirm)){
    showMessage("Las contraseñas son distintas");
}else{
    //Everithing is ok and all fields is correct, create a
       user account
    createUserAccount(email,name,password);
    }
}
```

Similar to the Login, when we click on the register button, a request is sent to the Firebase Authentication service. A new check is performed, and if all data is correct, a record is created for the user. Once the registration process is finished, the service sends a response with the status of the request.

```
mAuth.createUserWithEmailAndPassword(email,password).
   addOnCompleteListener(this, new OnCompleteListener<
   AuthResult>() {
@Override
public void onComplete(Task<AuthResult> task) {
  if(task.isSuccessful()){
        //user account created successfully
        showMessage("Cuenta creada");
  }else{
    try {
        throw Objects.requireNonNull(task.getException());
    } catch (FirebaseAuthUserCollisionException e){
        //If email already registered.
        userMail.setError(getString(R.string.
           error_email_alredy_exist));
    } catch (FirebaseAuthWeakPasswordException e) {
        //The password is so weak
        userPassword.setError(getString(R.string.
           error_weak_password));
    } catch (FirebaseAuthInvalidCredentialsException e) {
        //If email are in incorrect format
        userPassword.setError(getString(R.string.
           error_invalid_password));
}});
```

As you can see in the code above, if everything went well a confirmation message showMessage("account created") is displayed. Otherwise, a problem message is shown depending on the status returned. For example, if the email entered already exists in Firebase Authentication, the problem is indicated, and a new email is requested.

8.3.3 Main view

The main view or dashboard is the first view shown after logging in. The purpose of this view is to help the user keep track of the tasks performed and the tasks pending, by graphically representing the different system metrics. For this purpose, there is a set of tools also called widgets that will help us in this purpose (see Figures 8.5 and 8.6). We will explain each of them below. Note that we have listed each of the tools to facilitate their location, with values from 1 to 5 and from top to bottom.

Figure 8.5: Screen of Main view

1 At the top, we find a burger button, a native Android widget, which allows us to open the side menu called Drawer. Although it is not an element for monitoring user actions, it facilitates interaction and access to specific tasks. Besides, as it is

a fixed element, it will be painted in all the views so it will always be accessible. In the following code fragment, we can see how the navigation object is initialized.

```
//Menu drawer
  bottomNav.setOnNavigationItemSelectedListener(new
     BottomNavigationView.OnNavigationItemSelectedListener
     () {
  @Override
  public boolean onNavigationItemSelected(MenuItem
     menuItem) {
    return onNavigationBottomItemSelected(menuItem);}});
```

2 A daily progress bar is a bar that represents the percentage of tasks performed during the current day. Used to view the user's daily progress. As you upload data or complete a job, the progress bar updates your information until you reach 100 per cent completion. Below is a snippet of the code where it initializes and updates the percentage.

```
//Progress bar to show the progress day. Look in share
   preferences if there are changes.
progressBarDay = view.findViewById(R.id.progressBarDay);
progressBarDay.setProgress(percentage);
```

3 The carousel is a tool that allows you to visualize the user's tasks. It is composed of cards, each one of a unique colour, that they identifier one task. To visualize all the cards, the carousel tool implements the horizontal slide, and with a small gesture, we can slide to the right or left. Each card has an indicator with the number of tasks remaining. As we perform them, the indicator will decrease until it reaches zero. Besides, if we do on the indicator, it will take us to the pending task.

```
 //RecyclerView and Adpater from Carousel
recyclerViewTask.setHasFixedSize(true);
recyclerViewTask.setLayoutManager(new LinearLayoutManager(
   getContext(), LinearLayoutManager.HORIZONTAL, false));
recyclerViewTask.setAdapter(taskAdapter);
```

4 It is a graph with the user's weekly progress. It has been implemented using the SparkLineLayout library, which allows us to generate linear charts and adapt them to our needs. Each time the user enters a task, the indexes that represent the days of the week are updated. During seven days, you can view the changes in the graphics, and after this time, the graph is restarted. Below is the code snippet where we update the chart ratio index for one day of the week.

```
//Update of the current day graph
values.set(elementOfWeek - 1, taskCompleted);
sparkLineLayout.setData(values);
sparkLineLayout.setSplitLine(true);
float ratio = (float) 0.145 * elementOfWeek;
sparkLineLayout.setSplitLineRatio(ratio);
```

5 At the bottom of the Main view, you will find the lower menu. It is a fixed element on the screen and, like the burger menu, is always present in the view. It takes care of to connect all the displays of the system, allowing to change the view at any time. Each time a menu item is selected, it changes colour, thus highlighting the current view.

The menu consists of four items, each of which identifies a screen: nav_home (main view), nav_add (mood view), nav_camera (camera view), nav_audio (audio view). Every time the user clicks on an element, it launches a selection event (**selectFragmentById( )**), which returns the view associated to the element.

```
private boolean onNavigationBottomItemSelected(MenuItem
    menuItem) {
  Fragment selectFragment = selectFragmentById(menuItem.
      getItemId());
  getSupportFragmentManager().beginTransaction().replace(R
      .id.frame_layout_container, selectFragment).commit();
  return true;
}

private Fragment selectFragmentById(int itemId) {
  Fragment selectFragment = null;
  switch (itemId) {
    case R.id.nav_home:
        selectFragment = new HomeFragment(); break;
    case R.id.nav_add:
        selectFragment = new StateFragment(); break;
    case R.id.nav_camera:
        selectFragment = new CameraFragment(); break;
    case R.id.nav_audio:
        selectFragment = new AudioFragment(); break;
    }
return selectFragment;
}
```

As mentioned above, the chart allows, in a way, to keep track of the uploaded data. To better understand this idea, let's look at the dashboard of a patient who has been using the system for three days (see in Figure 8.6)

Figure 8.6: Screen of weekly progress view

As can be seen, the user started the system on Wednesday. During the first two days, he has made moderate use, uploading almost the same amount of indicators on Wednesday and Thursday. As can be seen on Wednesday, there has been an increase in the data uploaded, which is confirmed by the 100 per cent of tasks performed from the progress bar.

One week after the first data upload, the system empties the weekly progress bar graph, allowing the user to focus on the new week.

Based on this idea, we expect that shortly a monthly and a quarterly table will be implemented, which will allow a higher level of evaluation.

8.3.4 Mood state view

It is a view that contains the mood indicators that can be evaluated in the application. Each one is made up of a set of values represented on cards. For example, the emotion indicator has sadness, dissatisfied, neutral, satisfied and happy. As there are several indicators, and each one has several values associated with it, we have decided to

organize them in a matrix structure in rows and columns. Even so, the screen of a mobile device is not big enough, so we will also apply the slide and horizontal method to be able to visualize all the information.

As shown in Figure 8.7, the user can navigate by scrolling or sliding in different directions.

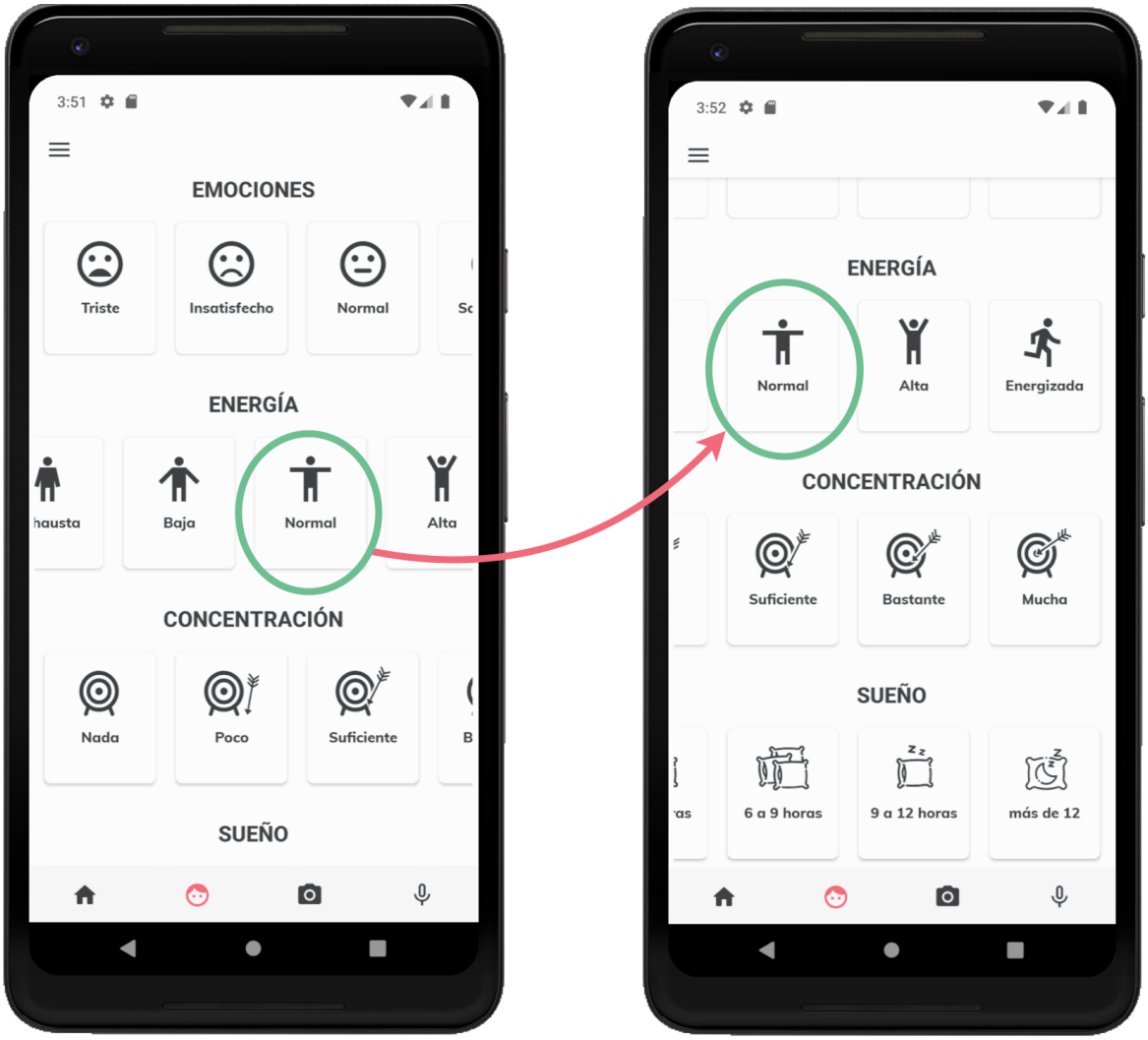

Figure 8.7: Screen of Mood state view

To optimize the resources of the device, we use the View holder object. View holders allow you to generate 'data tables' that only generate those elements that fit on the screen. If we have a row with ten values, of which only four fit on the screen, only four are painted and the rest are not generated until the user scrolls. Below is the initialization of a view holder.

```
1
2 myViewHolder.cardViewState.setOnClickListener(new View.
    OnClickListener() {
3   @Override
4   public void onClick(View v) {
```

```
5       Log.e("MyItemAdapater", "CardView clicked");
6
7       checkHasCellClicked();
8       //CardView state
9       myViewHolder.cardViewState.setCardBackgroundColor(
            ContextCompat.getColor(context,R.color.GreenCheck));
10      myViewHolder.imageViewCheck.setBackgroundResource(R.
            drawable.ic_check_circle);
11  }});
```

To indicate how they feel today, users can click on the status card that comes closest to their current condition. The card is then highlighted to indicate the selection. A user can change his or her mind, so choose another card to change the selection. Each time a flag is marked, the system recognizes that it is a completed task and updates the main view metrics (see Figure 8.8).

Figure 8.8: Screen of selection mood state

8.3.5 Camera View

Taking a selfie or a photo can be another exciting and different way to express our state of mind. To do this, it is necessary to access the camera view. After selecting the camera icon, an event is sent to the camera sensor, indicating that it has to be opened. The initialization and the intent that opens this sensor are displayed below:

```
private void openCameraToTakePhotos() {
  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
  // the Api version is support Android 6
    if (ActivityCompat.checkSelfPermission(getContext(),
       Manifest.permission.CAMERA) != PackageManager.
       PERMISSION_GRANTED) {
       requestPermissions(new String[]{Manifest.permission.
          CAMERA}, MY_CAMERA_PERMISSION_CODE);
    } else {
       dispatchTakePictureIntent();
    }
  }
}
```

In addition to opening the camera, the necessary permissions to access the device sensor are also requested. As the Android documentation specifies, every time an application, software or computer system tries to access the physical device; the user must allow this access.

Following the instructions, we will request access to the sensors used and the storage memory of the phone. If so, we can use a camera. Otherwise, the operating system will reject the request, and the sensor cannot be accessed. Only in this situation and if the user wishes to take a photo, permission will be requested again. It should be noted that we also request access to the storage of the mobile because of the idea that in addition to uploading the photos to the system, the user can view their photos and audios on the device, as do other multimedia or instant messaging applications.

```
void onRequestPermissionsResult(permissions,grantResults) {
  onRequestPermissionsResult(permissions,grantResults);

  if (requestCode == MY_CAMERA_PERMISSION_CODE) {

     if (grantResults[0] == PERMISSION_GRANTED) {
      Toast.makeText(getContext(), "Permisos de la cámara
         concedidos", Toast.LENGTH_LONG).show();
      dispatchTakePictureIntent();
    } else {
      Toast.makeText(getContext(), "Permisos de la cámara
         denegados", Toast.LENGTH_LONG).show();
```

```
11          }
12      }}
```

Then the native camera of the device is opened as you can see in Figure 8.9. At this point, the user can take the picture or not and depending on the action taken the system reacts in one way or another.Next we will explain step by step the possible actions:

Figure 8.9: Screen of camera open

With the camera open, we can take two paths. The first is to reject the photo. The second is to take a picture. For this example, we will use the camera of the emulator, because it allows us to make the screenshots as the transitions are made in the system.

First, we're going to comment on rejecting a photo. When the user cancels the action, the system displays the photo view (see Figure 8.10). This view contains several elements. In the central position, a container with a possible image and above it, a small button for photo editing. Just below, there is a text indicating the status of the task; in this case, the task to be performed and a save image button, which is not enabled. Finally, we have the lower menu with the respective coloured item.

Figure 8.10: Screen of camera view

This view contains several elements. In the central position, a container with a possible image and on it, a small button for photo editing. Below, there is a text indicating the state of the task; in this case, not performed. Next, a save image button, which is not enabled. Finally, we have the lower menu with the respective coloured item.

If he wants to upload an image again, clicks on the camera icon in the bottom menu or on the edit image button. Both widgets open the device's camera (see Figure 8.9).

When a user opens the camera (see Figure 8.9), it is most likely to take a picture. If so, click on the camera icon to take a picture (see figure 8.11). Then an event is triggered that captures the focused image; it can be a selfie or a normal one and shows it on display. Depending on the Android device, the way the result is processed may vary. Because there is so much variety, and because each manufacturer applies its policies to image processing, the result may change. Typically, after a photo is taken, there are three buttons: retry, accept, cancel.

Figure 8.11: Screen of take a photo

In case of acceptance, the camera API returns a response attempt that has to be captured by our system. As you can see in the following code when winning the event, we can access the information sent, which is only a set of bits in URL format, also known as URI.

```
@Override
public void onActivityResult(int requestCode, int resultCode,
    Intent intent) {
  super.onActivityResult(requestCode, resultCode, intent);
  try {
    switch (requestCode) {
      case REQUEST_TAKE_PHOTO: {
        if (resultCode == RESULT_OK) {
            File file = new File(mCurrentPhotoPath);
            Bitmap bitmap = MediaStore.Images.Media.getBitmap(
               getContext().getContentResolver(), Uri.fromFile(
               file));
```

```
10                if (bitmap != null)
11                  rotationImageToShowInFragment(mCurrentPhotoPath,
                        bitmap);
12              }
13            break;
14            }
15        }
16    } catch (Exception error) { error.printStackTrace(); }
17 }
```

But a URI is not a file that can be visualized. We have to treat this set to generate an image visible to the user. You need to convert it into a Bitmap and rotate the image so that it is displayed vertically. As a result, we will get a visible picture as you can see in Figure 8.12.

Figure 8.12: Photo display pending upload

Finally, we will explain the process of saving the image. At the moment the file is stored locally, on our mobile phone. It can be viewed by accessing the path system:

Android/Data/com.example.Psymood/files/Pictures. To finish the task and upload our image to Firebase Storage, you need to click on the Save button.

```
private void uploadPhoto() {
 textPicture.setText("Se esta subiendo ...");
 try {
    final StorageReference filepath = mStorage.child("
       daily_user_photos").child(currentUser.getUid()).child(
       currentDateandTime);
    File file = new File(mCurrentPhotoPath);
    Uri uri = Uri.fromFile(file);
    filepath.putFile(uri).addOnSuccessListener(new
       OnSuccessListener<UploadTask.TaskSnapshot>() {

    //Receive and manage the response
    @Override
      public void onSuccess(UploadTask.TaskSnapshot
         taskSnapshot) {
        filepath.getDownloadUrl().addOnSuccessListener(new
           OnSuccessListener<Uri>() {

        @Override
        public void onSuccess(Uri uri) {
            textPicture.setText(selfie_task_completed);
            //Update daily progress
            updateCounterPhoto();
            FirebaseInteractor.savePhotoInDatabase(uri.toString
               ());
            mListener.showMessageFragmentInHome("OK");
        }
        });
      }
    });
  ...
}
```

As it happens with the indicators of animal states, an event is sent to the Storage service with the image, which gives us back an **onSuccess()** object if everything has gone well and it has been possible to save it. The upload may take a few seconds and will depend on the size of the file and the internet connection of the device. To inform the upload, a transition is shown with the message: "uploading image ..." and at the end a confirmation message is displayed. Figure 8.13 can be seen in more detail.

Besides, the system updates the daily and weekly progress percentages, so that if the user consults the Main view, the number of tasks performed will have increased.

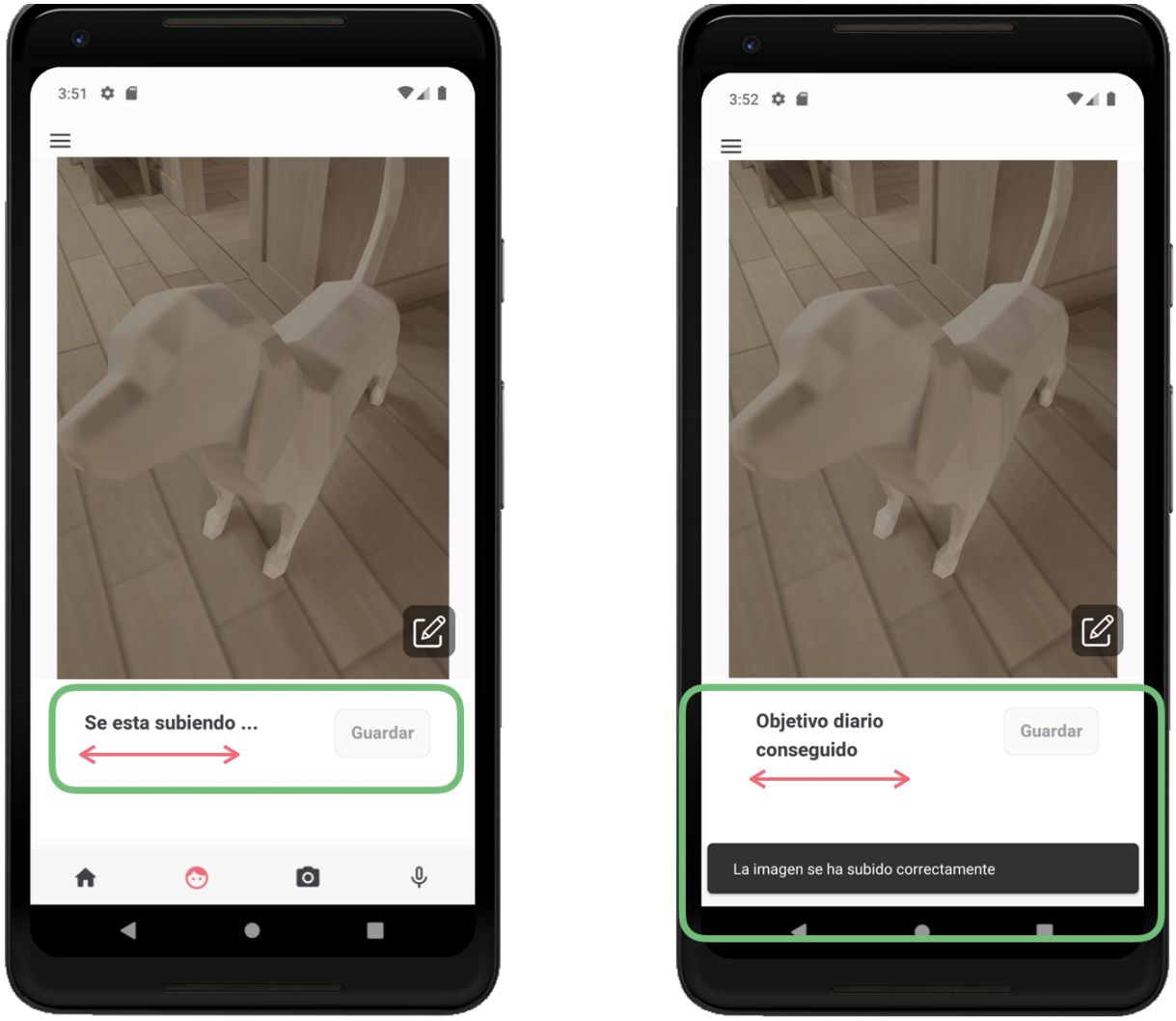

Figure 8.13: Screen of photo upload

8.3.6 Audio view

In this view, the patient's voice will be recorded. As with the other screens, it is possible to access it through the microphone icon in the bottom menu or by selecting the corresponding task in the task carousel of the dashboard.

Once in the view, we will find two main elements, as shown in Figure 8.14. At the top, there is a panel that displays a text that, depending on the situation, changes. In the centre of the screen, a large red button with a microphone, which is used to start recording. At the bottom, the navigation menu with the audio item selected.

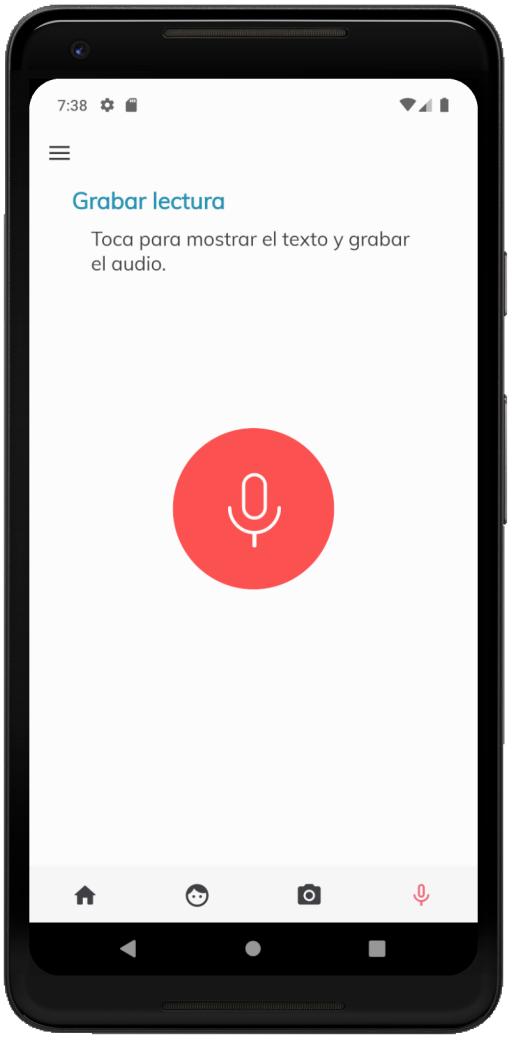

Figure 8.14: Screen of audio view

The operation is simple. When we touch the microphone button, a text is displayed on the top panel, and the recording starts with an animation of the microphone. The user has to read it and once finished, tap on the animation to stop the recording (see Figure 8.15 - Left device)

From a more technical point of view, when we click on the microphone button, a permission request is executed to access the recording sensor. As you can see in the following code, once the permissions are granted, we can start the recording.

```
1 public void onRequestPermissionsResult(...) {
2   switch (requestCode) {
3   case REQUEST_RECORD_AUDIO_PERMISSION: {
4     // If request is cancelled, the result arrays are empty.
5     if (grantResults.length > 0
6       && grantResults[0] == PackageManager.PERMISSION_GRANTED)
          {
7       onRecord(mStartRecording);
8       mStartRecording = !mStartRecording;
```

```
    }else {
      // permission denied, boo! Disable the
      // functionality that depends on this permission.
       Toast.makeText(getContext(), "permission denied",...);
}}
```

The voice audio is stored in a MediaRecorder object, native to Android studio. This object allows us to specify some parameters of the recording and how the final file should be. According to Android developers documentation, audios must be saved as files with .3gp extension.

```
...
mRecorder = new MediaRecorder();
mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP
    );
mRecorder.setOutputFile(mFileName);
mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
try {
  mRecorder.prepare();
} catch (IllegalStateException | IOException e) {
  Log.e(LOG_TAG, "Error to prepare recorder object");
}
mRecorder.start();
...
```

The recording continues until the stop event is received. This event is launched by the user when he touches the animation on the screen. It is captured and processed in the stopRercoding function, see the code below, which alerts the system that the recording has finished and that it has finished using the sensor.

```
private void stopRecording() {
    mRecorder.stop();
    //Save the release
    mRecorder.release();
    //Finished the recording
    mRecorder = null;
    showDialogConfirm();
}
```

After stopping the animation, a pop-up confirmation of sending appears instantly. The pop up contains an audio player and a confirmation button. If the user wants to listen to the audio, by clicking on the play, he can play it, or stop it with the pause button (see Figure 8.15).

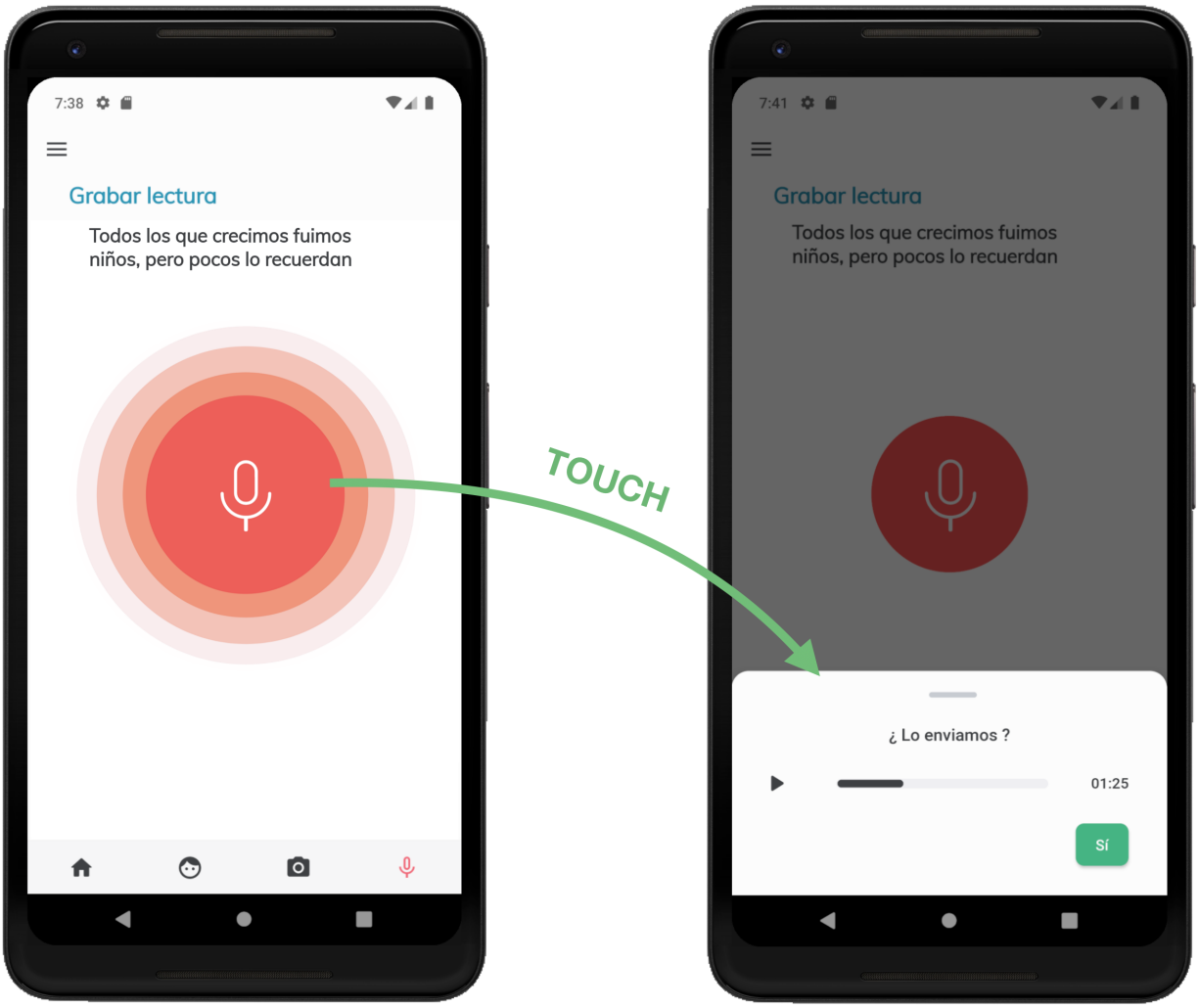

Figure 8.15: Recording and confirmation audio screen

To discard the audio, click outside the area occupied by the pop-up or slide it down. Either action will delete the file. If we say upload it, click on the OK button and it will be sent to Firebase Storage's audio record. As with the images and states, the audios are stored in the respective user's register.

It is possible that a problem may occur during the upload, as it happens with the images, or that when recording the information, an error may occur. The error and exception handling works identically to the error handling of the pictures, so we will not go into further details.

In case everything went well, a confirmation message is displayed with the following text: "audio has been uploaded successfully", as shown in figure 8.16. The system's measurement values are also updated, which involves updating the daily progress bar and the weekly evaluation graph. If the user returns to the main view, he can check these changes.

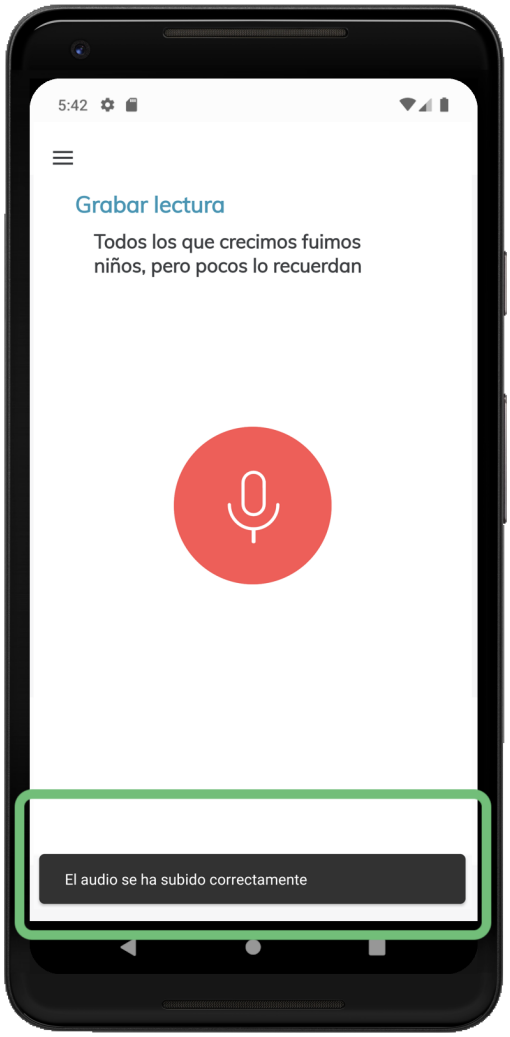

Figure 8.16: Audio upload view

8.3.7 Settings view

The settings view allows you to edit and configure aspects of the user's account, such as: changing the profile photo, updating the username, viewing the email associated with the account or setting your date of birth. Besides, it includes the option to download the information stored in the system's database.

To access this view, it is necessary to navigate through the hamburger menu. When you click on it, a drawer menu opens from the right side (see Figure 8.17). This menu can be hidden by sliding it to the opposite side or by clicking on the grey shaded area of the screen.

The drawer menu contains several three sections, each with different elements and navigation options. Each part has been listed for ease of explanation. Each is listed below.

1 This is the top drawer of the menu. It consists of the user profile image, the name and a button to access the settings.

2 The central object contains several links to the different screens and a logout button.

3 The session 'about' allows access to additional information such as terms and conditions of use, security policies and the version of the application.

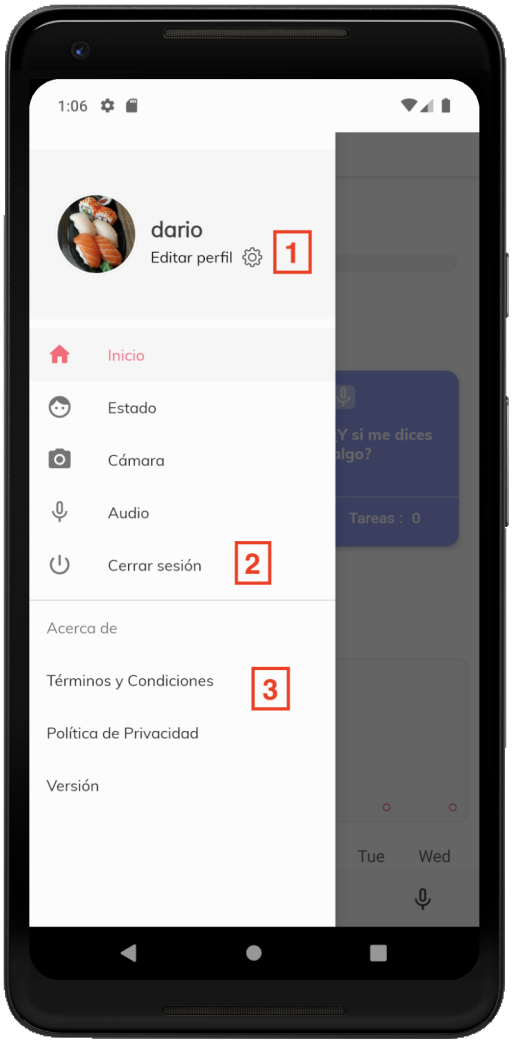

Figure 8.17: Menu drawer

When the gear button is pressed, the system opens the settings view (see Figure 8.18). As mentioned above, the view is composed of several fields that allow you to update the user information. The first items refer to the user's profile, such as a photo, name, email, age or gender. All the fields are editable except the email because it serves as an identifier in the application. To keep the changes, you have to click on the Save button.

As additional functionalities, we can download all the information and indicators registered in Psymood until that moment. The data is presented in a CSV file. It is essential to clarify that the images and audios, which take up a lot of space, will not be saved directly, instead of links to each of them are included.

Finally, a logout button has been included, which works similarly to the drawer menu.

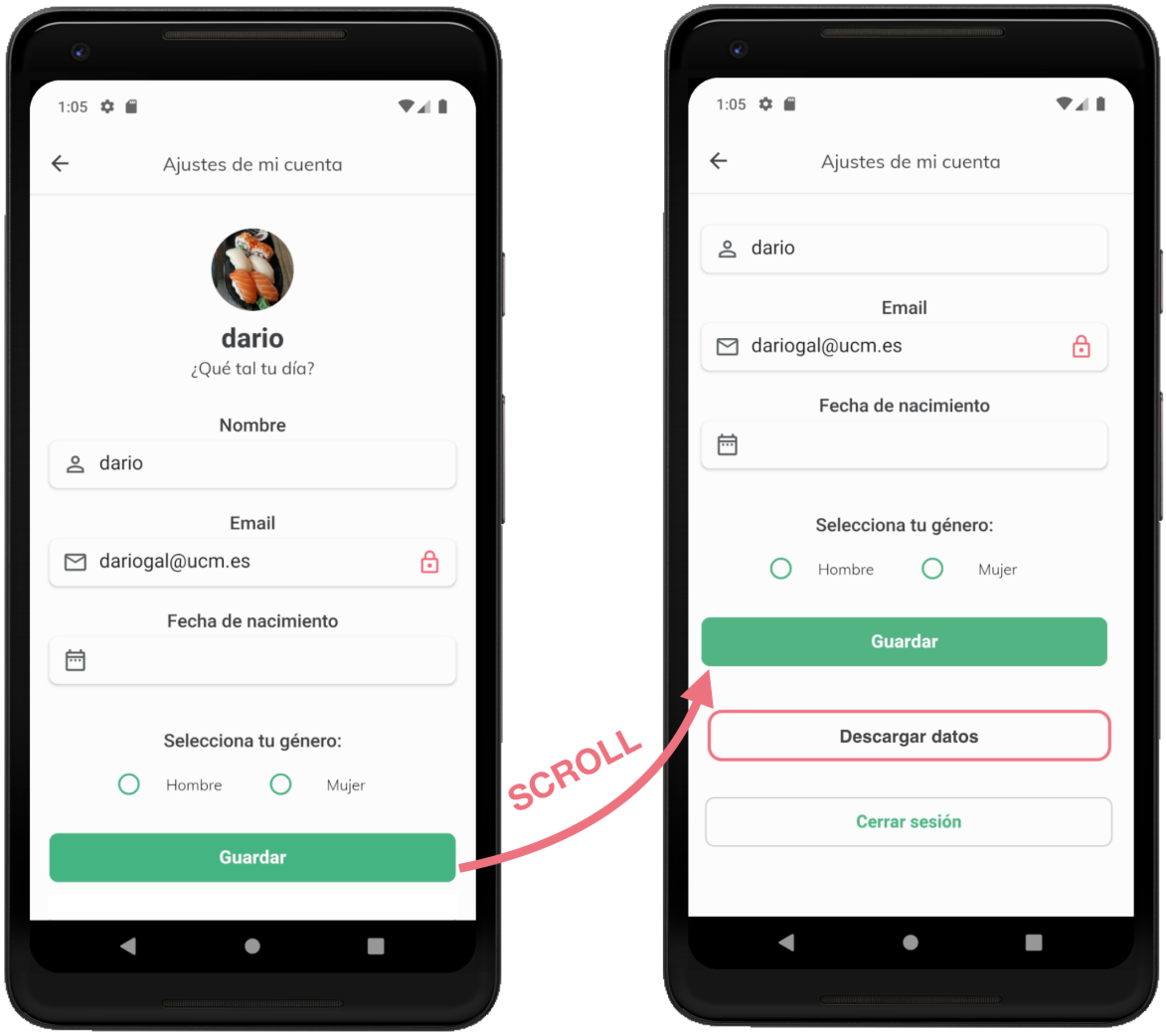

Figure 8.18: Sreen of Settings view

Chapter 9

Conclusions and future work

In this chapter, we are going to expose the conclusions obtained after finishing the system and the future work that we consider relevant to the improvement of the system.

9.1 Conclusions

This study aims to provide a solution to the difficulty of identifying mood disorders and the need to speed up this process through the use of technological tools.

To achieve this goal, a system has been developed that allows users to collect and store information through a scalable database capable of storing large amounts of information and multimedia files. Besides, the system not only allows the users to observe and carry out control of the information uploaded but also allows them to extract this information in the form of an ordered CSV file with all the data extracted from each patient. Last, the system counts with an intuitive and friendly that facilitates interaction with it and the collecting of the data.

As conclusions of this study, we can highlight two main ones. The first one is that having a deep understanding of the data we are working with is essential. Knowing which variables are more relevant for mood disorders can help that the stored information have more quality and future prediction models to be more precise. Second, the development of a real system from the beginning to the end is hard. So that it is important to spend time in the planning, analysis, and design since they facilitate the work and verify that the system meets with the established requirements.

9.2 Future work

First, we would like to carry out a pilot test of our system so we would like to contact the Bipolar Association of Madrid. Our goal is to get a group of volunteers to try the system and give us feedback later.

Second, we consider important to extend the system to other operating systems. Currently, this system only works in Android and despite this operating system was chosen because it covers a wide range of population, it is important to not restrict the public that can take advantage of the system.

Third, it would be beneficial to add more variables to collect more information. In special, we would like to highlight the upload of videos. A video mixes the voice and the gestures of the patient that could help to collect information of great help to the doctor. Besides, we also consider beneficial to include the monthly and weekly progress of the patient.

Last, it is important to improve the security of the system since the data stored is very sensible and to translate the system to more languages because as we know, it is a problem that affects whole Europe.

Chapter 10

Conclusiones y trabajo futuro

En este capítulo vamos a exponer las conclusiones obtenidas tras la finalización del sistema y el trabajo futuro que consideremos relevante para la mejora del sistema.

10.1 Conclusiones

Este estudio pretende dar solución a la dificultad de identificar los trastornos del estado de ánimo y a la necesidad de acelerar este proceso mediante el uso de herramientas tecnológicas.

Para ello, se ha desarrollado un sistema que permite a los usuarios recoger y almacenar información a través de una base de datos escalable capaz de almacenar grandes cantidades de información y archivos multimedia. Además, el sistema no sólo permite a los usuarios observar y llevar un control de la información cargada, sino que también les permite extraer esta información en forma de un archivo CSV ordenado con todos los datos extraídos de cada paciente. Por último, el sistema cuenta con un sistema intuitivo y amigable que facilita la interacción con el mismo y la recolección de los datos.

Como conclusiones de este estudio, podemos destacar dos principales. La primera es que es esencial tener un profundo conocimiento de los datos con los que estamos trabajando. Conocer qué variables son más relevantes para los trastornos del estado de ánimo puede ayudar a que la información almacenada tenga más calidad y a que los modelos de predicción futuros sean más precisos. En segundo lugar, el desarrollo de un sistema real desde el principio hasta el final es difícil. Por lo que es importante dedicar tiempo a la planificación, análisis y diseño ya que facilitan el trabajo y verifican que el sistema cumple con los requisitos establecidos.

10.2 Trabajo futuro

En primer lugar, nos gustaría realizar una prueba piloto de nuestro sistema, por lo que nos gustaría contactar con la Asociación Bipolar de Madrid. Nuestro objetivo es conseguir que un grupo de voluntarios prueben el sistema y nos den su opinión más adelante.

En segundo lugar, consideramos importante extender el sistema a otros sistemas

operativos. Actualmente este sistema sólo funciona en Android y a pesar de que se eligió este sistema operativo porque cubre un amplio rango de población, es importante no restringir el público que puede aprovechar el sistema.

En tercer lugar, sería beneficioso añadir más variables para recoger más información. En especial, nos gustaría destacar la subida de vídeos. Un vídeo mezcla la voz y los gestos del paciente que pueden ayudar a recoger información de gran ayuda para el médico. Además, también consideramos beneficioso incluir la evolución mensual y semanal del paciente.

Por último, es importante mejorar la seguridad del sistema ya que los datos almacenados son muy sensibles y traducir el sistema a más idiomas porque, como sabemos, es un problema que afecta a toda Europa.

Chapter 11

Individual contributions

In this chapter, we are going to explain the tasks that each student has done during the study. These tasks have been assigned taking into account the strengths of each member and their different backgrounds.

11.1 Mishell Janina Tigse Ortiz

Concerning Mishell's contribution to the project, the following sections are distinguished:

11.1.1 Prototyping

After my partner analyzed the requirements and after several sessions with our director, I proceeded to create the mockups of our systems that can be seen in the Design chapter. In addition, animations were also created in the program to create a final prototype. Finally, after the changes proposed by our director, a final Psymood prototype was obtained.

11.1.2 Terms and conditions

Every system has to have terms and conditions of service that include information about accounts and membership, not allowed uses of the system, changes and amendments, the terms that the user accepts when using the system and finally a contact in case of any doubt.

To develop this feature an analysis of the terms and conditions of mobile systems like ours was done. In this way, I established some terms and conditions of the system, without being too restrictive because we wanted to cover a large public but clear. So that the user had a deep understanding of who could use the application, which was not allowed, the terms they accepted when using Psymood and finally, an email account was created so that they could contact us in case of doubt.

As for the development of this feature, it was done in an activity that is displayed by clicking the 'Terms and Conditions' button. This activity includes a text view that is within a scroll that displays all the information described above. As for the difficulty of the development, the most difficult part was to center the text, to choose a good style and size of the font suitable for all screen sizes.

11.1.3 Privacy policy

In addition to the terms and conditions, every system must have its privacy policy. This privacy policy must contain the collection of information and use made of it, the data necessary for the registration, the security used, the privacy of children, the terms regarding changes in the privacy policy and a contact form.

Thus, the privacy policy gathers all these sections, first informing the user about the collection of photos, audios, and more personal information, as well as informing about the registration data such as email and name. Informing you about the security of your data, including children, as our system accepts people from thirteen years old and ends with our terms when changing the privacy policy and our contact email.

To develop this feature, activity has been used that is displayed by pressing the 'Privacy policy' button. This contains a scroll that in turn contains a text view with all this information. Again, the most difficult part of this development has been the adaptability to all screens.

11.1.4 Application version

Application version is the easiest feature because it is an only activity with a number in the middle. This number indicates the current system version and is updated each time a new function is implemented. Throughout the development, it was changed every time the functionalities planned for each sprint were included. This activity is displayed when you click on the 'Version' button.

11.1.5 Settings

The development of some system adjustments is important since the user may want to change the name and the photo, add the gender and date of birth and look at his email since this is immutable. In addition, on this screen, we also find the download of the data in CSV. This screen was made by my partner and me, but I will only focus on what I did, such as the field of name, email, date of birth and gender selection. The most complicated part of this screen was to take out the calendar and save the dates in Firebase since it was totally new for me. It should be noted that all these fields are saved and modified each time the user presses the 'Save' button.

11.1.6 Camera

The development of the camera has been a functionality with my partner Dario because he has more experience in Android development. The screen of the camera is not an activity but a fragment, so that added more difficulty. In this way, I took the photos from the fragment along with the acceptance of permits, however I had many problems when saving the photo in the database, because I could not find the route, in this way my partner helped me recover the photo taken and save it in Firebase.

11.1.7 Memory chapters

My partner and I divided the chapters that we should write each in memory. So I am the author of the following chapters, Introduction, Used Technologies, Design, Conclusion and Future Work, both in English and Spanish. In addition, I also wrote the Authorization of diffusion and use, Acknowledgment, Abstract and Keywords sections, in English and Spanish.

Finally, both the Android Development chapter and the bibliography have been written together. In the Development with Android chapter, I wrote the Android General Concepts section and in the bibliography, the contributions are made by the chapter of each one.

11.2 Darío Fernando Gallegos Qhispe

Concerning Dario's contribution to the project, the following sections are distinguished:

11.2.1 Define the requirements

To develop Psymood it was necessary to spend a great deal of time defining the requirements. I did not know very well the possible competition that our system would have, so we made a market analysis, evaluating the different applications, studies and algorithms in the field of mood and system disorder to define the requirements, determine the needs and conditions to be satisfied by the software.

11.2.2 Data model

To develop the data model, I identify the types of identities, attributes and indicators resulting from the prerequisites, and the possible relationships that could arise. Nomenclature conventions, models for modelling were applied to reduce data redundancy and improve database performance.

11.2.3 Design, development and implementation of Firebase system

From the data model, design the structure of the Firebase database. As this platform offers many services I had to learn how each one works and how they join our project. The services I finally use are the real-time database, which contains the entire organization and structure of the data collection system, the authentication service to generate user credentials, session validation and data encryption, allowing that Psymood is a secure system and the storage of large files in Storage.

Each of these services requires a separate implementation, so I had to develop a system of connections and information exchange that will be centralized in the database in real time. Besides, as Firebase allows access to the database from any device, it was necessary to configure the platform access rules, so that only administrators and developers had access to the platform console.

To test the performance of the database, test applications were generated to measure and improve the performance of connections and calls to Firebase. Once completed, it was implemented in the development of the mobile application.

11.2.4 Design, development and implementation of Android struct

To develop the mobile application, it was necessary to divide the process into different stages. From the mockup designed as a whole, I focused on creating the structure of the entire application with the mind that the system should be light and intuitive. Then I implemented the navigation system of the views and once everything was working, I added the operation of each of them.

The following view was the collection of mood indicators. While redesigning the state indicators, I applied design patterns from MVC, DAO, Facade and Singleton, to improve the graphic interfaces and make the view could hold a lot of information without overloading the system. On the other hand, the connection of the Realtime system was developed, so that every time the user indicated something, his information would be saved.

Development of the audio recording view. For this it is necessary to learn how to use the voice recording sensor, processing of the selected data to convert it into files that can be played as mp3 or 3gp, playback system, modification and confirmation of the audio obtained for your posters sending and uploading files It's Firebase Storage. As for the design, create an animation for voice recording and be better the audio playback and confirmation system.

Also, a file storage system will be implemented in the system's local memory, so that the audios and photos that the user loads are also stored in the device's memory.

The camera view allows us to collect daily photos of the user. Make a redesign of the view, implement the structure of the photo system and together with my partner we investigate develop the operation of taking photos and save the image in the device memory. Once we verify the correct operation, implement the function of validation, upload, storage and error control of the photos the Firebase Storage system.

So that the user can keep track of uploaded data and pending tasks, so with a report of daily and weekly progress, develop and implement the main or dashboard view. All the elements were programmed to work reactively and asynchronously so that when a change occurs, all the elements will be updated automatically.

Similar to other applications, it is necessary to have a view of settings that allows you to change the username, profile picture, date of birth and gender. Also, data is downloaded and a session is closed. Following the design of the mockups I implement this view, with certain changes to improve functionality and add new options. Develop the access functions to the system gallery to select the profile photo, the scroll of the view, download of files and information from the user's database generating a CSV in the internal memory of the device and updating the data of the profile in Firebase. As for the functionality of the storage, the birthday and the gender, it was developed and implied by my partner.

11.2.5 Version control

To keep track of code and work more safely, a control version system with Github and Git was added to the project. Each time we made an important change, the branch was opened to work and once the functionality was completed, it was drawn onto the develop branch.

11.2.6 Error correction

Once the application is finished, it is tested thoroughly to look for possible errors and errors. As I tested the operation, if I had an error, opened a problem report, generated a branch to work and once solved, merged in the development branch.

11.2.7 Memory chapters

For memory, my partner and I distributed the chapters that had to be written. I wrote the chapters of State of the art, Use cases and Firebase, and together with my partner the Android chapter.

Bibliography

[1] WHO | World Health Organization.
https://www.who.int

[2] Bip4Cast.
https://bip4cast.org

[3] P. LLamocca, D. Urgelés, M. Cukic, V. López, Bip4Cast: Some advances in mood disorders data analysis, Proc. 1st International 'Alan Turing' Conference on Decision Support and Recommender Systems (DSRS-Turing 2019), The Alan Turing Institute, London, United Kingdom, November 2019.

[4] Mood Disorders 2.
https://www.youtube.com/watch?v=yG04pMUasfQ

[5] P. Llamocca, A. Junestrand, M. Cukic, D. Urgelés, V. López, 'Data source analysis in mood disorder research', XVIII Proceedings of the XVIII Conference of the Spanish Association for Artificial Intelligence, CAEPIA, ISBN: 978-84-09-05643-9 F. Herrera et al. (Eds.), pp. 893-900, Granada, Spain, 2018.

[6] V. López, D. Urgelés, O. Sánchez and G. Valverde. 'Big Data in Healthcare and Social Sciences: Bip4Cast as a CAD system', International Journal of Information Systems and Social Change (IJISSS), ISSN: 1935-5688, vol. 8, No. 3, pp.1-16 July 2017.

[7] V. Lopez, G. Valverde, J. Anchiraico and D. Urgelés. 'Specification of a Cad Prediction System for Bipolar Disorder', Uncertainty Modelling in Knowledge Engineering and Decision Making, Proceedings of the 12th International FLINS conference, World Scientific Proceedings Series on Computer Engineering and Information Science, vol. 10, pp. 162-167. ISBN 978-981-3146-96-9, World Scientific, 2016.

[8] Clínica Nuestra Señora de la Paz.
http://www.nuestraseñoradelapaz.es/

[9] 3EGA.
https://3ega.com/

[10] A. Martínez. 'Introduction to Big Data and First Steps in a Big Data Project', Trabajo de Fin de Grado, Madrid: E-Prints Universidad Complutense de Madrid, Spain, Madrid, 2016.

[11] J. TorousEmail, P. StaplesJukka, P. Onnela. 'Realizing the Potential of Mobile Mental Health: New Methods for New Data in Psychiatry', Psychiatry in the Digital Age (JS Luo, Section Editor), 16 June 2015.

[12] A. Plitcha, S. Szomińsky. 'Models of it project management implementation and maintenance', 2016.
http:/yadda.icm.edu.pl

[13] A. Junestrand Leal. 'Application of Machine Learning Algorithmsfor Bipolar Disorder Crisis Prediction', Bachelor's Degree Final Project in Computer Science and Engineering, Madrid, Spain, 2018.

[14] J. Sadavoy, R. Meier, A. Yuk Mui Ong. 'Barriers to Access to Mental Health Services for Ethnic Seniors: The Toronto Study', March 1, Toronto, Canada, 2004.

[15] S. Richard Chan, J. Torous, L. Hinton, P. Yellowlees. 'Mobile Tele-Mental Health: Increasing Applications and a Move to Hybrid Models of Care', 6 May 2014.

[16] Top 25 Mental Health Apps: An Effective Alternative for When You Can't Afford Therapy?
https://www.psycom.net/25-best-mental-health-apps

[17] J. Scott, E. Paykel, R. Morriss, R. Bentall, P. Kinderman, T. Johnson, R. Abbott, H. Hayhurst. 'Cognitive–behavioural therapy for severe and recurrent bipolar disorders: Randomised controlled trial', Cambridge University, January, 2018.

[18] MindShift CBT - Anxiety Canada, 2016.
https://play.google.com/store/apps/details?id=com.bstro.MindShift&hl=es_419

[19] Anxiety Canada Association
https://www.anxietycanada.com/

[20] Moodpath - Depression Anxiety Test
https://play.google.com/store/apps/details?id=de.moodpath.android&hl

[21] MoodTools - Depression Aid
https://play.google.com/store/apps/details?id=com.moodtools.moodtools

[22] FearTools - Anxiety Aid
https://play.google.com/store/apps/details?id=com.feartools.feartools

[23] Depression Questionnaire (PHQ9)
https://www.fundacionmf.org.ar/visor-producto.php?cod_producto=3226

[24] Dobson, Keith S.,Hollon, Steven D.,Dimidjian, Sona,Schmaling, Karen B.,Kohlenberg, Robert J.,Gallop, Robert J.,Rizvi, Shireen L.,Gollan, Jackie K.,Dunner, David L.,Jacobson, Neil S. 'Randomized trial of behavioral activation, cognitive therapy, and antidepressant medication in the prevention of relapse and recurrence in major depression.', Journal of Consulting and Clinical Psychology, 76(3), 468–477, 2008.

[25] TalkLife - Lonely, Stressed or Battling Anxiety?
https://play.google.com/store/apps/details?id=com.bearpty.talklife

[26] Sanvello
https://play.google.com/store/apps/details?id=com.pacificalabs.pacifica

[27] React Native · A framework for building native apps using React.
https://facebook.github.io/react-native/

[28] Gitflow Workflow
https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[29] Git –local-branching-on-the-cheap
https://git-scm.com

[30] Github platform.
https://github.com/features/code-review/

[31] Psymood Github repository .
https://github.com/dariogallegos/Psymood

[32] Trello platform.
https://trello.com/

[33] Introduction to Kanban Methodology.
https://resources.collab.net/agile-101/what-is-kanban

[34] Reglamento (UE) 2016/679 de 27 de Abr DOUE (Reglamento general europeo de protección de datos (GDPR/RGPD)).
https://www.reglamento-ue-2016-679-27-abr-doue-reglamento-general-europeo-proteccion-datos-gdpr-rgpd

[35] Marvel App.
https://marvelapp.com/

[36] Firebase documents.
https://firebase.google.com/docs

[37] Firebase Analytics documents.
https://firebase.google.com/docs/analytics/get-started?platform=android&hl=es-419

[38] Google Analytics documents.
https://analytics.google.com/analytics/web/provision

[39] Dynamic audiences in Google Analytics.
https://www.blog.google/products/marketingplatform/360/dynamic-audiences-google-analytics-firebase/

[40] Firebase cloud messaging service.
https://firebase.google.com/docs/cloud-messaging/

[41] Cloud Storage | Firebase Cloud Storage.
https://firebase.google.com/docs/storage?hl=es

[42] Firebase Remote Config.
https://firebase.google.com/docs/remote-config

[43] Firebase Authentication | Firebase.
https://firebase.google.com/docs/auth/

[44] Firebase Realtime Database.
https://firebase.google.com/docs/database/

[45] Android SQLite database.
https://developer.android.com/training/data-storage/sqlite

[46] Mongodb documentation.
https://docs.mongodb.com/manual/

[47] Gradle Build Tool | Tutorial and guides.
https://docs.mongodb.com/manual/

[48] Firebase guide: Add firebase at Android proyect | Tutorial and guides.
https://firebase.google.com/docs/android/setup

[49] Google Play Store.
https://play.google.com/store?hl=es_419

[50] Overview Google Play Services.
https://developers.google.com/android/guides/overview

[51] Google Maven resository from Apache Maven.
https://maven.apache.org/

[52] MySQL reference manual.
https://dev.mysql.com/doc/refman/8.0/en/

[53] Oracle Database documentation.
https://docs.oracle.com/en/database/

[54] Saving Data | Firebase Realtime.
https://firebase.google.com/docs/database/admin/save-data

[55] SDK Platform Tools.
https://developer.android.com/studio/releases/platform-tools

[56] AndroidX Overview | Android Developers.
https://developer.android.com/jetpack/androidx

[57] Documentation for Android SDK.
https://developer.android.com/guide/components/activities

[58] Utilizar Fragmentos En Una Aplicación Android.
http://www.hermosaprogramacion.com/2014/09/android-aplicaciones-fragmento/