Development of an Android value-added service for patients in initial phases of Alzheimer

Desarrollo de una aplicación Android de valor añadido para pacientes en primeras fases de Alzheimer

Universidad Complutense de Madrid

Facultad de Informática

Trabajo Fin de Grado en Ingeniería Informática

Curso 2020/2021

Jaime Antolín Merino
Dirigido por: Antonio Sarasa Cabezuelo

Autorización de Difusión

Los abajo firmantes, alumno y tutor del Trabajo Fin de Grado (TFG) en Ingeniería Informática de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, el Trabajo Fin de Grado (TFG) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Titulo del TFG: **Desarrollo de una aplicación Android de valor añadido para pacientes en primeras fases de Alzheimer**

Curso Académico: 2020-2021

Nombre del Alumno: **Jaime Antolín Merino**

Director y Tutor del TFG: **Antonio Sarasa Cabezuelo**

Firma del alumno Firma del director

Abstract

The project consists of an Android application designed to help patients in the first stages of Alzheimer to recognize objects they can encounter around their homes and explain how to execute mundane activities with them.

The application allows a care taker or family member to create a customized knowledge base for the home of each patient. This knowledge base is then exploited by the patient in various ways. The first is through a search function, patients can search either by taking a picture of an object or by their name. Both types of searches will provide the patient the necessary information to recognize the object and be reminded of how to use it. The second way is through a multiple choice game that allows the patient to practice recognizing these objects, the care taker/ family member also has the ability to create personalized tests in order to guide the training of each patient.

Keywords

Alzheimer, Android, Google Cloud Platform, Image Labelling, Firebase, APIs, NoSQL

Resumen

El trabajo realizado consiste en una aplicación Android para ayudar a pacientes en primeras fases de Alzheimer a reconocer objetos que pueda encontrar por su casa y explicar como se realizan tareas cotidianas con ellos.

La aplicación permite a un cuidador o familiar crear una base de conocimiento personalizada para la casa de cada paciente. Esta base de conocimiento es luego explotada por el paciente de varias formas. La primera es mediante una funcionalidad de búsqueda, ya sea mediante una foto de ese objeto o el nombre. Ambos tipos de búsqueda proporcionan al paciente toda la información necesaria para reconocer y saber usar ese determinado objeto. La segunda forma es mediante un juego de selección múltiple que permite al paciente practicar reconociendo estos objetos, el cuidador/familiar también tiene la habilidad de crear tests personalizados para el paciente de manera que puede guiar su entrenamiento.

Palabras clave

Alzheimer, Android, Google Cloud Platform, Reconocimiento de Imagenes, Firebase, APIs, NoSQL

Índice general

Índice de figuras

Índice de cuadros

Chapter 1

Introduction

1.1. Introduction

Some of the common symptoms in patients who are in the first phases of Alzheimer are the difficulty to remember the word or name that they want to use, and the difficulty to carry out tasks[13]. It is apparent that technology cannot yet cure people with the disease, but it can help us to make their lives easier through the use of a mobile phone, which probably 99% of the patients already have.

With the use of a simple Android application like the one designed, this can help not only in aiding patients to remember mundane objects and how to use them, but also to train them to recognize them and that way maintain their minds active while delaying as much as possible the effects of the disease [14].

The application will allow a care taker or family member to generate a knowledge base in which they will insert the objects name, as well a brief description of how to use them and an image. Later on, the patients will be able to search for information about these objects by taking a picture of them, or by their name. The application will also allow the person in charge of taking care of the patient to create personalized tests with the objects found in the knowledge base to help patients remember and recognize the objects they can find around their house.

1.2. Motivation

The motivation for this project comes from two main pillars. The first one is about trying to make patients more autonomous and help them delay the effects of the disease. This is done by trying to reduce the amount of help they need from others to carry out a task or to try to identify what word they are trying to use. This will not only help the patients feel more fullfilled and practice cognitive functions that will help them, but will also liberate some stress from the care takers or family members since they can see that the person affected is able to be more autonomous.

The second pillar and a problem that has come to light more after the pandemic and specially the lockdown, is that a lot of people live alone and do not have someone with them 24h of the day. It is obvious that the application will not replace the care that a

person with Alzheimer needs, but the objective is that it can help the patient in the times that the person is alone to carry out tasks around their house.

Taking into acccount these issues, the application will be designed to be easy to use and with functionalitites that can firstly, be customized for each patient, and secondly, that add value to the patient in terms of recognizing and remembering house hold objects.

1.3. Objectives

The objectives for the project are then to create an Android application that is able to provide useful functionality to patients in a simple and accesible manner. It is very important that there is a 'Creator' figure who will be in charge of personalizing the experience for the patients. The role of this person will be to create and maintain the knowledge base, as well as creating the personalized tests for the patient to practice different contents each day.

After this first task is completed from the 'Creator' side, patients should be able to exploit the knowledge base in different ways depending on their needs. The first functionality will be a search option. Through a picture or the object name, the patients should be able to retrieve more information about the object. The second is to practice with personalized tests prepared by the 'Creator' to better recognize the objects in the future and to maximize cognitive activity to help in delaying more symptoms appearing.

1.4. Planning

The first thing to do is research the technologies that could give the functionalities needed for the project in a robust, but simple way:

- Android application design and SDK
- Cloud services that can provide authentication, database management and image recognition
- Conection and use of the cloud services via APIs

As the applicaiton will for now only be in Android, a deep dive into this technology and how application design works will be carrried out. For the development **Android Studio** [1] will be used and the **Java** [2] programming language, as they are the industry standards.

In terms of the cloud providers, the three main ones will be looked at(**AWS** [15], **Azure** [16], **GCP** [17]) and will be evaluated against their functionalities, their connection to the app and the possible cost of using them.

Research will also be carried out in UI design space to make sure the app is easy to use for everyone. The objective is that the app will be easy to use for people of any age and any level of technology savviness.

Capítulo 1

Introducción

1.1. Introducción

Algunos de los síntomas mas comunes para los pacientes en primeras fases de Alzheimer son la dificultad para recordar la palabra o nombre correctos que quieren usar, y la dificultad para realizar tareas [13]. Esta claro que curarles con el uso de la tecnología no es aun posible, pero si que se puede intentar hacerles la vida mas fácil utilizando un objeto que, casi seguro el 99 % de los pacientes tienen como es un teléfono móvil.

Con el uso de una simple aplicación Android como la que se ha diseñado, se puede ayudar a no solo facilitar a los pacientes recordar objetos cotidianos y como usarlos, sino también a entrenar reconociéndolos y así mantener la mente activa e intentar retrasar lo mas posible los efectos de la enfermedad [14].

La aplicación permitirá a un cuidador o familiar generar una base de conocimiento en la que se inserten los objetos junto con una descripción de como usarlo y una imagen. Posteriormente los pacientes podrán buscar información de estos elementos sacando una foto del objeto o con su nombre. La aplicación también permitirá a la persona al cargo crear tests personalizados con los objetos encontrados en la base de conocimiento para ayudar a los pacientes a recordar y reconocer los objetos que se pueda encontrar por su casa.

1.2. Motivación

La motivación para este trabajo surge por dos pilares fundamentales. El primero se trata de intentar aportar autonomía y retrasar los efectos de la enfermedad a los pacientes con Alzheimer. Tratar de hacer que no necesitan ayuda continua de un externo para realizar una tarea o para intentar descifrar como se llama lo que están intentando decir. Esto no solo ayuda a los pacientes a sentirse mas realizados y practicar funciones cognitivas que les ayudan, sino que libera algo de trabajo a los cuidadores/familiares al ver que la persona enferma es capaz de ser mas autónoma.

La segunda y un tema que ha salido a relucir después de los meses de pandemia que hemos vivido y especialmente de confinamiento, es que hay mucha gente que vive sola y no tienen a alguien las 24h del día para ayudarles en sus tareas. Es obvio que la aplicación diseñada no reemplaza ni mucho menos el cuidado que necesita una persona con esta enfermedad,

pero el objetivo es que pueda ayudar al paciente en los ratos que se encuentre sin compañía a orientarse por su casa y poder realizar las tareas que quiera.

Teniendo en cuenta estos problemas, se ha decidido diseñar la aplicación con la idea de que sea fácil de usar y que las funcionalidades permitan primero crear una experiencia personalizada para cada paciente, y segundo que aporten funcionalidades que le ayuden al paciente a reconocer y recordar que son y como usar objetos de sus casas.

1.3. Objetivos

Los objetivos del trabajo son entonces crear una aplicación Android que sea capaz de proporcionar funcionalidades útiles a los pacientes de una forma sencilla y accesible. Es muy importante también que haya una figura de 'Creador' que se encargue de personalizar la experiencia para los pacientes. El rol de esta persona es crear y mantener la base de conocimiento, además de crear los tests personalizados para que los pacientes puedan practicar cosas distintas cada día.

A partir de esta primera tarea realizada por los 'Creadores' los pacientes deberán ser capaces de explotar la base de conocimiento de distintas formas dependiendo de sus necesidades en ese momento. La primera funcionalidad sera la opción de búsqueda, mediante una imagen tomada por el paciente o mediante el nombre del objeto sobre el que quieren conseguir mas información. La segunda sera la realización de los tests preparados por el 'Creador' para potenciar al máximo la actividad cognitiva y de esta forma retrasar los efectos adversos de la enfermedad.

1.4. Plan de Trabajo

Lo primero que se realizara es un estudio sobre las tecnologías que podrán aportar las funcionalidades que se buscan de una manera robusta, pero sencilla:

- Diseño de aplicaciones Android y SDK
- Servicios Cloud que puedan proporcionar autenticación de usuarios, gestión de bases de datos y reconocimiento de imagenes
- Conexión y explotación de los servicios cloud mediante APIs

Como de momento la aplicación solo estará disponible en Android, se realizara una investigación mas profunda sobre esta tecnología y como mejor utilizarla para el diseño de aplicaciones. Para el desarrollo se utilizara **Android Studio** [1] y el lenguaje de programación **Java** [2] ya que son los estándares de la industria.

En cuanto a los servicios cloud se investigaran los tres principales proveedores(**AWS** [15], **Azure** [16], **GCP** [17]) y se evaluara las funcionalidades, la conexión a la aplicación y el coste que se podría llegar a incurrir.

También se investigara como realizar una interfaz de usuario simple y fácil de usar para todo el mundo, el objetivo es que no tenga perdida como usar la aplicación y la pueda usar alguien de cualquier edad y con cualquier nivel de soltura con la tecnología.

Capítulo 2

Estado del arte

El Alzheimer es por suerte una enfermedad muy estudiada y que tiene bastante visibilidad a nivel mundial. Es por esto que no es ninguna sorpresa que haya muchas aplicaciones diseñadas a ayudar a los pacientes, pero también a los cuidadores de estas personas. Estudiando el mercado actual se ha podido observar que hay aplicaciones que se asemejan a ciertas funcionalidades de la aplicación diseñada pero la gran mayoría han optado por ayudar en otro tipo de necesidades que estas personas pueden tener.

- **Iridis** [18] - Esta aplicación permite a los cuidadores o familiares analizar los espacios donde vive el paciente y analiza en base a unos estudios realizados por la Universidad de Stirling si ese espacio es favorable o no para un enfermo de Alzheimer o demencia. Favorable en este caso se refiere a que no le cause estrés o pueda disparar una situación de pánico.
- **Timeless** [19] - Timeless se centra mas en la comunicación entre el paciente y los familiares, y también tiene unas funcionalidades de calendario y de reconocimiento de personas en su circulo mas próximo. Cabe destacar que en esta aplicación también existen los roles, y la aplicación para el paciente no es la misma que para el cuidador o incluso hay otro rol de familiar. Los familiares pueden enviar imágenes al paciente y pedir crear eventos en su calendario. El paciente también puede hacer llamadas directamente desde la aplicación a sus personas mas allegadas y es avisada cuando ya ha llamado a alguien en los últimos 5 minutos.
- **MindMate** [20] - Mindmate esta puramente centrada en mantener el cerebro activo y así estimular las funciones cognitivas. Esto lo hace a través de ofrecer a los usuarios mini-juegos para estimular la actividad cerebral, también recomienda diariamente artículos o vídeos de los que se pueda aprender algo nuevo. También se centra un poco en la salud física y tiene una sección de recomendaciones de ejercicios e incluso recetas. La idea es mantener a las personas activas de todas las formas posibles y realizando una vida saludable.
- **Spaced Retrieval Therapy** [21] - Esta aplicación también se centra exclusivamente en el entrenamiento cognitivo de los pacientes. Implementa la probada técnica de "Spaced Retrieval"que se basa en preguntar algo en intervalos crecientes de tiempo para asegurar que se queda en la memoria a largo plazo. La aplicación incrementa el tiempo entre intervalos de las preguntas que se responden correctamente y lo decrementa para las respuestas incorrectas. La idea es usar la aplicación para que

los pacientes recuerden que hacer en situaciones de peligro o de estrés.

Como podemos ver de momento en el mercado existen muchas aplicaciones que promueven la actividad mental de los pacientes a través de diversos ejercicios. Pero no existe nada que cree una base de conocimiento personalizada y luego permita a los pacientes explotarla para recordar información sobre los objetos que se pueda encontrar por su casa.

Capítulo 3

Especificación de requisitos

3.1. Actores

La aplicación cuenta con los siguientes actores:

- **Usuarios no registrados** - Un usuario que aun no esta validado por el administrador por lo que no tiene acceso a ninguna funcionalidad de la aplicación
- **Pacientes** - Aquellos usuarios que tendrán acceso a las funcionalidades de explotación de la base de conocimiento
- **Creadores** - Usuarios que estarán encargados de crear y mantener la base de conocimiento y de crear los tests personalizados para los pacientes
- **Administrador** - Encargado de gestionar la validación de usuarios

3.2. Módulos

La funcionalidad de la aplicación se ha juntado el los siguiente módulos:

- Módulo Autenticación
- Módulo Explotación
- Módulo Base de conocimiento

3.2.1. Módulo Autenticación

En este modulo se ha agrupado toda la funcionalidad de registro, autenticación y acceso a la aplicación.

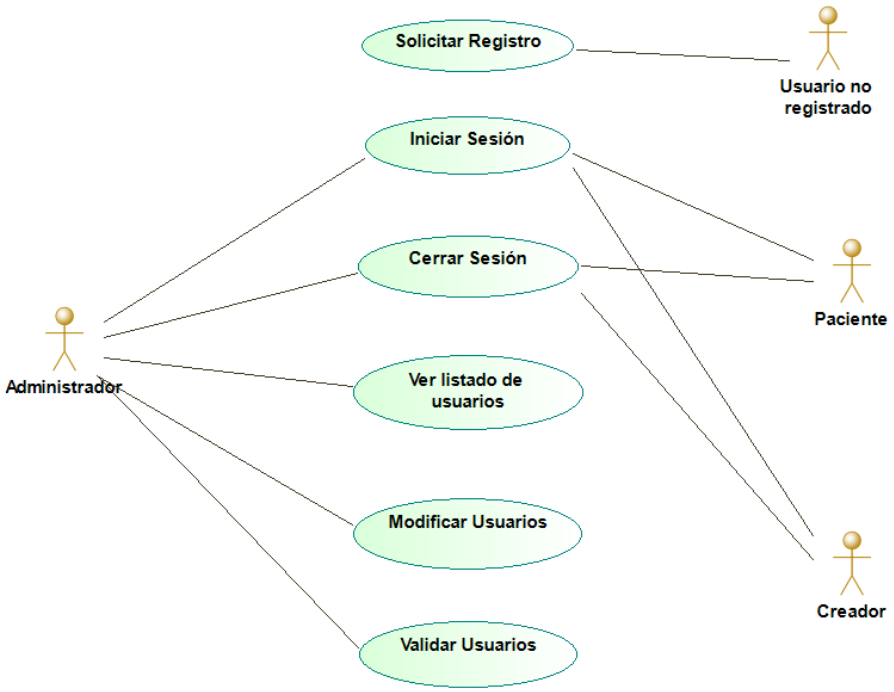


Figura 3.1: Módulo Autenticación

Los usuarios no registrados solo podrán solicitar el registro y esperar a que el administrador valide o no su cuenta. El paciente y el creador son usuarios que han solicitado un tipo de cuenta y ya han sido validados por el administrador por lo que podrán acceder a la aplicación y cerrar sesión. El administrador podrá ver el listado de usuarios, modificar sus datos, y su función mas importante validarlos para que puedan acceder a la aplicación.

3.2.2. Módulo Explotación

En este modulo se ha agrupado toda la funcionalidad en relación con la explotación de la base de conocimiento por parte del paciente y el creador.

Aquí los protagonistas son los pacientes ya que todo se trata de explotar los datos que tenemos en la base de conocimiento. Los pacientes pueden hacer una búsqueda a través de una foto o un nombre y recibir información a cambio. El paciente puede también ver el listado de tests personalizados y elegir el que quiera, o puede generar uno aleatorio. El creador en cambio solo puede ver el listado de posibles elemento y desde ahí crear tests personalizados para los pacientes.

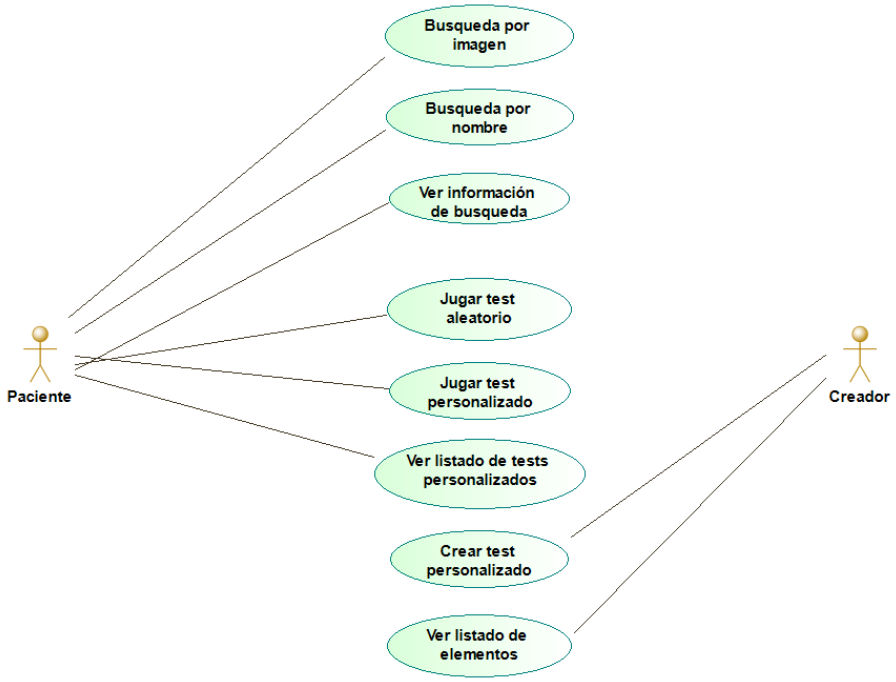


Figura 3.2: Módulo Explotación

3.2.3. Módulo base de conocimiento

En este modulo se ha agrupado toda la funcionalidad en relación con la creación de la base de conocimiento, en este caso el único actor sera el creador.

Como podemos ver aquí el creador es el único que puede manipular la base de conocimiento de su paciente. Es el encargado de moverse por la casa del paciente e ir añadiendo los objetos que cree que le puedan ser útiles tener documentados. Tendrá también que sacarles una foto para que podamos enseñársela luego al paciente, y para que podamos pasarla por la API de reconocimiento de imagen y que nos devuelva una etiqueta que usaremos luego para matchear el objeto. Una descripción con información relevante sobre el objeto también sera necesaria. Una vez creado un objeto como elemento en la base de conocimiento el creador podrá verlo en la lista, seleccionarlo y modificar cualquiera de sus atributos (excepto la etiqueta aportada por la API) e incluso eliminar el elemento si se decide que ya no es útil tenerlo.

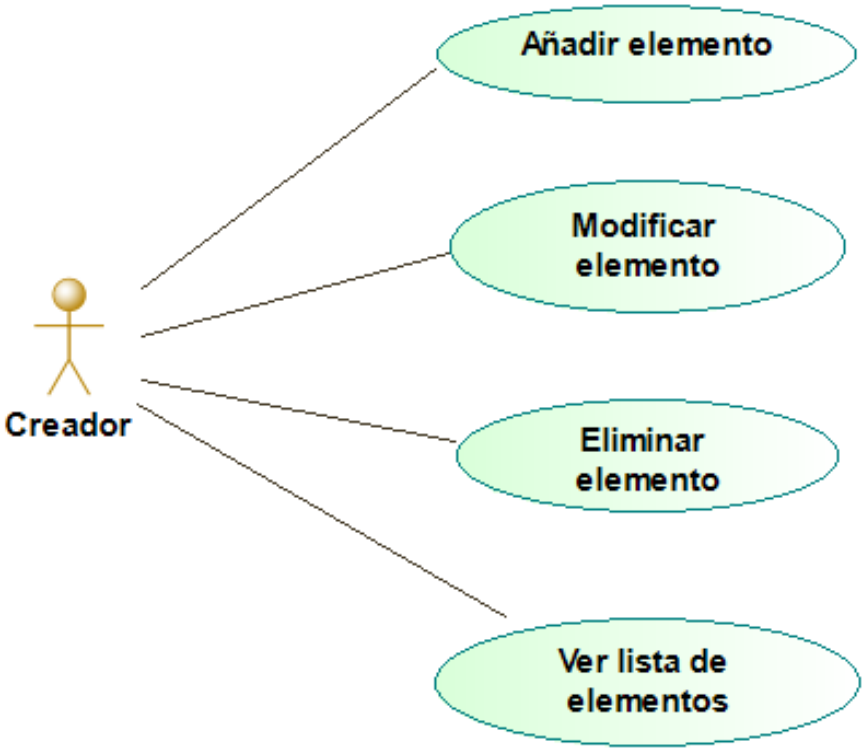


Figura 3.3: Módulo base de conocimiento

3.3. Diagramas de flujo

3.3.1. Búsqueda de un elemento

Una vez el paciente ha podido autenticarse, ya sea con su ultima sesión que se guarda automáticamente o metiendo sus credenciales, se cargara la vista de paciente. Esta vista tiene dos opciones de búsqueda cada una con su botón. Si se selecciona el botón de búsqueda por imagen, se abrirá la cámara automáticamente. El paciente debe entonces sacar una foto, esta foto se enviara a la plataforma cloud de Google a través de una llamada a la API, y nos devolverá todas las etiquetas de posibles cosas que detecta en la foto. La aplicación cogerá la etiqueta con el mayor porcentaje de certeza y se hará una búsqueda en la base de conocimiento con esa etiqueta. Si la llamada a la API no devolviera ninguna etiqueta se le pedirá al paciente que saque otra foto. Ahora si la etiqueta es encontrada en algún elemento de la base de conocimiento se enseñara la información asociada, si no se mostrara un mensaje de error y se le pedirá al paciente que saque otra foto.

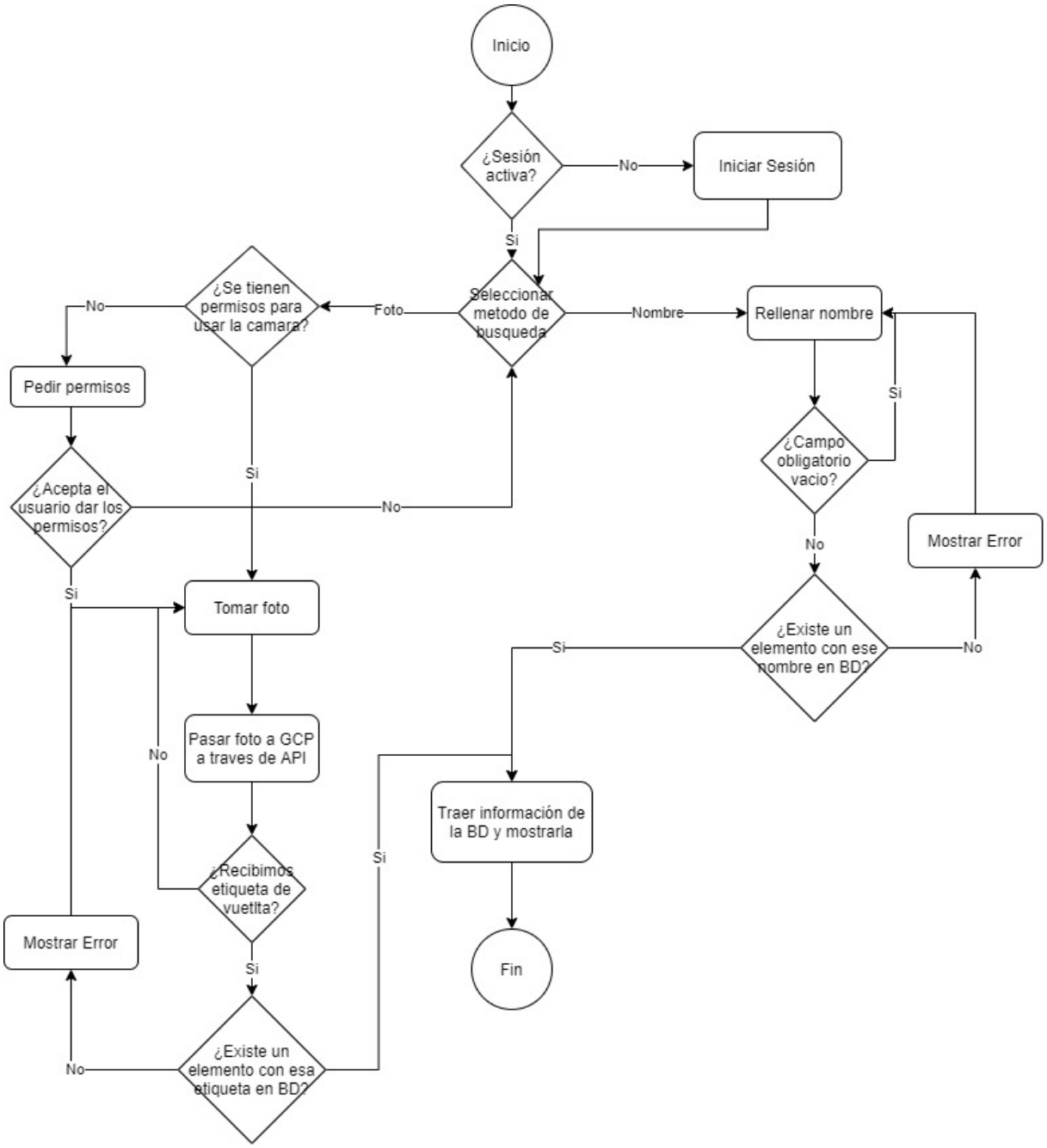


Figura 3.4: Diagrama de flujo de la búsqueda de un elemento por un paciente

3.3.2. Añadir un elemento

Una vez que el creador ha podido autenticarse, se le presentara la vista de creador en la que podrá ver todos los elementos que ya están en la base de conocimiento. Desde aquí deberá apretar el botón de 'añadir' y se ira a la vista que le pedirá la información requerida para crear un nuevo elemento. Desde aquí el creador deberá rellenar los campos obligatorios (nombre y descripción) y deberá subir una foto mediante una captura en ese momento o podrá seleccionar una foto ya echa desde su galería. Da igual que opción seleccione se tendrán que comprobar que los permisos están dados. Desde aquí se cargara la foto en la vista y se pasara a GCP a través de una API para que nos devuelva una etiqueta de esa imagen. Entonces mostraremos la etiqueta y el elemento estará listo para ser subido, se apretara el botón de subir y habremos terminado, podremos ver ese elemento ahora en el listado de elementos.

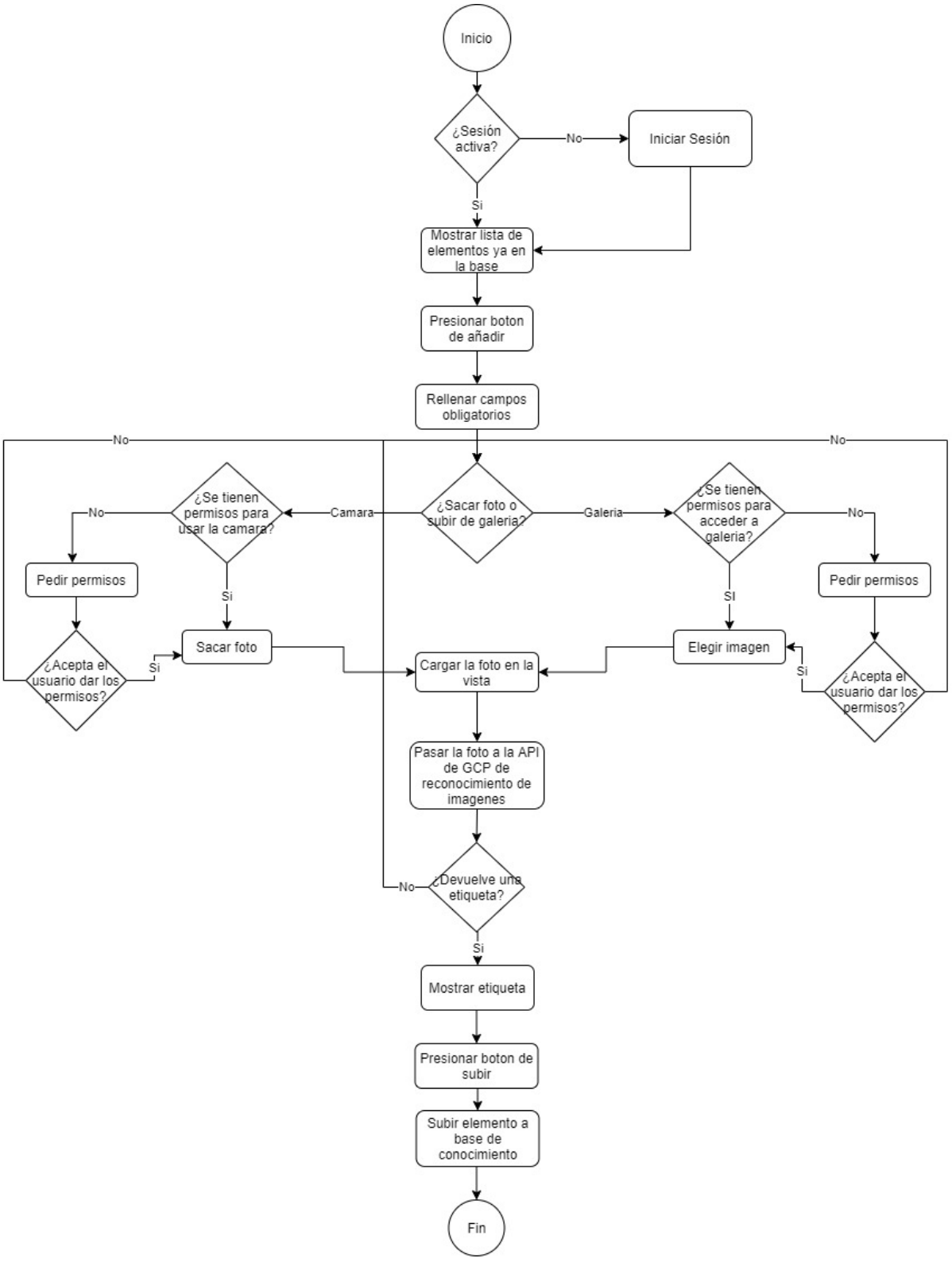


Figura 3.5: Diagrama de flujo del proceso para subir un nuevo elemento a la base de conocimiento

3.3.3. Validar un usuario

Una vez el administrador ha podido autenticarse, se mostrara todo el listado de usuarios que se han registrado, y el campo de validación de cada uno que sera 0 o 1. El administrador podrá presionar sobre un usuario no autenticado (valor 0) y decidir si lo valida o no. En caso de querer validarlo cambiara el valor del campo de validación a un 1, en caso de no querer validarlo podrá dejarlo como 0, o eliminar el usuario completamente.

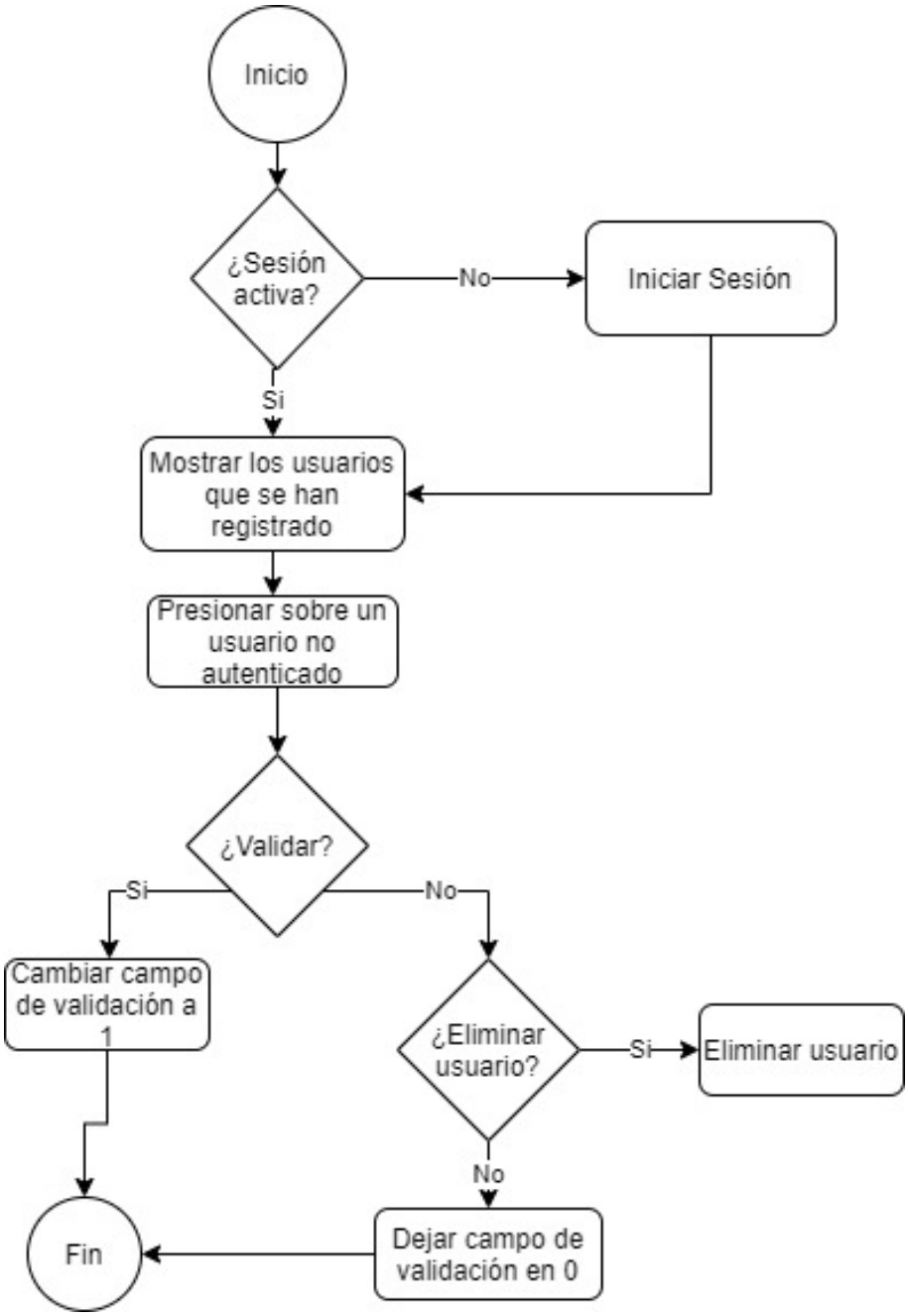


Figura 3.6: Diagrama de flujo del proceso para autenticar un usuario como administrador

3.4. Análisis de requisitos

Aquí se hará el análisis de requisitos funcionales y no funcionales de la aplicación

- Los requisitos funcionales son los que describen como los usuarios podrán interactuar con la aplicación y que funcionalidades deberá ofrecer la aplicación.
- Los requisitos no funcionales describen cosas como las tecnologías a usar para el desarrollo o que plataformas usar.

3.4.1. Requisitos funcionales

- Un usuario no registrado podrá solicitar su registro especificando un email, contraseña y el tipo de cuenta que quiere (paciente, creador)
- El administrador podrá validar a los usuarios y darles acceso
- Cualquier usuario que ha sido validado podrá acceder a la aplicación
- Cualquier usuario podrá cerrar sesión de la aplicación
- Cualquier usuario podrá acceder directamente a la aplicación si tiene una sesión activa en la aplicación
- El administrador podrá ver un listado de todos los usuarios registrados
- El administrador podrá modificar los datos de cualquiera de los usuarios
- El administrador podrá validar y des-validar a los usuarios que vea conveniente
- El administrador podrá eliminar a un usuario
- El creador podrá ver todos los elementos de la base de conocimiento
- El creador podrá añadir elementos a la base de conocimiento
- El creador podrá modificar elementos de la base de conocimiento
- El creador podrá eliminar elementos de la base de conocimiento
- El creador podrá añadir elementos a la base de conocimiento
- El creador podrá aportar la foto a los elementos tomando una foto a través de la cámara o navegando en la galería y subiendo una foto tomada previamente
- El creador podrá seleccionar los elementos que quiera y crear tests personalizados para el paciente
- El paciente podrá realizar una búsqueda mediante la toma de una foto y recibir información a cambio
- El paciente podrá realizar una búsqueda mediante el nombre de un elemento y recibir información a cambio
- El paciente podrá ver un listado de los tests personalizados que el creador ha preparado para el
- El paciente podrá jugar a uno de los tests personalizados que elija

- El paciente podrá jugar a un test generado aleatoriamente con los elementos de la base de conocimiento
- El paciente podrá ver el resultado de su tests al finalizarlo y decidir si jugarlo otra vez o no

3.4.2. Requisitos no funcionales

- La aplicación se desarrollará para Android exclusivamente
- El desarrollo se llevara acabo en el IDE Android Studio
- Se utilizara el lenguaje de programación de Java para todo el desarollo
- Se utilizara XML y el editor de Android Studio para el diseño de las diferentes pantallas de la aplicación
- Para la autenticación de la aplicación usaremos los servicios de Firebase
- Para el almacenamiento de los elementos, los usuarios y los tests usaremos Firestore database
- Para almacenar las imágenes y poder tenerlas conectadas a los elementos en Firestore utilizaremos Firebase Storage
- Para el etiquetado de imágenes usaremos la API de Cloud Vision de GCP que llamaremos a través del servicio de Functions de Firebase
- El control de versiones de la aplicción se hara con Github

Capítulo 4

Tecnologías

4.1. Android Studio [1]

Android Studio es el IDE mas popular para el desarrollo de aplicaciones Android, esto se debe en parte a que es el IDE que mantiene Google, los creadores de todo el ecosistema Android. Por supuesto que también destaca por sus innumerables funcionalidades. Lo primero es que permite al usuario desarrollar tanto en Java como en Kotlin(una versión de Java optimizada por Google para el desarrollo Android). También tiene un editador interactivo del diseño visual de nuestras vistas, usando un archivo XML nos permite modificar nuestras vistas tanto modificando el código como interactuando directamente con los elementos que coloquemos en nuestras vistas. Otra funcionalidad muy útil sobre todo cuando tienes aplicaciones de mayor escala es el analizador de APK, que te permite analizar sitios donde puedes reducir el tamaño de tus aplicaciones en cuanto a almacenamiento. Similarmente proporciona análisis en directo del uso que hace tu aplicación de recursos como CPU, memoria o actividad de red, lo cual facilita mucho su optimización. Por ultimo destacar la funcionalidad mas útil que seria el emulador. Android Studio te permite probar tu aplicación en terminales de diferentes tamaños, diferentes características en cuanto a CPU, RAM etc. El emulador siempre funciona de forma rápida y responde bien a todos los inputs. Aun así si quieres probar tu aplicación directamente en un terminal Android también te permite conectarte por cable y debugear tu aplicación mientras la utilizas en tu dispositivo.

4.2. Java [2]

El lenguaje de programación Java aunque es bastante longevo sigue siendo muy potente y ampliamente utilizado a día de hoy. Probablemente por las características que ofrece y por su moderada complejidad. Todo el lenguaje esta orientado al paradigma de los objetos por lo que en un entorno de desarrollo de aplicaciones lo hace muy atractivo. Probablemente lo mas importante es que cuando hablamos de una aplicación móvil es que es independiente a la plataforma lo que nos permite ejecutarlo sobre cualquier tipo de hardware que tenga un ecosistema Android. También destaca por su seguridad y robustez, y por el soporte a los programas multi hilo lo cual nos permite hacer ejecuciones simultaneas en los diferentes cores y mejorar el rendimiento.

4.3. XML

El XML es un lenguaje de marcado que nos ayuda mediante la definición de unas reglas a codificar documentos. Como comentamos antes es lo que usa Android Studio para facilitarnos el diseño de las diferentes vistas de las aplicaciones. Sus principales virtudes son la simplicidad y el fácil uso por lo que esta bastante extendido en muchos servicios web. Cada elemento que posicionamos en nuestras vistas es un elemento contenido entre marcas y sus atributos especifican sus características y comportamientos.

4.4. Firebase

4.4.1. Authentication [3]

La gran mayoría de las aplicaciones necesitan saber la identidad de un usuario, ya que permite guardar y dar acceso a información personal de cada usuario y a proporcionar una experiencia personalizada a cada usuario de la aplicación. La autenticación de Firebase proporciona los servicios backend que se acceden a través de llamadas a la API para autenticar usuarios de muchas formas distintas como podría ser con email y contraseña, teléfono móvil o a través de terceros como Google, Facebook etc. Una de las principales ventajas que tiene utilizar la autenticación de Firebase es que están fuertemente conectadas con todos los demás servicios de Firebase que también se aprovecharan para el desarrollo de la aplicación.

Para poder autenticar a un usuario en la app primero se requieren sus credenciales, después se pasan a al SDK de autenticación, el backend de Firebase verificará los datos y dará una respuesta al cliente para darle acceso o no a la app. Una vez autenticados se podrá acceder a la información del perfil del usuario y controlar el acceso que tiene el usuario a otros servicios de Firebase.

4.4.2. Cloud Firestore [4]

Como es evidente para el desarrollo de la aplicación se necesitara una base de datos, mas concretamente se usara Cloud Firestore, una base de datos flexible y escalable NoSQL para guardar y sincronizar los datos de los usuarios y de la base de conocimiento. Parte del poder de Firestore es que mantiene todos los datos sincronizados para todos los usuarios a través de listeneres en tiempo real y ofrece soporte para uso offline para que tus aplicaciones funcionen bien independientemente de la cobertura que tengas. Como la autenticación tiene una integración muy fácil con los demás servicios de Firebase y de Google Cloud como se vera luego con la integración que se hace con Google Cloud Vision.

Al ser una base de datos NoSQL se guardan los datos en documentos que contienen campos que posteriormente se mapean a valores. Estos documentos se guardan en colecciones que ayudan a organizar los documentos y posteriormente lanzar consultas. Los documentos son compatibles con muchísimos tipos de datos y son muy flexibles en cuanto a poder personalizarlos para amoldarse a tus necesidades.

Las consultas en Firestore son muy potentes ya que permiten controlar exactamente que se extrae, si solo un documento, o la colección entera, se puede ordenar los datos, filtrarlos poner limites etc. Para ayudar en la sincronización de datos, ofrece un sistema de “realtime

listeners" que avisan cuando se han modificado datos y solo actualiza los cambios sin necesidad de volver a extraer la base de datos entera.

4.4.3. Cloud Storage [5]

Este servicio de Firebase está diseñado para almacenar contenido generado por los usuarios como podría ser imágenes o videos. Para este proyecto se usara para almacenar las imágenes que los creadores asocian a los elementos de la base de conocimiento. Como con los otros servicios hay que conectarse a través de los SDKs de Firebase y se puede conectar los elementos en Firestore a las fotos en storage a través del uso de una URI. Además, este servicio aporta flexibilidad con la conexión de usuarios, al tratarse de archivos mas grandes, el usuario puede intentar de nuevo una operación desde donde paro la ultima vez en caso de fallo.

Los archivos son guardados en un bucket de Google Cloud lo cual hace a los archivos accesibles desde cualquier servicio de Firebase. El servicio también escala automáticamente proporcionándote justo los recursos que necesitas en cada momento. Como podríamos imaginarnos solo usuarios autenticados tienen los permisos necesarios para subir y descargar archivos a Cloud Storage.

4.4.4. Cloud Functions [6]

Cloud Functions es un framework "serverless" que permite ejecutar código automáticamente en respuesta a eventos lanzados por funcionalidades de Firebase o por solicitudes HTTPS. El código de JavaScript o TypeScript se guarda en la nube de Google y se ejecuta en un entorno controlado, no hay necesidad de manejar los servidores. Una vez se hace el despliegue, los servidores de Google manejan la ejecución de la función. Podemos activar la función con una solicitud HTTP o generarse asociada a ciertos eventos que se detectaran automáticamente. Aquí también se tienen los beneficios de ejecutar todo en la nube ya que según las demandas de nuestra función incrementan o decrementan los recursos se ajustarán automáticamente. También cabe resaltar que cada instancia de la función se ejecuta en un entorno personalizado para anticiparnos a cualquier posible conflicto. En este proyecto se usara este servicio para lanzar las llamadas a la API de Cloud Vision, cuando se detecten ciertos triggers por entradas de los usuarios, functions se encargará de lanzar el código y recibir la respuesta.

4.5. Google Cloud Vision [7]

Cloud Vision de Google proporciona un servicio a través de su plataforma cloud para todo tipo de reconocimiento de imágenes y detección de objetos. Permite usar sus modelos ya entrenados o incluso te ayudan a entrenar modelos personalizados para tus necesidades. En el caso de este trabajo se usara específicamente la API de Vision, esta API permite detectar y extraer información sobre la imagen y devuelve etiquetas dentro de un rango de categorías. Las etiquetas pueden detectar objetos, lugares, animales, actividades etc. La API devuelve todas las etiquetas que ha detectado y el porcentaje de certeza que tiene el modelo de que esa etiqueta esta representada en la imagen. Esto se usara para etiquetar todos los elementos de la base de conocimiento cogiendo su etiqueta más fiable. También en la búsqueda de los pacientes por imagen, la imagen subida por los pacientes será pasada

a la API y se guardara su etiqueta más fiable, posteriormente se hará la búsqueda en la base de conocimiento en Firestore utilizando la etiqueta guardada.

Cabe destacar que como en este caso no se entrena un modelo especializado en reconocer objetos, se hizo una mejora en el plan de Google cloud y se paso a un plan de pago según consumo (plan Blaze). Las ventajas de este plan son entre otras cosas que los modelos que usa ahora la API para detectar categorías pasan de tener 400 categorías a mas de 9000, por lo que debería poder aportar una etiqueta mucho más fiable. Por suerte este plan ofrece las primeras 1000 peticiones de la API al mes gratis, por lo que se pudo debugear la aplicación sin ningún coste, pero sería un factor a tener en cuenta si la aplicación se lanzara al público.

4.6. Github

GitHub es el sistema de control de versiones por excelencia. En cualquier proyecto que se haga que empiece a tener un tamaño considerable es absolutamente necesario usar un sistema de control de versiones. No solo permite tener una copia de seguridad del código en la nube, sino que también permite ver como se han ido implementando funcionalidades nuevas, hacer rollbacks en el caso de que quiera revertir a una versión antigua del código y permite ver el progreso de cómo va avanzando el proyecto. Por supuesto tiene funcionalidades claves para los equipos de desarrollo permitiendo a cada usuario crear sus ramas y poder controlar luego los commits y los merges en la rama master. Podemos de esta forma ver quien aporta que al proyecto y detectar y resolver conflictos antes de causar un problema en la rama máster del proyecto. GitHub tiene una comunidad enorme de programadores open source aunque te permite también poner las licencias que creas conveniente a tu código.

Capítulo 5

Modelo de datos

5.1. Diagrama de relaciones

Como se ha explicado en el apartado anterior, la base de datos sera Cloud Firestore que tiene como característica principal el ser NoSQL. Esto significa que no se puede dibujar un diagrama de entidad relación clásico ya que las reglas mas relajadas sobre las relaciones en el paradigma NoSQL no lo requieren. Pero si que se pueden plasmar las relaciones y mostrar los atributos de cada tipo de fichero.

Fijándonos en la Figure 5.1 se puede observar que las relaciones entre los diferentes tipos de ficheros son bastante simples. Desde el punto de vista del usuario se puede ver que tiene una relación de 1 a muchos con los otros dos tipos de ficheros ya que cada usuario tendrá en su base de conocimiento personalizada todos los elementos que quiera meterle el creador sin ningún tipo de restricción, y lo mismo pasa con los tests, esta en la mano del creador preparar todos los tests que vea necesario. En cuanto a la relación entre los tests y los elementos de la base de conocimiento también hay una relación de 1 test a muchos elementos, ya que como mínimo cada test contendrá tres elementos sobre los que se le preguntara posteriormente al paciente. Como se explico en la parte de tecnología Firestore permite agrupar cada tipo de fichero en una colección para la mejor organización y facilitar las consultas, y es exactamente lo que se ha echo en este caso.

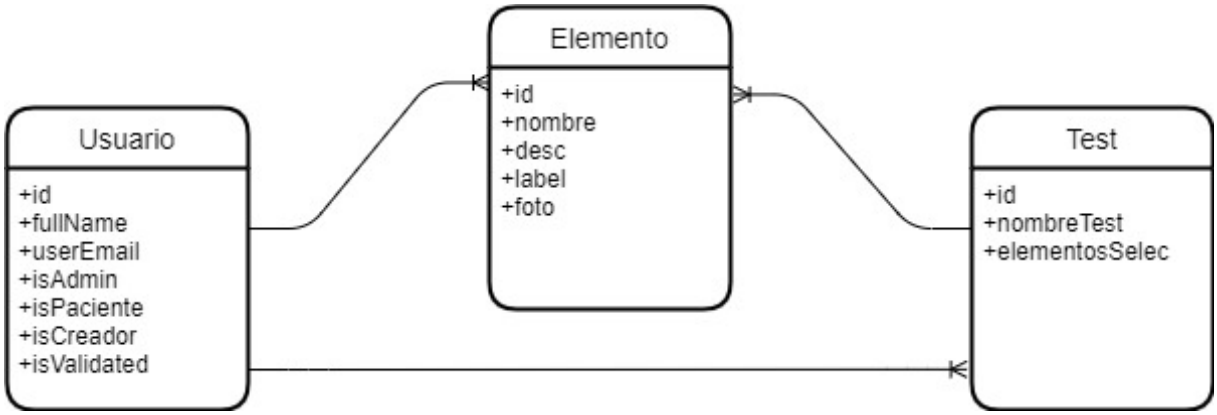


Figura 5.1: Diagrama de relaciones de nuestra base de datos NoSQL

5.2. Tipos de ficheros

5.2.1. Fichero Usuario

Este tipo de fichero sera utilizado para almacenar la información necesaria para la autenticación de los usuarios. Se usara su email como clave única de inicio de sesión y luego diversos parámetros tipo bool para el proceso de autenticación y para diferenciar los tipos de usuarios y poder dar acceso a las partes de la aplicación que le corresponden a cada uno.

Usuario	
id	Identificador único generado por Firestore para diferenciar cada fichero creado.
fullName	Nombre completo del usuario (nombre y apellidos). Este campo es de tipo String.
userEmail	El email personal de cada usuario, no puede haber dos usuarios registrados con el mismo email por lo que son unicos. Este campo es de tipo string.
isAdmin	Campo tipo bool para indicar si un usuario es admin o no. False = no admin, True = admin. Un usuario solo puede ser True en uno de los campos isAdmin, isPaciente y isCreador.
isPaciente	Campo tipo bool para indicar si un usuario es paciente o no. False = no paciente, True = paciente.
isCreador	Campo tipo bool para indicar si un usuario es creador o no. False = no creador, True = creador.
isValidated	Campo tipo bool para indicar si un usuario ha sido validado o no por el admin. False = no validado, True= validado.

Cuadro 5.1: Explicación de atributos para el tipo de fichero Usuario.

5.2.2. Fichero Elemento

Este tipo de fichero se usara para almacenar toda la información que debemos guardar para cada objeto que se meta en la base de conocimiento. Los atributos metidos serán los que luego el paciente podrá explotar, como el nombre, la descripción o la foto asociada. La etiqueta también juega un papel crucial ya que es lo que luego permitirá a los pacientes hacer la búsqueda por imagen.

Elemento	
id	Identificador único generado por Firestore para diferenciar cada fichero creado.
nombre	Nombre usado por el paciente para llamar al objeto. Este campo es de tipo String y debe ser único ya que luego se harán búsquedas basándonos en este campo.
desc	Descripción del objeto pensada en que el paciente pueda entender fácilmente como usar el objeto o que hace. Este campo es de tipo string.
label	Este campo se usa para almacenar la etiqueta que nos devuelve la API de Vision de Google. También debe ser unica ya que haremos búsquedas basándonos en este atributo. Este campo es de tipo String.
foto	Campo usado para almacenar la URI de cada imagen, es generado por Cloud Storage y la vinculamos de esta forma con nuestra base de datos en Firestore. Este campo es de tipo String.

Cuadro 5.2: Explicación de atributos para el tipo de fichero Elemento.

5.2.3. Fichero Test

Este tipo de fichero se usara para almacenar las selecciones de elementos que haga el creador en cada test personalizado. Adicionalmente se almacenara un nombre que deberá ayudar al paciente a reconocer los contenidos de ese test para poder elegir el correcto a realizar.

Test	
id	Identificador único generado por Firestore para diferenciar cada fichero creado.
nombreTest	Nombre del test que ayudara al paciente identificar sus contenidos. Este campo es de tipo String y debe ser unico.
elementosSelec	Este campo es un array de elementos, una vez el creador hace la selección de que elementos entran en cada tests, se guardan los elementos enteros ya que necesitaremos casi todos sus atributos luego para generar los tests. Cada test deberá contener mínimo 3 elementos.

Cuadro 5.3: Explicación de atributos para el tipo de fichero Test.

Capítulo 6

Implementación

6.1. Elementos Android

6.1.1. Adapter

El patrón adapter es una pieza clave para implementar un RecyclerView, permite asociar los datos con los ViewHolders. El adapter tiene el rol de generar tantos nuevos ViewHolders como sean necesarios, colocarlos y luego fijar los datos correctos que demande la vista que se haya generado.

6.1.2. RecyclerView [8]

Esta vista hace mostrar listas de datos de una manera simple y eficiente muy facil. El trabajo del desarrollador es dar los datos y definir como cada elemento se debería ver, y el RecyclerView crea los elementos dinámicamente cuando se necesitan.

Lo que da nombre a la vista es la funcionalidad que tiene de reciclar elementos. Cuando un elemento desaparece de la pantalla, la vista no lo destruye, si no que reutiliza la vista creada y reemplaza los datos con los elementos que ahora deberían estar en pantalla. Esta reutilización mejora mucho la eficiencia y hace la aplicación mas ágil.

En este proyecto se usara para crear listas dinámicas de usuarios, elementos y de tests.

6.1.3. Toast [9]

Toast es una manera simple de dar feedback sobre una acción que el usuario ejecuta en un pequeño mensaje de popup. Solo ocupa el texto requerido para que entre el texto y la actividad actual sigue visible y operativa. Los Toasts desaparecen automáticamente después de un tiempo.

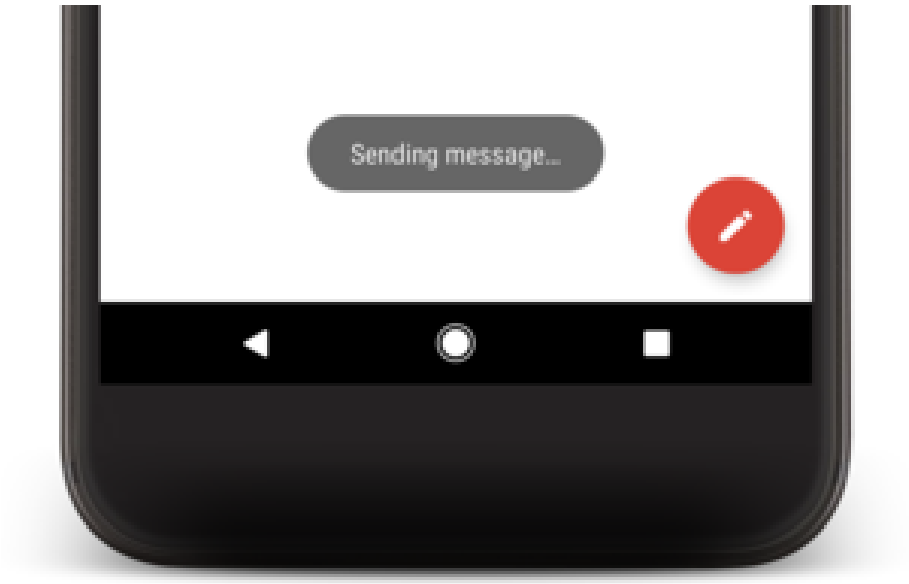


Figura 6.1: Un ejemplo de un Toast despues de apretar un boton de enviar

6.1.4. Dialogs [10]

Un dialog es una ventana pequeña que incita al usuario a tomar una decisión o a rellenar información adicional. Un dialog no ocupa la pantalla entera sino que aparece encima de la actividad actual y suele ser necesario completar la acción que te piden antes de continuar usando la aplicación. En este trabajo se usara para pedir por ejemplo el nombre del objeto que se busca al paciente o al creador para nombrar los tests que crea.

6.1.5. Picasso [11]

La librería Picasso permite cargar fotos en las vistas de la aplicación de una forma sencilla, casi siempre con una única linea de código. La librería se encarga de todo el proceso de de reciclado de vistas, transformaciones de tamaños y el cacheo en memoria y disco.

6.1.6. Constraint Layout [12]

Constraint layout es un formato que se puede usar para los archivos de diseño XML. Este layout permite crear vistas complejas con muchos elementos de una forma sencilla, las vistas se organizan de forma que la posición de cada componente se basa en relación al resto de componentes y a los bordes de la vista. Este tipo de layout explota al 100 % las funcionalidades de Android Studio de edición visual. Poder hacer un diseño drag and drop ayuda mucho para poder ir viendo donde cae cada cosa y como queda relativo a los demás elementos. Todos los elementos deben tener una fijación horizontal y otra vertical como mínimo, aunque se recomienda siempre poner el máximo posible.

6.1.7. Gestos

Android da soporte a varios gestos para facilitar la interacción entre el usuario y las aplicaciones, en este caso se podrán usar las siguientes:

- **Toque** - Se usa para navegar por la aplicación e ir pulsando botones o cambiar

donde se quiere escribir etc. Hay muchos listeners por toda la aplicación esperando toques en los elementos.

- **Toque largo** - El toque largo se hace manteniendo el dedo durante un tiempo mas largo en el mismo sitio. En la aplicación solo se usa en al vista de creadores para comenzar una selección de elementos para crear un test. Al igual que con el toque administramos el flujo de ejecución con listeners.
- **Deslizar** - Dependiendo donde se haga el desliz y en que dirección tendrá un efecto u otro. Por ejemplo si deslizamos verticalmente en una lista pues se ira viendo el resto de elementos, pero si se desliza horizontalmente desde el borde izquierdo de la pantalla se ira atrás en cuanto al flujo de la aplicación se refiere (la vista anterior, si no hubiera vista anterior se cerraría la aplicación).

6.2. Paquetes

6.2.1. Paquete Activities

6.2.1.1. `LoginActivity`

En esta actividad se gestiona toda la lógica asociada a autenticarse para poder acceder a la aplicación. Esta actividad es la que esta marcada como la inicial de toda la aplicación, así que el código de esta actividad es lo primero en ejecutarse al abrir la aplicación. Una de las funcionalidades que tiene esta actividad que esta pensada primordialmente pensando en el paciente, es que no te obliga siempre a meter tu email y contraseña, si tienes una sesión ya abierta en la aplicación eres autenticado directamente. Con esto se busca facilitar a paciente el acceso y que nunca se quede fuera por olvidarse de alguno de los elementos necesarios para la autenticación. Esto se hace sobre escribiendo el método onStart que se ejecuta al iniciar la aplicación y comprobando si hay algún usuario activo en este dispositivo, si la respuesta es positiva se manda a su vista correspondiente (paciente, creador o admin) y si no la hay se comienza la actividad de login y se carga la vista.

```
@Override
protected void onStart() { //if already logged in
    super.onStart();
    if(FirebaseAuth.getInstance().getCurrentUser() != null){
        DocumentReference df = FirebaseFirestore.getInstance().collection("Users").document(FirebaseAuth.getInstance().getCurrentUser().getUid());
        df.get().addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
            @Override
            public void onSuccess(DocumentSnapshot documentSnapshot) {
                if(documentSnapshot.getString("isAdmin").equals("1")){
                    startActivity(new Intent(getApplicationContext(), AdminActivity.class));
                    finish();
                }
                if(documentSnapshot.getString("isCreador").equals("1")){
                    startActivity(new Intent(getApplicationContext(), CreadorActivity.class));
                    finish();
                }
                if(documentSnapshot.getString("isPaciente").equals("1")){
                    startActivity(new Intent(getApplicationContext(), PacienteActivity.class));
                    finish();
                }

            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                FirebaseAuth.getInstance().signOut();
                startActivity(new Intent(getApplicationContext(), LoginActivity.class));
                finish();
            }
        });
    }
```

Figura 6.2: Método onStart de la clase `LoginActivity`

En la vista de login hay cuatro elementos, dos de texto para el email y la contraseña, y dos botones uno para intentar el acceso y otro para moverse a la vista de registro. Lo interesante se encuentra en lo que se hace cuando se presiona el botón de acceso, lo primero que se hace es comprobar que los campos de email y contraseña no están vacíos, en caso de que lo estén se envía un Toast de error y no se intenta la autenticación. En caso de que se hayan metido datos se cogerá la instancia de Firebase Authentication y se intentara hacer un acceso con email y contraseña. Si la autenticación va bien devolverá un id único del usuario con el que ahora se podra hacer una consulta a Firestore y recopilar sus datos para comprobar su validación. Si el usuario ya esta validado se le redirecciónа a su vista correspondiente, en el caso de que no lo este se manda un toast de error y se cierra su sesión en Firebase. Si desde un principio no se ha podido autenticar al usuario se envía otro Toast indicando el error.

```
loginBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        checkField(email);
        checkField(password);


        if(valid){
            fAuth.signInWithEmailAndPassword(email.getText().toString(), password.getText().toString()).addOnSuccessListener(new OnSuccessListener<AuthResult>() {
                @Override
                public void onSuccess(AuthResult authResult) {
                    String uid = authResult.getUser().getUid();
                    DocumentReference df = fStore.collection("Users").document(uid);
                    df.get().addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {

                        @Override
                        public void onSuccess(DocumentSnapshot documentSnapshot) {
                            if(documentSnapshot.getString("isValidated").equals("1")){
                                Toast.makeText(LoginActivity.this, "Logged in Succesfully", Toast.LENGTH_SHORT).show();
                                checkUserAccessLevel(documentSnapshot.getId());
                            }
                            else{
                                Toast.makeText(LoginActivity.this, "User not validated", Toast.LENGTH_SHORT).show();
                                FirebaseAuth.getInstance().signOut();
```

Figura 6.3: onClick listener del botón de acceso

6.2.1.2. `RegisterActivity`

En esta actividad se gestiona todo lo que gira en torno al registro de nuevos usuarios. Para poder dar de alta un nuevo usuario se necesitan los siguientes campos; nombre completo, email, contraseña y el tipo de cuenta. Esta vista también tiene dos botones, uno para realizar el registro y otro para volver a la vista de acceso. En cuanto a las casillas para seleccionar el tipo de usuario se ha implementado una logica que solo permita tener uno presionado, en caso de que ya haya uno presionado y se presione el otro, el inicial sera de seleccionado.

Una vez el nuevo usuario ha proporcionado todos los datos y estos campos han sido comprobados se pasa al registro del usuario. Se vuelve a usar la instancia que hay de Firebase Authentication y se usa un método de creación de usuario con email y contraseña. En caso de éxito se debe ahora guardar los datos de este nuevo usuario en la base de datos Firestore. Por lo que se coge el id único que ha generado el servicio de autenticación y se crea un fichero nuevo de usuario con todos los datos proporcionados y este id. Una vez echo esto, se cierra la sesión del usuario para que la pantalla de login no lo reconozca como una sesión activa y le obligue a esperar a ser validado. Después se direcciona al nuevo usuario a la pagina de acceso ya que tiene que esperar a ser validado.

```
fAuth.createUserWithEmailAndPassword(email.getText().toString(), password.getText().toString()).addOnSuccessListener(new OnSuccessListener<AuthResult>() {
    @Override
    public void onSuccess(AuthResult authResult) {
        FirebaseUser user = fAuth.getCurrentUser();
        Toast.makeText(RegisterActivity.this, "Account Created", Toast.LENGTH_SHORT).show();
        DocumentReference df = fStore.collection("Users").document(user.getUid());
        Map<String, Object> userInfo = new HashMap<>();
        userInfo.put("fullName", fullName.getText().toString());
        userInfo.put("userEmail", email.getText().toString());
        userInfo.put("isValidated", "0");

        if(isCreador.isChecked()){
            userInfo.put("isCreador", "1");
        }
        if(isPaciente.isChecked()){
            userInfo.put("isPaciente", "1");
        }

        df.set(userInfo);
```

Figura 6.4: Proceso de registro en la clase `RegisterActivity`

6.2.1.3. `AdminActivity`

La vista de admin es muy simple, solo tiene dos elementos, el RecyclerView de los usuarios y un botón para cerrar sesión. Para configurar el RecyclerView hay que seguir los siguientes pasos muy simples. Lo primero hay que encontrar el diseño de la vista que se ha creado para esta vista y fijar su tamaño y su layout manager. Ahora se inicializa el adaptador y se le asigna al elemento recyclerView, aquí también se inicializa la lista que se ira llenando con los elementos que se quieran enseñar. Una vez hecho esto se coge la colección entera de usuarios de Firestore y se va iterando sobre los elementos añadiéndolos uno a uno a la lista. Cuando se haya cargado todo en la lista, se notifica al adapter para que pueda ir fijando los datos en los elementos deseados.

```
recyclerView = findViewById(R.id.recyclerview_users);
recyclerView.setHasFixedSize(true);
recyclerView.setLayoutManager(new LinearLayoutManager(this));

userList = new ArrayList<>();
adapter = new UsersAdapter(this, userList);

recyclerView.setAdapter(adapter);


db = FirebaseFirestore.getInstance();


db.collection("Users").get()
        .addOnSuccessListener(new OnSuccessListener<QuerySnapshot>() {
            @Override
            public void onSuccess(QuerySnapshot queryDocumentSnapshots) {


                if(!queryDocumentSnapshots.isEmpty()){

                    List<DocumentSnapshot> list = queryDocumentSnapshots.getDocuments();

                    for(DocumentSnapshot d : list){

                        User u = d.toObject(User.class);
                        u.setId(d.getId());
                        userList.add(u);

                    }

                    adapter.notifyDataSetChanged();

                }
```

Figura 6.5: Proceso de configuración del RecyclerView en la clase `AdminActivity`

6.2.1.4. `UpdateUserActivity`

Esta vista sirve para que el administrador, sobre todo, valide usuarios después de revisar los datos. Tiene 6 campos de texto modificables que nada mas cargar la vista se rellena con los datos del usuario que se ha presionado en el RecyclerView. Se sabe que usuario es porque el adapter lo pasa. Una vez echas las modificaciones necesarias se presiona el botón de modificar. Esto crea un nuevo objeto user con el contenido de los campos de texto en ese momento y sube el nuevo fichero a la base de datos de Firestore utilizando el id del elemento como clave para reemplazar los datos antiguos. Inmediatamente se vuelve a la vista del admin y se podrán ver los nuevos cambios en la lista de usuarios. Tambien

esta implementada la lógica para eliminar usuarios. Aunque realmente no se elimina de el servicio de autenticación de firebase ya que permitir a un usuario eliminar la cuenta de otro sabiendo su UID seria una brecha de seguridad muy gorda. Lo que hace es eliminar su fichero de la colección de usuarios de Firebase. Al final tiene el mismo resultado ya que al no estar en firebase el admin no podrá validarlo. Lo único que no podrá volver a intentar registrarse con el mismo email ya que si que estará registrado en los servicios de autenticación.

6.2.1.5. `CreadorActivity`

La actividad de creador tiene 4 componentes principales. La primera es un RecyclerView con todos los elementos de la base de conocimiento con la que podrá interactuar de diferentes formas. Después hay 3 botones, uno para cerrar la sesión otro para añadir un nuevo elemento a la base de conocimiento, y un tercero para confirmar una selección de elementos y guardarlo como test. La única lógica que esta contenida en esta actividad es la de el botón de guardar un test así que la explicaremos a continuación.

Una vez el botón de guardar selección ha sido presionado, lo primero que hace es llamar a un método del adapter del RecyclerView que devolverá un arraylist de elementos con solamente los elementos seleccionados. Ahora se lanza un dialog para que el creador pueda nombrar el test y guardarlo en Firestore. Cuando se presiona el botón de guardar test se hacen un par de validaciones previas, primero que haya un string en la entrada del nombre, y segundo que en el arraylist de elementos seleccionados haya al menos 3 elementos. En caso de que no se cumpla alguno de estos requisitos se darán avisos de error. Si todo esta en orden, se creara un nuevo fichero de tipo test en el que se meterá el nombre insertado y el arraylist y se subirá a la colección de tests de Firestore. Seguidamente se volverá a cargar la vista de creador para reiniciar todas las selecciones anteriores.

```
public void seleccionDialog() {
    final Dialog dialogSeleccion = new Dialog(CreadorActivity.this);
    dialogSeleccion.requestWindowFeature(Window.FEATURE_NO_TITLE);
    if (dialogSeleccion.getWindow() != null) {
        ColorDrawable colorDrawable = new ColorDrawable(Color.TRANSPARENT);
        dialogSeleccion.getWindow().setBackgroundDrawable(colorDrawable);
    }
    dialogSeleccion.setContentView(R.layout.dialog_seleccion);
    dialogSeleccion.setCancelable(false);
    dialogSeleccion.show();

    Button buttonGuardar = (Button) dialogSeleccion.findViewById(R.id.dialogSeleccion);
    EditText editTextSeleccion = (EditText) dialogSeleccion.findViewById(R.id.editTextSeleccion);
```

Figura 6.6: Creación del dialog en la clase `CreadorActivity`

6.2.1.6. `NewElementActivity`

Esta es una de las actividades mas importantes de toda la aplicación ya que contiene toda la lógica para crear un nuevo elemento en la base de conocimiento. El creador deberá rellenar los campos de nombre y descripción además de aportar una foto a través de una

captura en ese momento o a través de la selección en la galería. También habrá un campo con la etiqueta de la foto pero este se llenara automáticamente y no podrá ser modificado. A continuación se explicara el proceso de obtener la imagen y etiquetarla ya que es lo mas complejo.

Lo primero que se hace es pedir permisos de cámara, y de escritura y lectura en almacenamiento. Esto se hace recopilando los permisos que se quieren verificar en un array de strings y comprobando si están dados o no, en caso afirmativo se lanza un ïntent"que lanza o la interfaz de la cámara o abre la galería y en caso contrario se piden los permisos.

```
private void pedirPermisos() {
    Log.d("TAG", "VerifyingPermission : Asking for permission ");
    String[] permissions = {Manifest.permission.CAMERA, Manifest.permission.READ_EXTERNAL_STORAGE, Manifest.permission.WRITE_EXTERNAL_STORAGE};

    // index 0 = camera, index 1 = readStorage , index 2 = write Storage

    if (ContextCompat.checkSelfPermission(this.getApplicationContext(), permissions[0]) == PackageManager.PERMISSION_GRANTED
            && ContextCompat.checkSelfPermission(this.getApplicationContext(), permissions[1]) == PackageManager.PERMISSION_GRANTED
            && ContextCompat.checkSelfPermission(this.getApplicationContext(), permissions[2]) == PackageManager.PERMISSION_GRANTED) {

        dispatchTakePictureIntent();
    }
    else {
        ActivityCompat.requestPermissions(NewElementActivity.this, permissions, All_PERMS_CODE);
```

Figura 6.7: Método para pedir permisos en la clase `NewElementActivity`

Ya se ha lanzado la interfaz de la cámara y se ha tomado la foto o se ha seleccionado desde la galería. En caso de que la foto se acabe de tomar se debe ahora guardar la foto en almacenamiento y prepararla para pasarla a la API de visión. Da igual de donde haya venido la foto, una parte muy importante de este proceso es asignarle una URI a la foto, ya que sera lo que se use para identificar la imagen. Una vez asignada la URI se pasa al proceso de etiquetado. Para que la API pueda interpretar la imagen hay que hacer una serie de transformaciones. Lo primero es convertir la imagen a un bitmap y posteriormente a un string codificado en base64. Ahora hay que crear la request a cloud vision, que se hace en un objeto JSON. En este objeto JSON llamado request se van metiendo otros objetos JSON que contienen la imagen y las características sobre la petición. Esto es para indicar que queremos que nos devuelva la llamada.

```
private void labelImage(){
    try {
        Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), contentUri);
        // Convert bitmap to base64 encoded string
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.JPEG, 100, byteArrayOutputStream);
        byte[] imageBytes = byteArrayOutputStream.toByteArray();
        String base64encoded = Base64.encodeToString(imageBytes, Base64.NO_WRAP);

        // Create json request to cloud vision
        JsonObject request = new JsonObject();
        // Add image to request
        JsonObject image = new JsonObject();
        image.add("content", new JsonPrimitive(base64encoded));
        request.add("image", image);
        //Add features to the request
        JsonObject feature = new JsonObject();
        feature.add("maxResults", new JsonPrimitive(5));
        feature.add("type", new JsonPrimitive("LABEL_DETECTION"));
        JsonArray features = new JsonArray();
        features.add(feature);
        request.add("features", features);

        annotateImage(request.toString())
```

Figura 6.8: Método para etiquetar imágenes, preparación para llamada a la API en la clase `NewElementActivity`

Una vez se tiene el JSON listo se llama al método 'annotateImage'. Este método hace la llamada a la API de cloud vision y devuelve su contenido. Para hacer esta llamada se debe usar Cloud Functions para lanzar una petición HTTPS a los servidores de Google. Una vez se consigue la respuesta, se coge la etiqueta con mayor porcentaje de certeza y se fija en su campo de texto. Este campo no es modificable y solo esta ahí para que el creador compruebe que la etiqueta tiene sentido y evaluar si es mejor tomar otra foto.

```
private Task<JsonElement> annotateImage(String requestJson) {
    return mFunctions
            .getHttpsCallable("annotateImage")
            .call(requestJson)
            .continueWith(new Continuation<HttpsCallableResult, JsonElement>() {
                @Override
                public JsonElement then(@NonNull Task<HttpsCallableResult> task) {
                    // This continuation runs on either success or failure, but if the task
                    // has failed then getResult() will throw an Exception which will be
                    // propagated down.
                    return JsonParser.parseString(new Gson().toJson(task.getResult().getData()));
                }
```

Figura 6.9: Método que lanza la petición usando la API en la clase `NewElementActivity`

Lo único que quedaría ahora es subir el elemento a Firestore para poder usarlo mas adelante. Como se comento antes la imagen no puede ser subida a Firestore, sino que se tendrá que subir primero a Cloud Storage y guardar su URL de descarga, esto sera lo que se guarde en el fichero de Firestore y permitirá recuperar la foto. Una vez se tiene el URL creamos un nuevo objeto elemento que se usara como esqueleto del fichero y se rellenara con todos los datos que se han recopilado después de verificarlos. Ya solo quedaría subir el fichero a la colección que almacenara todos los ficheros de nuestra base de conocimiento.

```
private void uploadImageToFirebase(String name, Uri contentUri) {
    final StorageReference image = storageReference.child("images/" + name);
    image.putFile(contentUri).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
        @Override
        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
            image.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
                @Override
                public void onSuccess(Uri uri) {
                    Log.d("tag", "onSuccess: Uploaded Image Uri is " +  uri.toString());
                    String nombre = editTextNombre.getText().toString().trim();
                    String desc = editTextDesc.getText().toString().trim();
                    String label = textViewLabel.getText().toString().trim();
                    String image = uri.toString();

                    if(!validateInputs(nombre, desc, label, image)){

                        CollectionReference dbConocimiento = db.collection("Conocimiento");

                        Elemento elemento = new Elemento(nombre, desc, label, image);

                        dbConocimiento.add(elemento).addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
                            @Override
                            public void onSuccess(DocumentReference documentReference) {
                                Toast.makeText(NewElementActivity.this, "Añadido a base de conocimiento", Toast.LENGTH_SHORT).show();
                            }
                        }).addOnFailureListener(new OnFailureListener() {
                            @Override
                            public void onFailure(@NonNull Exception e) {
                                Toast.makeText(NewElementActivity.this, e.getMessage(), Toast.LENGTH_SHORT).show();
                            }
                        });

                        startActivity(new Intent(getApplicationContext(), CreadorActivity.class));
                    }
```

Figura 6.10: Método que prepara los datos y sube los elementos a Firestore en la clase `NewElementActivity`

6.2.1.7. `UpdateElementoActivity`

Esta actividad contiene mucha de la lógica que se acaba de explicar en NewElementActivity ya que si deseamos modificar la foto hay que seguir el mismo procedimiento. Lo que añade de funcionalidad es que permite modificar solo algunas partes del elemento y volver a subirlo y también permite eliminar el elemento entero en caso de ya no se necesite. Para modificar un elemento es la misma idea que con la modificación de usuarios. Se crea un objeto elemento nuevo con los nuevos datos, y en vez de subirlo como un nuevo elemento se hace una búsqueda del elemento que se quiere modificar usando su id y se reemplaza los atributos del fichero con los nuevos.

Para el proceso de eliminación se usa un tipo de dialog de alerta, .ªlertDialog"que te pregunta si estas seguro de que quieres eliminar el elemento y de que es permanente. En caso de elegir si, se hace una búsqueda en la colección usando el id del elemento y se elimina todo el fichero, y en caso de presionar no, simplemente se cierra el dialogo.

```
findViewById(R.id.button_eliminar).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        AlertDialog.Builder builder = new AlertDialog.Builder(UpdateElementoActivity.this);
        builder.setTitle("¿Estas seguro?");
        builder.setMessage("Eliminar es permanente");

        builder.setPositiveButton("Si", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                eliminarElemento();
            }
        });

        builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {

            }
        });

        AlertDialog ad = builder.create();
        ad.show();
```

Figura 6.11: AlertDialog usado en funcionalidad de eliminación de elemento en la clase `UpdateElementoActivity`

6.2.1.8. `PacienteActivity`

Esta Actividad se encarga de re direccionar al paciente a las vistas correspondientes y pasar los datos necesarios consigo. También incluye toda la lógica de tomar la foto y etiquetarla, que es la misma que la explicada anteriormente con el único cambio de que ahora la etiqueta que nos devuelve se guarda y se pasa a la actividad de `DatosEscanearActivity`. Para controlar este flujo la vista contiene 4 botones, uno para la búsqueda por imagen, otro para la búsqueda por nombre, otro para jugar, y uno ultimo para cerrar sesión. El de jugar simplemente te lleva a la actividad de selección de tests, y el de cerrar sesión cierra tu sesión en Firebase Authentication y te lleva a la actividad de login. Al presionar el botón de búsqueda por nombre se lanza un dialog para poder insertar el nombre a buscar y pasárselo a la actividad de DatosBuscarActivity que realizara la consulta en Firestore.

6.2.1.9. `DatosEscanearActivity`

En esta actividad simplemente se obtiene la etiqueta pasada, y se hace una búsqueda en la colección de conocimiento usando la etiqueta. En caso de que la búsqueda encuentre un fichero se mostraran el nombre y la descripción en una vista alternativa. En caso de que no se encuentre se enviara un mensaje de error. También se incluye un botón de .ªtras"para poder volver a la vista general del paciente.

6.2.1.10. `DatosBuscarActivity`

Esta actividad tiene exactamente la misma funcionalidad que la anterior, la única diferencia es que ahora la búsqueda se hace con el string que el paciente metió en el dialog en la vista anterior. Aquí también se enseña la foto del objeto ya que se supone que no lo tiene delante.

6.2.1.11. `JuegoSelecActivity`

Esta actividad brinda al paciente la capacidad de seleccionar que test es el que quiere realizar. Hay dos elementos en la vista, primero un botón de "Todo"que esta fijo y un RecyclerView con todos los tests personalizados. En caso de presionar el botón "Todo"lo que se hace es iterar sobre todos los ficheros y añadirlos a un ArrayList de elementos que pasaremos a la actividad de juego para que pueda generar las preguntas. En caso de seleccionar uno de los elementos del RecyclerView el adaptador se encargara de pasar la lista de elementos asociada a la actividad del juego.

6.2.1.12. `JuegoActivity`

La clase JuegoActivity es la encargada de gestionar toda lo que tiene que ver con el paciente realizando un test. Maneja desde la creación de las preguntas a el cambio de datos entre las preguntas y el recuento de los aciertos. Los elementos que se contienen en la vista son tres botones (A,B y C) que se usaran para las opciones en el test. Luego hay un campo de imagen para ir cargando los diferentes elementos y un elemento de texto que se ira actualizando y que sera el contador de aciertos.

Lo primero que se hace es recuperar la lista de elementos que se ha pasado y generar las preguntas para este test. Para poder generar las opciones para cada foto se crea una copia de la lista de elementos que permitirá eliminar elementos sin perder datos. Esto es necesario para que no salgan opciones repetidas entre los tres botones. A parte de esto se va iterando sobre la lista de elementos original, y poniéndolos en una de las tres posiciones de botones aleatoriamente, las otras dos opciones se rellenan con opciones aleatorias de la copia de elementos que hemos creado. Cada combinación de foto, opciones y nombre del elemento se va guardando en un Objeto de tipo pregunta que a su vez se mete en una lista de preguntas. Cuando se ha iterado sobre toda la lista de elementos, se mezcla el orden de las preguntas, se pone como pregunta actual la primera de la lista y se llama al método que actualiza las vistas.

```
private void generateQuestions(){

    for (Elemento e : elementList){
        String optionA = "";
        String optionB = "";
        String optionC = "";
        Random rand = new Random();
        int ansIndex = rand.nextInt(3);

        ArrayList<Elemento> elementListCopy = (ArrayList<Elemento>) elementList.stream()
                .map(elem -> new Elemento(elem))
                .collect(Collectors.toList());

        String[] options = new String[2];

        int numberOfElements = 2;

        for (int i = 0; i < numberOfElements; i++) {
            int randomIndex = rand.nextInt(elementListCopy.size());
            options[i] = elementListCopy.get(randomIndex).getNombre();
            if(options[i] == e.getNombre()) // Si uno de los elegidos aleaatorios es la respuesta repetir asignacion aleatoria
                i--;
            elementListCopy.remove(randomIndex);
        }


        if(ansIndex == 0) {
            optionA = e.getNombre();
            optionB = options[0];
            optionC = options[1];
        }
        else if(ansIndex == 1) {
            optionA = options[0];
            optionB = e.getNombre();
            optionC = options[1];
        }
        else if (ansIndex == 2){
            optionA = options[0];
            optionB = options[1];
            optionC = e.getNombre();
        }

        Pregunta p = new Pregunta(e.getFoto(), optionA , optionB, optionC , e.getNombre());
        questionList.add(p);
    }

    Collections.shuffle(questionList);
```

Figura 6.12: Metodo para generar preguntas en la clase JuegoActivity

La lógica sobre si una respuesta del paciente es correcta o no se encuentra en los listeners de cada botón. Lo primero que se comprueba es si la opción que se ha seleccionado es la respuesta correcta. Si lo es, se cambia el color de ese botón a verde y se incrementa el contador de correctos. Si aun quedan mas preguntas se abrirá un dialogo diciendo que se ha respondido correctamente y un botón para pasar a la siguiente pregunta. En la lógica de este dialogo es donde se resetean todos los colores y se actualizan todos los elementos de la vista para la siguiente pregunta, también se avanza el índice que representa cuantas preguntas llevamos. En caso de no quedar mas se finaliza el juego y se mueve a la actividad de resumen del test, aquí se pasaran datos sobre el numero de preguntas correctas e incorrectas para poder visualizarlos en la nueva actividad, tambien se pasa la lista de elementos para que en caso de querer repetir el mismo test, esta actividad pueda devolverla luego a la clase JuegoActivity. En el caso opuesto de que no se haya respondido

bien la pregunta se hace exactamente lo mismo excepto que el color de botón se mueve a rojo se muestra un dialog distinto y el contador de correctos no incrementa.

```
buttonA.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(preguntaActual.getOptionA().equals(preguntaActual.getAns())){
            buttonA.setBackgroundColor(ContextCompat.getColor(getApplicationContext(),R.color.lightGreen));
            correctosCounter++;
            if(pregInd < questionList.size() - 1){
                disableButton();
                correctDialog();
            }
            else{
                // no quedan mas preguntas - pantalla final
                finalJuego();
            }
        }
        else{
            // respuesta incorrecta - incorrect dialog
            buttonA.setBackgroundColor(ContextCompat.getColor(getApplicationContext(),R.color.red));
            if(pregInd < questionList.size() - 1){
                disableButton();
                incorrectDialog();
            }
            else{
                finalJuego();
            }
```

Figura 6.13: Listener de un botón en la clase JuegoActivity

6.2.1.13. `SinPreguntasActivity`

En esta vista se le muestra al paciente el numero de respuestas correctas e incorrectas y su porcentaje de acierto. Además como se comento antes se le da la opción de volver a jugar el mismo test, en este caso lo único que se tendría que hacer es volver a enviar la lista de elementos al juego ya que no se pasara por la actividad de selección de test desde donde normalmente se le pasan las listas de elementos al juego. También hay un botón para volver a la vista general de paciente.

6.2.2. Paquete Adapters

6.2.2.1. `ElementosAdapterCreador`

Como se ha explicado previamente los adapters se usan para manejar la interacción entre los elementos de la vista y el flujo de datos según la interacción del usuario con el RecyclerView. En el caso de este adapter tiene una funcionalidad extra que es la de crear la selección de elementos para la creación de los tests. Es por eso que hay dos parámetros extra, uno booleano para controlar si estamos en modo selección o no, y un ArrayList para ir guardando los elementos seleccionados. Esta clase extiende a RecyclerView.Adapter por lo que sobre escribe también los métodos de onCreate, onBind y getItemCount.

```
public class ElementosAdapterCreador extends RecyclerView.Adapter<ElementosAdapterCreador.ElementoViewHolder>{
    private Context mCtx;
    private List<Elemento> elementoList;
    private boolean haySeleccion = false;
    private ArrayList<Elemento> elementosSelec = new ArrayList<>();

    public ElementosAdapterCreador(Context mCtx, List<Elemento> elementoList) {
        this.mCtx = mCtx;
        this.elementoList = elementoList;
    }
```

Figura 6.14: Clase `ElementosAdapterCreador`

La clase `ElementoViewHolder` implementa la parte que directamente interactúa con la vista de la aplicación, aquí es donde se fijan los datos de cada elemento y donde se especifica la interacción de gestos con los distintos elementos. En el caso de este holder como podemos ver solo hay tres campos de texto, uno para el nombre, otro para la etiqueta y otro para la descripción.

```
class ElementoViewHolder extends RecyclerView.ViewHolder{

    TextView textviewNombre, textViewLabel, textViewDesc;

    public ElementoViewHolder(View itemView){
        super(itemView);

        textviewNombre = itemView.findViewById(R.id.textview_nombre);
        textViewLabel = itemView.findViewById(R.id.textview_label);
        textViewDesc = itemView.findViewById(R.id.textview_desc);
```

Figura 6.15: Clase `ElementoViewHolder`

Al crearse una vista para cada elemento del ReclyclerView se puede programar interacciones directamente con cada elemento único de la lista. En este caso hay dos gestos distintos que proporcionan distinta funcionalidad. Primero, la pulsación larga que lo que hace es arrancar el modo selección de elementos cambiando la variable bool a true. Lo primero que se observa aquí es si el elemento que se esta tocando esta ya en la lista de seleccionados, si es así, se le resetea el color y se le elimina de la lista. Si no esta aun en la lista se considera que se esta comenzando el modo selección, se le cambia el color de fondo a amarillo y se le añade a la lista de seleccionados. Finalmente se comprueba si se han de seleccionado todos los elementos, en este caso se vuelve al modo normal poniendo la variable bool a false.

La segunda opción es solo un toque, y aquí hay dos opciones importantes, si el bool del modo selección no esta activado nos vamos a la vista de modificación del elemento que hemos tocado. Si por el contrario si que estamos en modo selección tenemos una lógica muy parecido a antes,si se esta tocando un elemento ya seleccionado lo quitamos de nuestra selección, y si no lo esta se añade. De nuevo si el tamaño de la lista llega a 0, o lo que es lo mismo se han quitado todos los elementos de la selección se termina el modo selección.

```
itemView.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
        haySeleccion = true;
        if(elementosSelec.contains(elementoList.get(getAdapterPosition()))){
            itemView.setBackgroundColor(Color.TRANSPARENT);
            elementosSelec.remove(elementoList.get(getAdapterPosition()));
        } else {
            itemView.setBackgroundColor(Color.parseColor("#F3FF00"));
            elementosSelec.add(elementoList.get(getAdapterPosition()));
        }
        if(elementosSelec.size() == 0)
            haySeleccion = false;
        return true;
    }
});
itemView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(haySeleccion){
            if(elementosSelec.contains(elementoList.get(getAdapterPosition()))){
                itemView.setBackgroundColor(Color.TRANSPARENT);
                elementosSelec.remove(elementoList.get(getAdapterPosition()));
            } else {
                itemView.setBackgroundColor(Color.parseColor("#F3FF00"));
                elementosSelec.add(elementoList.get(getAdapterPosition()));
            }
            if(elementosSelec.size() == 0)
                haySeleccion = false;

        } else {
            Elemento element = elementoList.get(getAdapterPosition());
            Intent intent = new Intent(mCtx, UpdateElementoActivity.class);
            intent.putExtra("elemento", element);
            mCtx.startActivity(intent);
        }
```

Figura 6.16: Listeners en la clase `ElementoViewHolder`

6.2.2.2. `TestsAdapter`

Este adaptador sigue los mismos patrones que el anterior, la única diferencia es que en este caso el único elemento en la vista es un botón al cual le pondremos el nombre del test. Este RecyclerView se usa en el menú de selección de tests de los pacientes. Hay tambien un listener para cada botón, en caso de ser pulsado nos moveremos a la vista del juego y se le pasara a esa actividad también la lista de elementos asociados a cada test para que se puedan generar las preguntas acorde el test seleccionado por el paciente.

```
class TestsViewHolder extends RecyclerView.ViewHolder {

    Button testsBtn;

    public TestsViewHolder(View itemView) {
        super(itemView);

        testsBtn = itemView.findViewById(R.id.testsBtn);

        testsBtn.setOnClickListener(new View.OnClickListener(){

            @Override
            public void onClick(View v) {
                Test test = testList.get(getAdapterPosition()); //Pasamos los elementos contenidos en el test elegido
                Intent intent = new Intent(mCtx.getApplicationContext(), JuegoActivity.class);
                intent.putExtra("Test", test.getElementosSelec());
                mCtx.startActivity(intent);
            }
        });
```

Figura 6.17: Clase `TestsViewHolder`

6.2.2.3. `UsersAdapter`

Este adaptador se usa en la vista del administrador para ver a todos los usuarios registrados en la aplicación. Tiene tres elementos de texto que son el nombre completo del usuario, su email y el campo de validación. Desde aquí si un usuario es presionado se le llevara al admin a la vista de modificación de usuario desde donde podrá modificar algún dato o verificar al usuario.

```
class UserViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener{

    TextView textViewfullName, textuserEmail, textisValidated;

    public UserViewHolder(View itemView) {
        super(itemView);

        textViewfullName = itemView.findViewById(R.id.textview_fullName);
        textuserEmail = itemView.findViewById(R.id.textview_email);
        textisValidated = itemView.findViewById(R.id.textview_isValidated);

        itemView.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        User user = userList.get(getAdapterPosition());
        Intent intent = new Intent(mCtx, UpdateUserActivity.class);
        intent.putExtra("user", user);
        mCtx.startActivity(intent);
    }
```

Figura 6.18: Clase `UserViewHolder`

6.2.3. Paquete DB-objects

En este paquete se definen clases para representar los diferentes tipos de ficheros que luego se suben a la base de datos de Firestore. Antes de subir nada a Firestore se recopila todas las entradas de datos que ha dado el creador o el admin y desde ahí se crea un nuevo objeto con este esqueleto y se sube a la base de datos. Es por eso que son clases simples con getter y setters ya que toda la información se manipula entorno a estos objetos.

La única excepción es la clase `Pregunta` que aunque no se sube a la base de datos sigue la misma estructura de esqueleto de fichero. En este caso se usa en la parte del juego para generar y guardar los datos sobre el elemento que estamos preguntando y las posibles opciones que generamos aleatoriamente. Todos los datos que están en las preguntas vienen de los elementos seleccionados en algún test.

6.3. Acceso al repositorio GitHub

En el siguiente enlace se puede acceder al repositorio publico del trabajo. Están todos los archivos de la aplicación excepto un JSON con las claves de las APIs que se usan para la conexión a la cuenta de Google Cloud Platform.

https://github.com/jaimeantolin/TFG

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

La aplicación implementada cumple con todos los requisitos funcionales y no funcionales propuestos en el capitulo 3. Por lo que se podría decir que el desarrollo del trabajo ha sido un éxito. Los tres actores principales pueden desarrollar sus tareas sin ningún tipo de problema. El administrador es capaz de validar a los usuarios y modificar sus datos. El creador tiene las herramientas necesarias para visualizar, crear y modificar los elementos de la base de conocimiento y además crear los tests personalizados para el paciente. El paciente puede explotar la base de conocimiento de diversas formas. Con la función de búsqueda que le permite recuperar información de dos maneras. A través de una foto o del nombre del objeto. También pueden entrenar su memoria practicando con los tests personalizados o con un test de todos los elementos que hay en su base de conocimiento.

En cuanto a la usabilidad de la aplicación, ha quedado una aplicación muy limpia y simple. Lo único que habría que explicar son ciertos gestos como el del toque largo para iniciar la selección de elementos, pero aun así todos los gestos usados son muy intuitivos y comunes en muchas aplicaciones. También se han incluido detalles como el que te guarde la sesión para que el paciente no tenga que estar autenticándose todo el rato y nos arriesguemos a una situación en la que el paciente quiere usar la aplicación estando solo y no se acuerda de alguna de sus credenciales de acceso.

Dicho todo esto la aplicación tiene mucho donde mejorar. La aplicación desarrollada es mas una prueba de concepto de que tenemos las tecnologías para poder desarrollar esta idea y ofrecer esta funcionalidad a los pacientes que un producto terminado, para eso habría que limar muchos detalles aun. Por ejemplo un potencial problema es que se esta basando muchos de los servicios que ofrecemos en la plataforma cloud de Google esto para hacer pruebas de funcionalidad y demostrar que la aplicación funciona en un sand box esta genial, pero si se fuera a desplegar se tendrían que plantear los potenciales costes de utilizar todos estos servicios y evaluar si hay algún otro vendor que pueda ofrecer las mismas funcionalidades a menos precio. Otro problema importante es que en la actual versión no se crean bases de conocimiento personalizadas para cada paciente, sino que solo hay una para probar toda la funcionalidad. En cuanto se quisiera usar en un entorno real habría que añadir esta funcionalidad.

7.2. Trabajo futuro

- Modelo personalizado de reconocimiento de imágenes
  - Para la generación de las etiquetas de las fotos se usa la API de Google Cloud Vision con uno de sus modelos pre-entrenados. Como se explico anteriormente se hizo una mejora en la cuenta de Google cloud para poder optar a un modelo que reconocía 9000 tipos de etiquetas en vez de algo mas de 400. Aun así la granularidad de las etiquetas que son devueltas no son siempre lo mejor, ya que los grupos son muy amplios como por ejemplo electrodoméstico o "peripheral". Para que la funcionalidad de búsqueda por imagen funcionara perfectamente, lo óptimo seria entrenar un modelo propio de imágenes de objetos que se puedan encontrar por casa, haciendo esto se obtendrían etiquetas mucho mas fiables y por ende mejores búsquedas.
- Reconocimiento de personas
  - Ya que se esta proponiendo entrenar modelos para los objetos se podría entrenar otro modelo para reconocer a la gente en el circulo mas cercano del paciente y que al aportar una foto de esa persona la aplicación fuera capaz de decirle quien es y su relación con el paciente.
- Conectar pacientes y creadores
  - Esto es el punto mas importante si se quisiera avanzar con la aplicación y llevarla a producción. La idea seria que cada creador tuviera uno o mas pacientes a su cargo cada uno con su base de conocimiento personalizada, y que cuando el creador entrara en su cuenta pudiera seleccionar para quien quiere modificar la base de conocimiento o crear tests personalizados. Luego el paciente cuando accediera a la aplicación solo vería el contenido creado para el y no el de otros pacientes.
- Comunicación entre creador y paciente
  - Una posible funcionalidad que mejoraría aun mas la funcionalidad de los tests personalizados seria si se pudiera crear algún tipo de comunicación entre el creador y el paciente. Esto se podría hacer de distintas formas, con mensajes directos, o incluso asignando diferentes tests a diferentes días y obligar a los pacientes a realizar los tests que el creador le ha asignado ese día. Esto permitiría un tratamiento aun mas personalizado y aseguraría que los pacientes están practicando los contenidos que el creador quiere.
- Desarrollo IOS
  - Seria también interesante una vez la aplicación estuviera pulida y se tuviera todas las funcionalidades necesarias para llevarlo a usuarios de verdad poder ofrecer la aplicación en IOS. Esto permitiría llegar a mas gente ya que Apple tiene una gran cuota de mercado, y sobre todo no obligar a nadie a cambiar de dispositivo exclusivamente para poder usar la aplicación.

Chapter 8

Conclusions and future work

8.1. Conclusions

The application that has been implemented meets all the functional and non functional requierements that were proposed in chapter 3. So it could be said that the development of the project has been a success. The 3 main actors are able to carry out there tasks without any problem. The creator has the necessary tools to visualize, create and modify objects in the knowledge base as well as creating personalized tests for the patient. The patient can exploit the knowledge base in diverse ways. Firstly with the search functions that allows the patient to retrieve information in various ways, with an image and with the name of the object. The patient can also train his memory by practicing the personalized tests or with a test of everything in the knowledge base.

When it comes to the usability of the application, the app is very clean and simple. The only thing that may require some explaining is how to use the different gestures like the long press to start an object selection in the creator view. Having said that, all the gestures used are very intuitive and common in other applications so their is zero to no learning curve. Other usability features have been considered like the fact that the app saves your session and does not require to authenticate yourself every time you use the app. This is to try and avoid a situation where a patient want to use the application when they are alone and they cannot access it due to not being able to remember some part of his credentials.

Having said all of this, the application has a lot of room for improvement. The developed app is more a proof of concept that we have the technologies to develop this idea and offer these functionalities to patients than a finished product, in order to be something deployable there would have to be details to be figured out. For example a potential problem is that a lot of the services that are being offered are based on Google Cloud Platform. This is great to be able to test things out and make sure the app works in a sand box environment, but if it was to be deployed the costs of using these services would have to be evaluated and analyze if there is another cloud vendor that can offer the same capabilities for a cheaper price. Another problem is that in the current version there are not individual knowledge bases for each patient, there is just one to test all the functionalities. This would be the first thing to do if there was any intent of deploying the application to real users.

8.2. Future Work

- Custom image labelling model
  - In order to generate the tags that are associated to objects, the Google Cloud Vision API is used with one of their pre-trained models.It was explained earlier that an upgrade to the Google cloud account was done to be able to use a model with 9000 types of tags instead of just over 400. Even with this upgrade the granularity of the tags was not always the best, it tended to be too broad with tags like "electronics" or "peripheral". In order to make the image searching functionality as good as possible we should implement a custom model to recognize objects that can be found around a household. If we were able to achieve this the tags would be much more reliable and would improve our searches.
- People recognition
  - Since we are proposing to train models for the objects, we could train a second model to recognize people who have a relationship with the patient. The idea would be that the patient can give the app a picture of that person and the app would be able to tell them their name and their connection to the patient.
- Connecting patients and creators
  - This is the most important point if we wanted to advance with the application and take it to production. The idea would be that each creator could have one or more patients under their control and that each one would have their personalized knowledge base. When the creator would sign in he would first select for what patient he wants to modify the knowledge base or create tests. Then the patient on sign in would just see the content that has been created for them and not for others.
- Communication between patient and creator
  - Another possile functionality that would further improve the personalized tests feature would be if we were able to establish some kind of communication between them. This could be done in several ways, with direct messages, or even being able to assign different tests for different days. This would further personalize the experience for the patient and would guarantee that they are doing the work their care takers set for them.
- IOS Development
  - It would also be interesting to once the application was stable in Android and ready to be deployed that we could also offer it for IOS users. This would allow us to reach more people since Apple has a large market share, and would allow people to not have to change their device to use the app.

Capítulo 9

Manual de instalación

A continuación se explicara como poder instalar la aplicación en cualquier terminal. Las únicas dependencias serán, tener una cuenta de GitHub y tener instalado en un ordenador git y AndroidStudio.

1. Lo primero que se hará es ir al enlace del repositorio y presionar en el botón 'Fork' esto lo que hará es crear una copia de este repositorio la cuenta, que se podrá usar y modificar como se quiera.

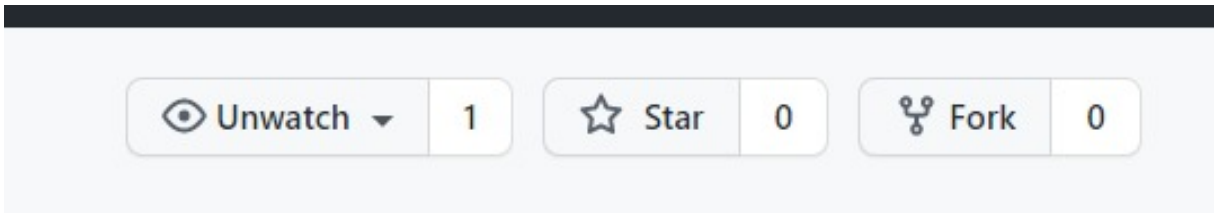


Figura 9.1: Botón fork en el repositorio de GitHub

2. Ahora lo se debe abrir una terminal en el sistema. Después ir al directorio donde se quiera descargar el repositorio usando el comando 'cd'. Una vez ahí usar el comando 'git clone https://github.com/jaimeantolin/TFG'. Una vez echo ya se tendrá el repositorio en el sistema.

Figura 9.2: Comando de terminal para clonar el repositorio

3. Lo siguiente seria importar el proyecto en AndroidStudio. Una vez abierto el programa se importara un nuevo proyecto en blanco y con lenguaje Java. Ahora se

navegara hasta la carpeta donde se ha guardado el repositorio y se seleccionara como ruta del proyecto. Una vez echo esto AndroidStudio se ocupara de importar todo el proyecto.

4. Como ultimo paso, una vez importado correctamente el proyecto solo queda ejecutarlo. Aquí se elegirá el que tipo de emulador con el que se quiere probar la aplicación o en el caso de conectar un terminal Android si se quiere ejecutar ahí. Después ya solo quedaría presionar el botón de ejecutar aplicación (símbolo del play), que construirá la aplicación e instalara todas las dependencias antes de ejecutar.

Figura 9.3: Opciones de ejecución de la aplicación

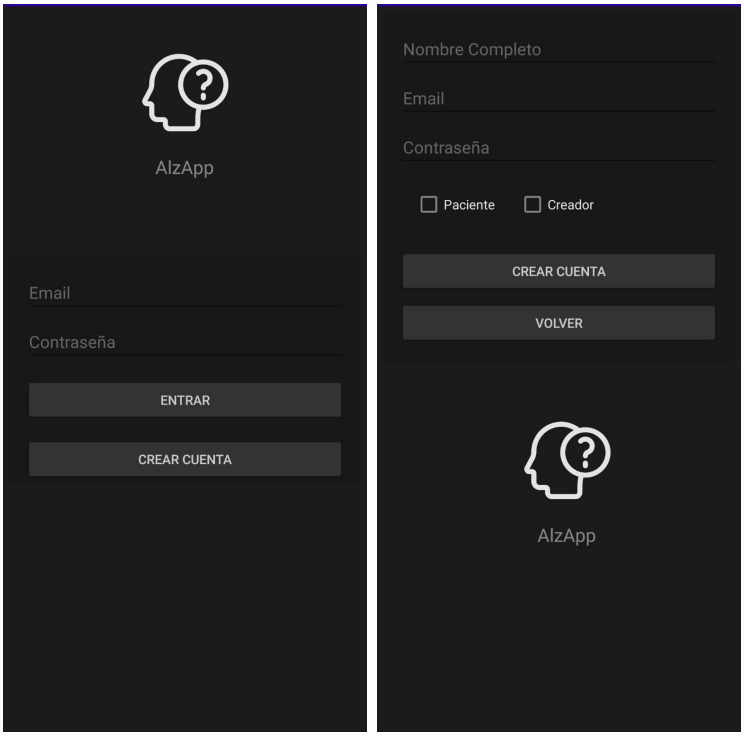
Capítulo 10

Manual de usuario

10.1. Todos los usuarios

Todos los usuarios cuando acceden por primera vez a la aplicación pasaran por estas dos pantallas. La primera es para poder iniciar sesión usando el email y la contraseña y la segunda es para registrarse en caso de que sea la primera vez que un usuario usa la aplicación. Para llegar al registro hay que presionar el botón de crear cuenta en la vista de inicio de sesión. La única forma de evitar estas vistas es si ya tienes una sesión activa en el dispositiva, en ese caso iras directamente a la vista de tu tipo de usuario.

En el caso de la pantalla de registro todos los campos deberán ser rellenados y un tipo de usuario seleccionado.

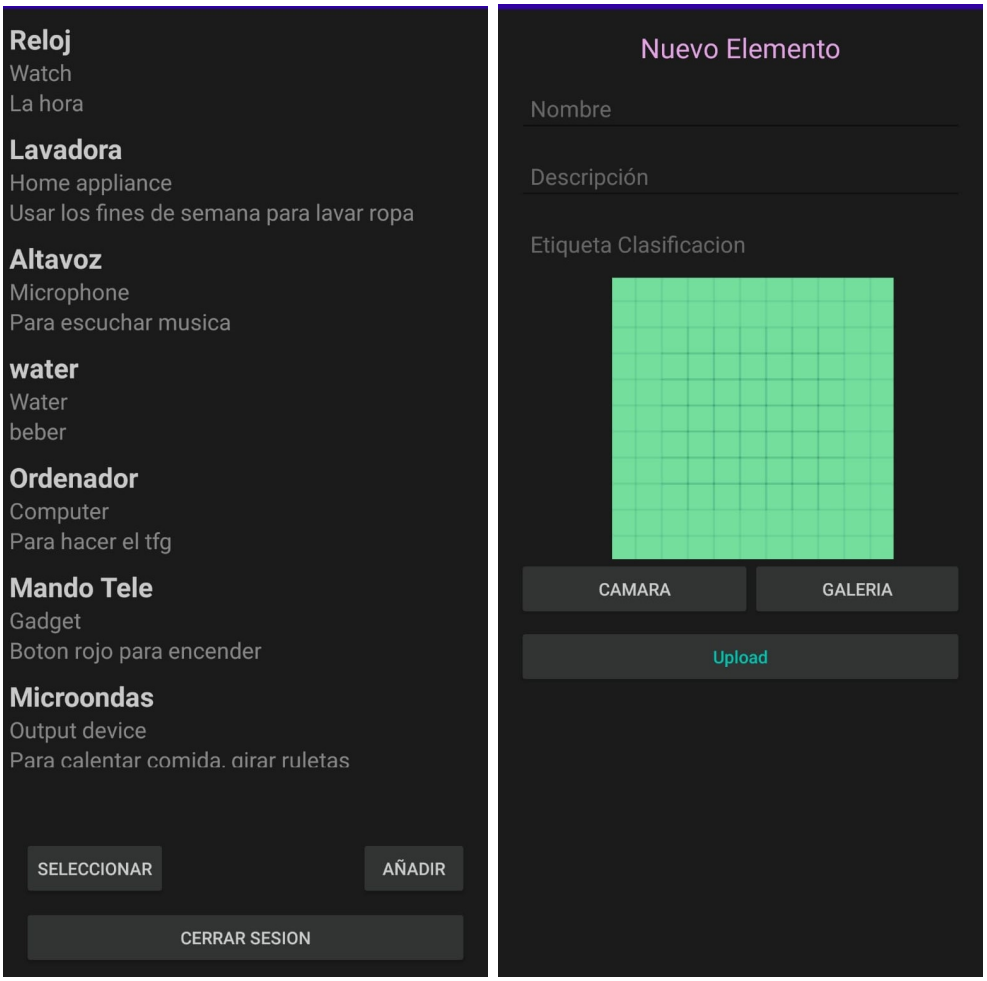


(a) (b)

Figura 10.1: Vistas de inicio de sesión y registro

10.2. Creador

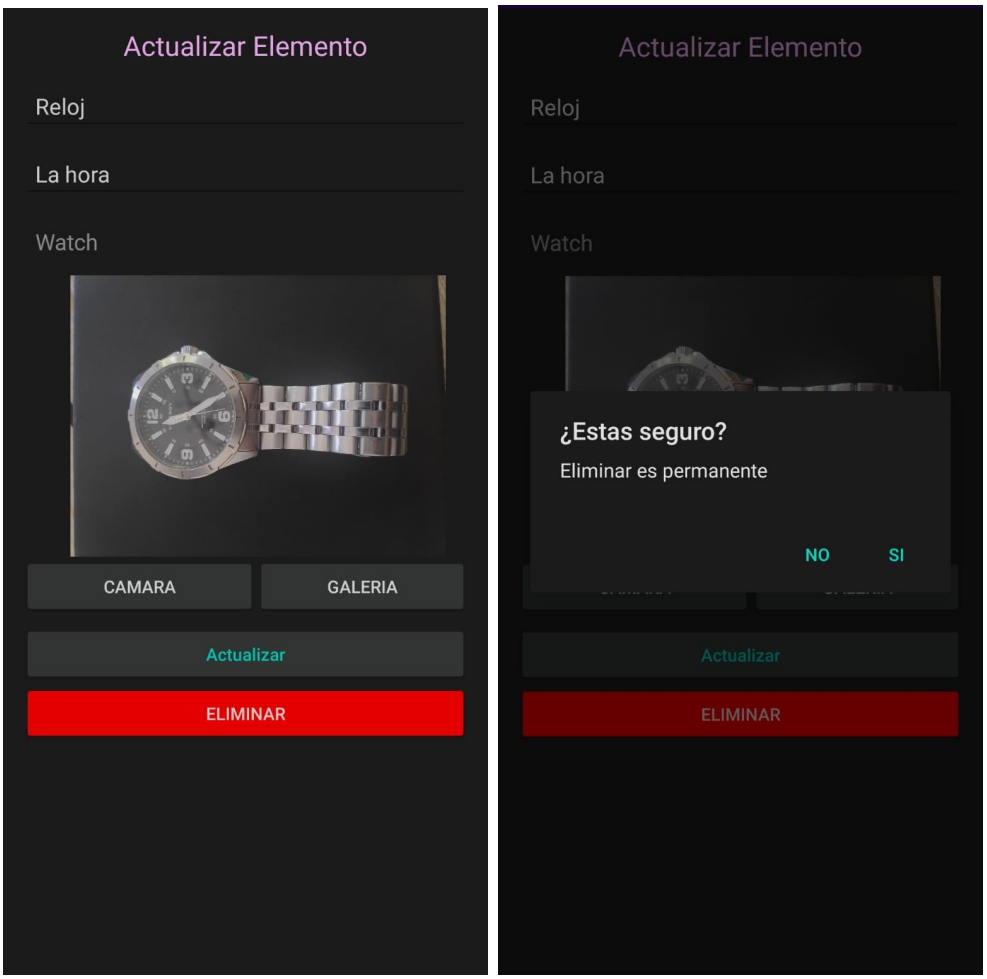
Una vez se ha autenticado el creador accederá a la pantalla general. En esta pantalla podrá ver el listado de elementos que hay en la base de conocimiento y realizar varias tareas. La primera sera pulsando el botón añadir que le llevara a la vista de crear un nuevo elemento, en el que tendrá que rellenar los campos de nombre y descripción y aportar una foto mediante la cámara o la galería antes de presionar el botón de subir el elemento.



(a) (b)

Figura 10.2: Vista de creador y nuevo elemento

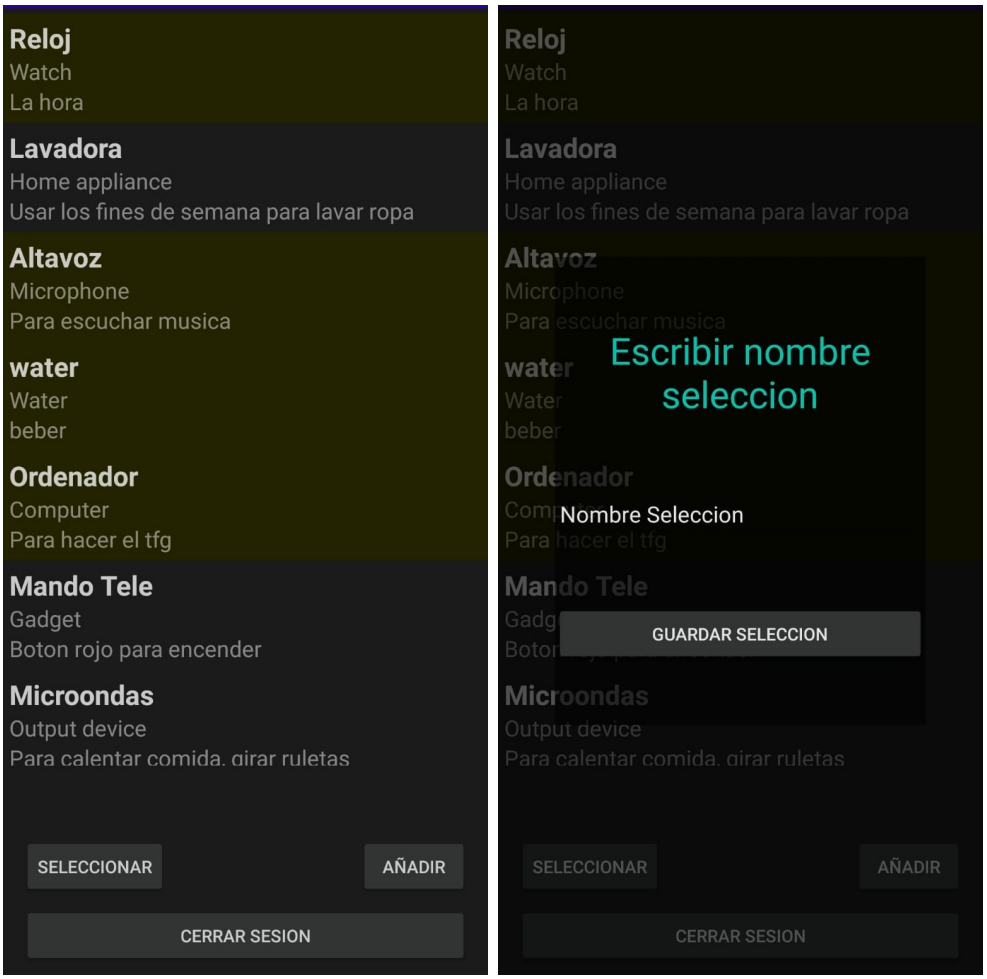
También podrá actualizar los elementos que ya existen presionando sobre ellos en la lista, aquí podrá cambiar el nombre o la descripción e incluso cambiar la foto que generara una nueva etiqueta. Para guardar los cambios debe presionar el botón de actualizar. En caso de que quiera eliminar el elemento por completo podrá presionar sobre el botón de eliminar, ahora saldrá un dialogo de alerta para confirmar la eliminación ya que es permanente.



(a) (b)

Figura 10.3: Vistas de actualizar elemento y alerta de eliminación

La ultima funcionalidad que tiene la vista de creador es la generación de tests personalizados. Con un toque largo se empieza este modo, y tocando otros elementos se añaden a la selección. Tocando elementos ya seleccionados se eliminan de la selección. Una vez se quiere guardar la selección que se ha echo se presiona el botón seleccionar que abrirá un dialog para meter el nombre, y presionando el botón guardar selección se guarda en la base de datos.

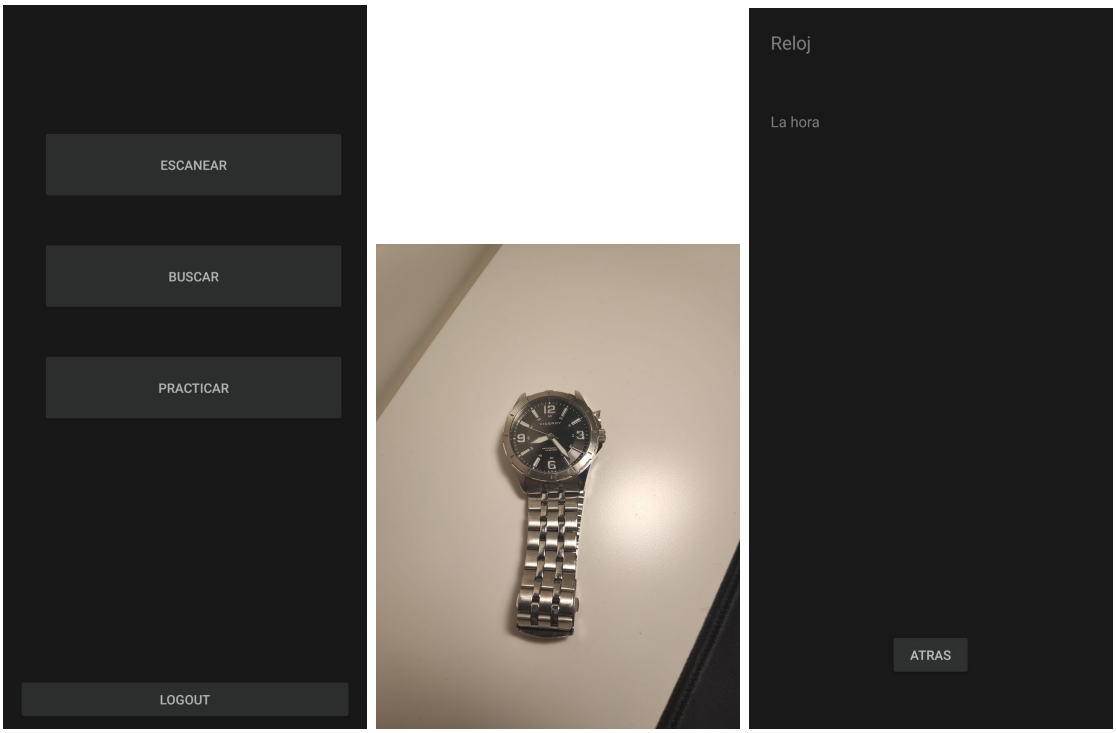


(a) (b)

Figura 10.4: Modo selección en la vista de creador, y dialog para guardar

10.3. Paciente

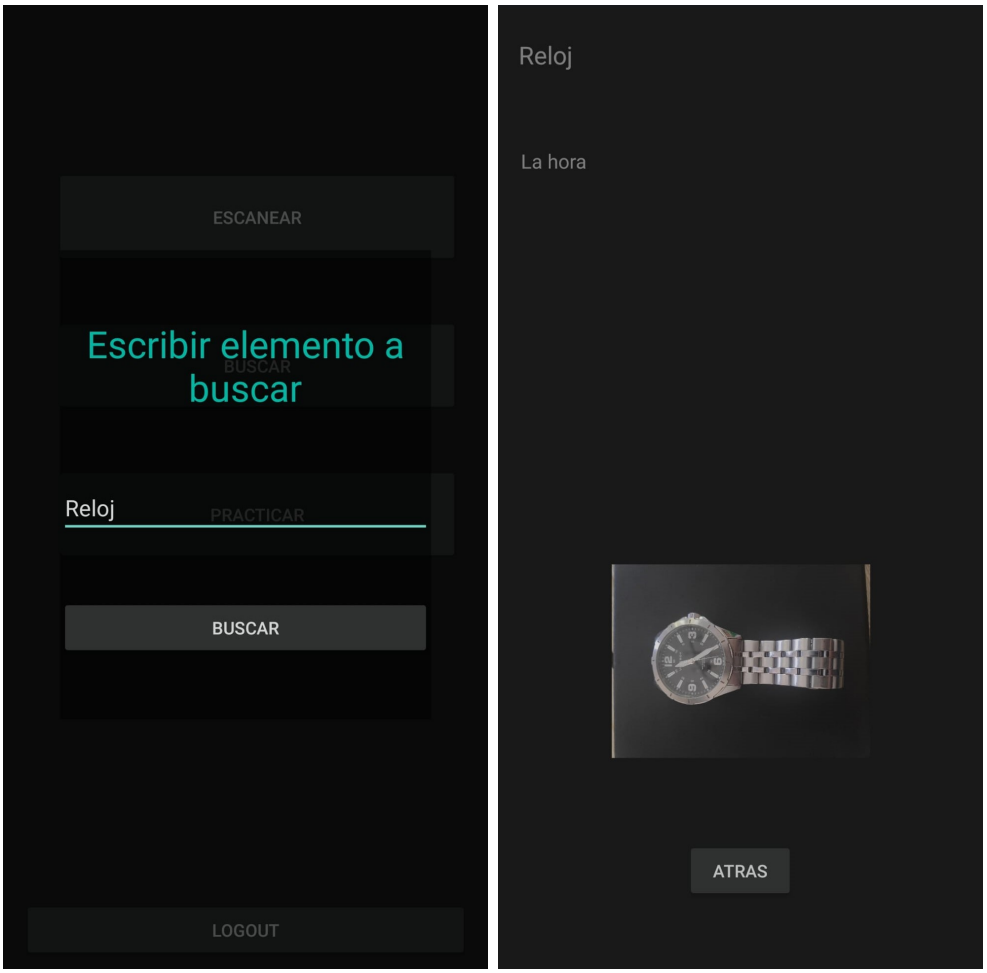
El paciente tiene tres funcionalidades principales, buscar usando una imagen o por el nombre, y practicar con tests sobre la base de conocimiento. En esta primera secuencia de imágenes se puede ver el proceso de la selección por imagen, una vez se presiona el botón se abre la cámara y se toma una foto. En este caso de un reloj, la aplicación ahora etiqueta la foto y hace una búsqueda en la base de conocimiento usando esa etiqueta y devuelve su nombre y su descripción como se puede ver en el ultimo panel. Ahí también hay un botón para volver a la vista principal del paciente.



(a) (b) (c)

Figura 10.5: Vista general del paciente, foto tomada para reconocimiento y el resultado de la búsqueda usando la etiqueta

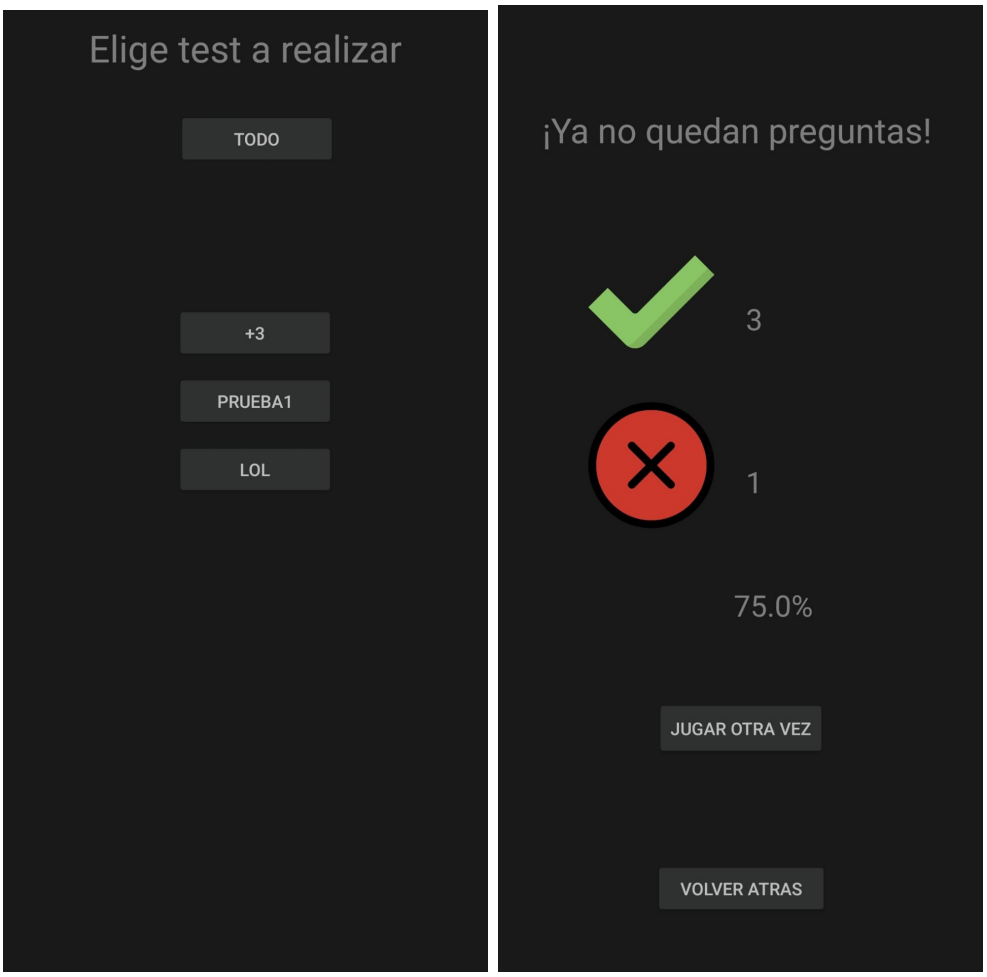
A continuación se puede ver el dialog que se abre cuando se presiona lo opción de búsqueda por nombre. Hay un campo de texto que se debe rellenar y presionar el botón de buscar. Después se puede ver la vista con toda la información recopilada para que el paciente pueda acordarse del objeto.



(a) (b)

Figura 10.6: Dialog para rellenar el nombre de la búsqueda y pantalla de resultado

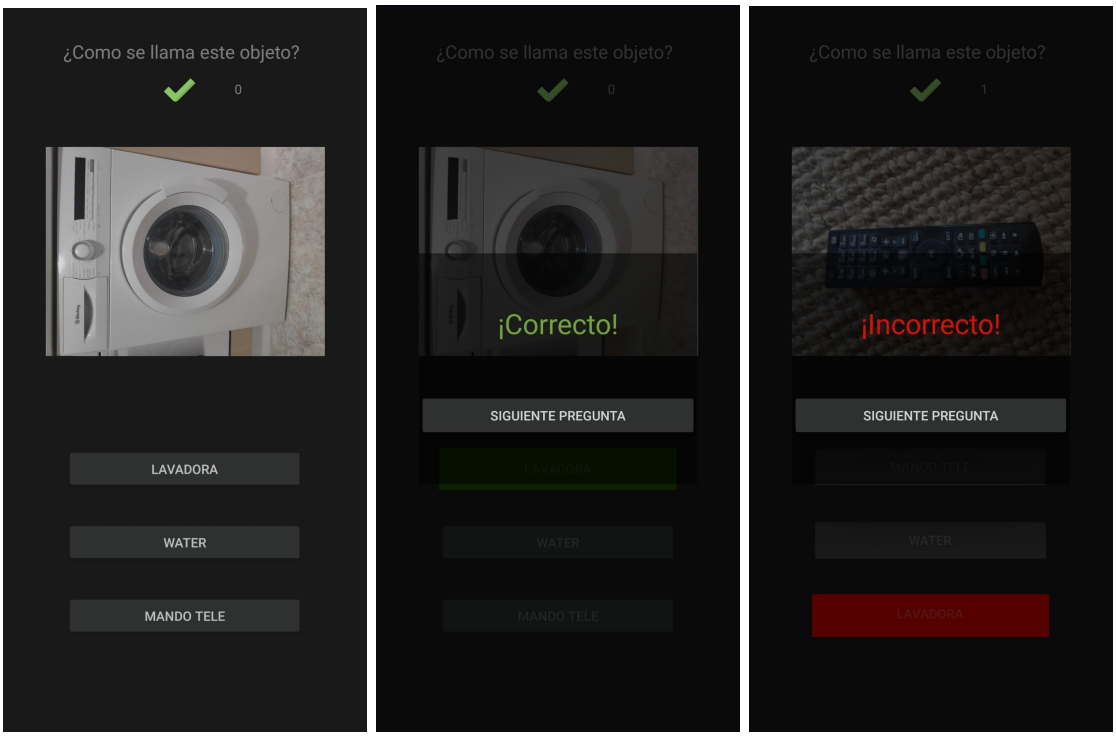
Si el paciente decide practicar recordando la base de conocimiento con algunos tests sera direccionado a la primera vista. Como se puede ver hay muchos botones, el primero de 'todo' es fijo y te genera un test aleatorio con todos los elementos de la base de conocimiento. Los demás representan cada uno, un tests personalizado diseñado por el creador. La segunda vista es la del final del test que te muestra tus aciertos y fallos y tu resultado en un porcentaje. Tambien te permite volver a repetir el test o volver a la vista general del paciente.



(a) (b)

Figura 10.7: Vistas de selección de tests y de resultados del test

Aquí se puede ver la estructura de cualquier pregunta, en cada una cambia la foto y las tres opciones. Después de que un paciente elija una de las opciones se muestra un dialog contando el resultado y cambia el color del botón presionado. Hasta que no se aprieta el botón de siguiente pregunta del dialog no se avanza ni se actualiza la pregunta.

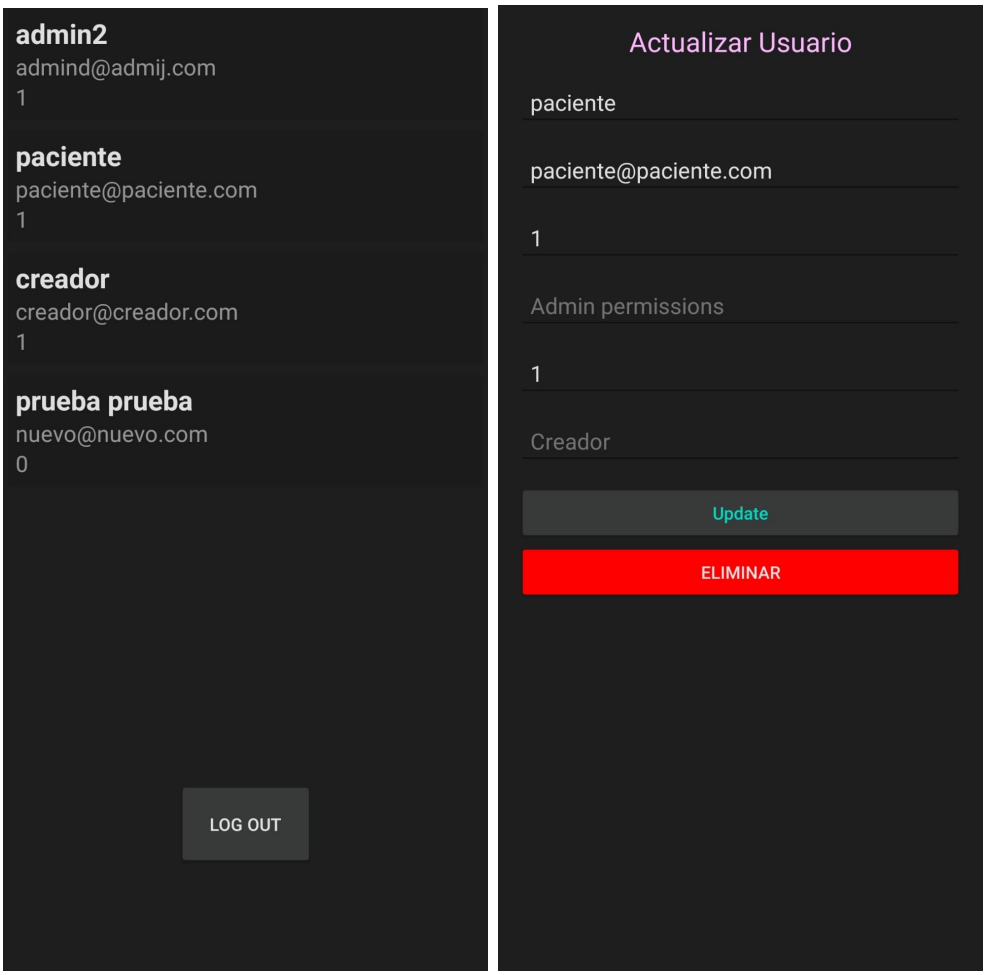


(a) (b) (c)

Figura 10.8: Vista de una pregunta de un test, dialogs de correcto e incorrecto

10.4. Admin

En la vista predeterminada del administrador solo puede ver el listado de usuarios registrados, su email, su nombre y si esta validado o no. Si presiona sobre ellos puede modificar sus credenciales, y sobre todo validarlos. También puede eliminar un usuario de Firestore impidiendo que jamas sea validado.



(a) (b)

Figura 10.9: Vistas del admin y actualización de usuarios

Bibliografía

[1] Google. Android studio. https://developer.android.com/studio. [Online; Accessed: 2021-02-01].

[2] Oracle. Java. https://www.java.com/en/. [Online; Accessed: 2021-02-01].

[3] Google. Firebase authentication. https://firebase.google.com/docs/auth. [Online; Accessed: 2021-08-06].

[4] Google. Firebase firestore. https://firebase.google.com/docs/firestore. [Online; Accessed: 2021-08-06].

[5] Google. Firebase storage. https://firebase.google.com/docs/storage. [Online; Accessed: 2021-08-06].

[6] Google. Firebase functions. https://firebase.google.com/docs/functions. [Online; Accessed: 2021-08-06].

[7] Google. Google cloud vision. https://cloud.google.com/vision. [Online; Accessed: 2021-08-06].

[8] Google. Create dynamic lists with recyclerview. https://developer.android.com/guide/topics/ui/layout/recyclerview?gclid=CjwKCAjwvuGJBhB1EiwACU1AiZBUlSt83-nLmxqacslzoaLIi2eMF6cid57FkA7QQb9JPCh3JbDiQhoCd0BwE&gclsrc=aw.ds. [Online; Accessed: 2021-08-14].

[9] Google. Toast overview. https://developer.android.com/guide/topics/ui/notifiers/toasts. [Online; Accessed: 2021-08-14].

[10] Google. Dialogs. https://developer.android.com/guide/topics/ui/dialogs. [Online; Accessed: 2021-08-15].

[11] Open Source. Picasso. https://square.github.io/picasso/. [Online; Accessed: 2021-08-14].

[12] Google. Build a responsive ui with constraintlayout. https://developer.android.com/training/constraint-layout. [Online; Accessed: 2021-08-15].

[13] Alzheimer Association. Etapas del alzheimer. https://stockfishchess.org. [Online; Accessed: 2021-08-01].

[14] Amèrica Morera Sara Domènech Ana Llorente Lluís Tárraga, Mercè Boada. *Volver a empezar*. Glosa Ediciones, 2018.

[15] Amazon. Amazon web services. aws.amazon.com. [Online; Accessed: 2021-02-01].

[16] Microsoft. Azure. https://azure.microsoft.com/. [Online; Accessed: 2021-02-01].

[17] Google. Google cloud platform. https://cloud.google.com/. [Online; Accessed: 2021-02-01].

[18] Spacegroup. Iridis. https://play.google.com/store/apps/details?id=com.SpaceAppliedTechnology.Iridis&hl=en&gl=US. [Online; Accessed: 2021-08-01].

[19] Emma Yang. Timeless. https://www.timeless.care/. [Online; Accessed: 2021-08-01].

[20] MindMate Inc. Mindmate. https://www.mindmate-app.com/about-us. [Online; Accessed: 2021-08-01].

[21] Tactus Therapy Solutions LTD. Spaced retrieval therapy. https://apps.apple.com/us/app/spaced-retrieval-therapy-memory/id498787795. [Online; Accessed: 2021-08-01].

[22] Google. Android developer fundamentals. https://developer.android.com/courses/fundamentals-training/overview-v2. [Online; Accessed: 2021-03-15].

[23] Google. Build your first app. https://developer.android.com/training/basics/firstapp. [Online; Accessed: 2021-03-15].

[24] SmallAcademy. Android firebase admin/user access level management. https://www.youtube.com/watch?v=_I7PHFrAd-g&ab_channel=SmallAcademy. [Online; Accessed: 2021-03-17].

[25] bikashthapa01. firebase-admin-user-roles. https://github.com/bikashthapa01/firebase-admin-user-roles. [Online; Accessed: 2021-03-18].

[26] Belal Khan. android-cloud-firestore-example. https://github.com/probelalkhan/android-cloud-firestore-example. [Online; Accessed: 2021-03-26].

[27] Simplified Coding. 2 firestore android tutorial - reading data. https://www.youtube.com/watch?v=NWEfGZeDuAY&list=PLk7v1Z2rk4hhN_d5e1ACuhVNrTDOew66L&index=7&ab_channel=SimplifiedCoding. [Online; Accessed: 2021-03-27].

[28] Google. Label images securely with cloud vision using firebase auth and functions on android. https://firebase.google.com/docs/ml/android/label-images. [Online; Accessed: 2021-05-10].

[29] Firebase. functions-samples. https://github.com/firebase/functions-samples/blob/main/vision-annotate-images/functions/src/index.ts. [Online; Accessed: 2021-05-11].

[30] Google Samples. mlkit. https://github.com/googlesamples/mlkit/tree/master/android/material-showcase. [Online; Accessed: 2021-05-17].

[31] Google. Obtén datos con cloud firestore. https://firebase.google.com/docs/firestore/query-data/get-data?hl=es#java. [Online; Accessed: 2021-06-05].

[32] Adam McQuistan. How to clone java collections with streams. https://thecodinginterface.com/blog/java-collections-stream-cloning/. [Online; Accessed: 2021-07-11].

[33] Sarvesh Chavan. Quiz game. https://github.com/sarveshchavan7/Quiz-Game. [Online; Accessed: 2021-07-11].

[34] The Code City. Select multiple rows in a recyclerview. https://www.youtube.com/watch?v=E01Nmp_Db3U&ab_channel=TheCodeCity. [Online; Accessed: 2021-08-15].

[35] Ashraff Hathibelagal. How to add multiple selection to android recyclerview. https://code.tutsplus.com/tutorials/how-to-add-selection-support-to-a-recyclerview--cms-32175. [Online; Accessed: 2021-08-15].