UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño de una aplicación que ayude en la detección precoz de la demencia

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Fernández Plaza, Jaime

Tutor/a: Vidal Oriola, Germán Francisco

CURSO ACADÉMICO: 2022/2023

Resum

L'augment de l'esperança de vida ha portat a un increment en la prevalença del nombre de persones afectades per malalties neurodegeneratives. La naturalesa crònica d'aquestes malalties les posiciona com un dels grans desafiaments en el món de la medicina, la psicologia i la bioquímica.

MindTest sorgeix com una proposta per a ajudar a persones afectades o en risc de patir aquestes malalties. Aquesta proposta pretén facilitar la detecció precoç de possibles símptomes mitjançant la realització de tests i la possibilitat de consultar estadístiques de les puntuacions. La realització d'aquests test també pot funcionar com a exercici cognitiu per a afectats que comencen a experimentar els efectes adversos d'aquestes malalties.

A causa de la gran popularitat i integració dels telèfons intel·ligents en la societat actual, *MindTest* es presenta com una aplicació mòbil multiplataforma desenvolupada amb el popular *framework React Native* i disponible per a dispositius *iOS* i *Android*.

Addicionalment la nostra proposta també fa ús d'altres serveis com *Cloud Firestore*, *Firebase Authentication*, *Cloud Storage* i *EmailJS* .

El present treball recull tots els processos duts a terme per al desenvolupament de la nostra aplicació. Des de la investigació i documentació, l'anàlisi dels requisits, les etapes de disseny i la posterior implementació i testatge de l'aplicació.

Paraules clau: malalties neurodegeneratives, telèfons intel·ligents, aplicació, React Native, framework, multiplataforma, desenvolupament

Resumen

El aumento de la esperanza de vida ha llevado a un incremento en la prevalencia del número de personas afectadas por enfermedades neurodegenerativas. La naturaleza crónica de estas enfermedades las posiciona como uno de los grandes desafíos en el mundo de la medicina, la psicología y la bioquímica.

MindTest surge como una propuesta para ayudar a personas afectadas o en riesgo de padecer estas enfermedades. Esta propuesta pretende facilitar la detección precoz de posibles síntomas mediante la realización de tests y la posibilidad de consultar estadísticas de las puntuaciones. La realización de estos test también puede funcionar como ejercicio cognitivo para afectados que comiencen a experimentar los efectos adversos de estas enfermedades.

Debido a la gran popularidad e integración de los teléfonos inteligentes en la sociedad actual, *MindTest* se presenta como una aplicación móvil multiplataforma desarrollada con el popular *framework React Native* y disponible para dispositivos *iOS* y *Android*.

Además, nuestra propuesta también hace uso de otros servicios como *Cloud Firestore*, *Firebase Authentication*, *Cloud Storage* y *EmailJS* .

El presente trabajo recoge todos los procesos llevados a cabo para el desarrollo de nuestra aplicación. Desde la investigación y documentación, el análisis de los requisitos, las etapas de diseño y la posterior implementación y testeo de la aplicación.

Palabras clave: enfermedades neurodegenerativas, teléfonos inteligentes, aplicación, React Native, framework, multiplataforma, desarrollo

Abstract

Increased life expectancy has led to an increase in the prevalence of the number of people affected by neurodegenerative diseases. The chronic nature of these diseases positions them as one of the great challenges in the world of medicine, psychology and biochemistry.

MindTest arises as a proposal to help people affected or at risk of suffering from these diseases. This proposal aims to facilitate the early detection of possible symptoms by performing tests and the possibility of consulting statistics of the scores. The performance of these tests can also function as a cognitive exercise for sufferers who are beginning to experience the adverse effects of these diseases.

Due to the great popularity and integration of smartphones in today's society, MindTest is presented as a cross-platform mobile application developed with the popular React Native framework and available for iOS and Android devices.

Additionally, our proposal also makes use of other services such as Cloud Firestore, Firebase Authentication, Cloud Storage and EmailJS.

This work collects all the processes carried out for the development of our application. From the research and documentation, the requirements analysis, the design stages and the subsequent implementation and testing of the application.

Key words: neurodegenerative diseases, smartphones, app, React Native, framework, cross-platform, development

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación

Las enfermedades neurodegenerativas (EN) constituyen un grupo específico de afecciones que afectan directamente al desempeño del sistema nervioso central. Dichas enfermedades se deben a un aumento de la muerte de neuronas en diversas áreas del cerebro. Esta degeneración neuronal puede ocasionar gran diversidad de síntomas, desde la pérdida de memoria, falta de coordinación, ansiedad, cambios en el estado de ánimo, hasta incluso llegar a afectar a muchas actividades del organismo como pueden ser el equilibrio, la movilidad, el habla, la respiración e incluso el funcionamiento cardíaco.

El aumento general de la esperanza de vida ha supuesto un incremento en la prevalencia de este tipo de afecciones y de las tasas de muerte por este tipo de enfermedades, llegando a suponer un gran desafío a nivel social, humano e incluso económico.

A pesar de no existir tratamientos para este tipo de enfermedades, la detección precoz puede ser un factor esencial para ralentizar sus efectos. El diagnóstico temprano de una enfermedad neurodegenerativa puede suponer una gran mejoría en el tratamiento del paciente, así como en la transformación de sus hábitos de vida. Además, la estimulación cognitiva mediante ejercicios o problemas puede ayudar a ralentizar la aparición de estos síntomas.

A nivel personal, he sido testigo de los perjuicios que las enfermedades neurodegenerativas son capaces de producir en los afectados y en su círculo cercano.

Es por todo esto que nace la motivación para trabajar en *MindTest*, una aplicación que, mediante el uso de tests, permita estimular y facilitar el seguimiento de las capacidades de personas de la tercera edad para, así, facilitar la detección de estas enfermedades, contribuyendo de este modo al bienestar de las personas que las padecen y los posibles afectados de su entorno.

1.2 Objetivos

Este trabajo tiene como principal objetivo el desarrollo de una aplicación móvil que permita identificar algunos de los síntomas típicos de las enfermedades neurodegenera-

tivas a partir de la realización de test que evalúen distintas capacidades intelectuales del usuario.

En ningún caso el objetivo de esta aplicación es emitir un diagnóstico con validez médica. El objetivo principal es permitir al usuario monitorizar la evolución de diversas aptitudes que tienden a deteriorarse con la aparición de este tipo de enfermedades y así poder detectar de forma temprana déficits en alguna de estas capacidades.

Además de lo ya mencionado, la aplicación deberá brindar información útil y consejos generales sobre enfermedades neurodegenerativas.

Para la consecución de estos objetivos, el presente trabajo persigue la realización de los siguientes hitos:

- Desarrollo de una aplicación que permita al usuario realizar test de aptitudes, así como llevar un seguimiento de sus puntuaciones. Para ello se deberá:
  - Diseñar y crear una batería de test que, a pesar de no tener rigor médico, permita la detección de déficits en las siguientes capacidades intelectuales:
    - Memoria.
    - Reconocimiento de imágenes y rostros.
    - Lenguaje.
    - Análisis matemático.
  - Desarrollar la aplicación con tecnologías aptas para ejecutarse en un dispositivo móvil.
  - Crear, almacenar y mostrar los datos relacionados a las puntuaciones que el usuario consiga a lo largo del tiempo.
  - Mostrar dichos datos con un formato que facilite la legibilidad y el seguimiento de la evolución de las puntuaciones (gráficas, diagramas,etc).
- Implementación de un *backend* que permita desarrollar procesos de autenticación de usuarios y de almacenamiento persistente de datos en una base de datos. Para ello, se debe:
  - Implementar un sistema que permita la creación y autenticación de los usuarios de la aplicación.
  - Diseñar y crear una base de datos que nos permita almacenar la información relacionada con cada usuario, así como sus puntuaciones y la batería de preguntas utilizadas en los diversos tests.
- Estudio y búsqueda de información relevante sobre enfermedades neurodegenerativas con el objetivo de que la aplicación pueda brindar información veraz y contrastada.
- Seguir buenas prácticas durante todas las etapas del proceso de desarrollo. Generar código de calidad acorde a los estándares de las tecnologías empleadas.

1.3 Metodología

El gran apogeo en el sector del desarrollo de *software*, así como las diversas denominadas *'crisis del software'*, han puesto de manifiesto la importancia de elegir una correcta metodología a la hora de abordar este tipo de proyectos. En esta sección describiremos la metodología seleccionada en el desarrollo de este proyecto y se esgrimirán las causas de esta elección.

1.3.1. Matriz Stacey, evaluación de la complejidad del proyecto.

La matriz de Stacey [1] es un mapa que nos permite evaluar la complejidad de un proyecto en función de dos ejes cartesianos. El eje X (abscisas) representa el grado de certeza que se tiene respecto a un problema, mientras que el eje Y (ordenadas) representa el grado de acuerdo de los integrantes del proyecto respecto a dicho problema.

Esta matriz ha sido utilizada en el mundo del desarrollo de *software* para evaluar el tipo de metodología a seguir en nuevos proyectos. En estos casos, el eje X representa el nivel de conocimiento de las tecnologías utilizadas (certeza). Por su parte, el eje Y representa el nivel de concreción de los requerimientos (acuerdo).

Atendiendo al proyecto que se va a desarrollar se han estipulado los siguientes valores para cada uno de los ejes:

- Nivel de conocimiento de las tecnologías utilizadas: A pesar de poseer conocimientos en los procesos de desarrollo de *software*, este proyecto representa la segunda incursión del desarrollador del mismo en el mundo de las aplicaciones para dispositivos móviles. Por lo tanto, al estar familiarizado con el lenguaje de programación utilizado y conocer los *frameworks* con los que se desarrollará el proyecto, se considera que el nivel de conocimiento en las tecnologías utilizadas es medio-alto.
- Nivel de concreción de los requerimientos: La elicitación de requisitos y la definición de objetivos son etapas determinantes en el desarrollo de proyectos *software*. En este caso, ambos apartados han sido evaluados y concretados en los primeros compases de la creación de este proyecto. Sin embargo, la naturaleza de un trabajo fin de grado puede conllevar la modificación, eliminación o incluso la creación de objetivos para lograr ajustarse correctamente a plazos y correcciones. Por todo esto, se podría considerar que el nivel de concreción de los requerimientos es medio.

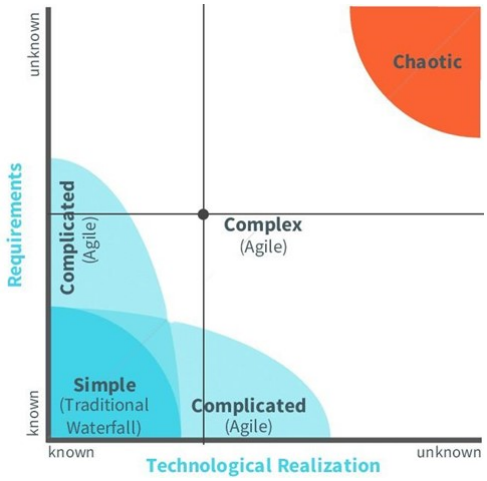


Figura 1.1: Matriz de Stacey adecuada a proyectos software.

Si representamos ambos niveles en la matriz Stacey (Figura 1.1) se puede comprobar que el punto de intersección se encuentra dentro del espectro de lo complejo. Para los proyectos englobados dentro de este espectro se recomienda la utilización de metodologías ágiles. Estas, debido a su gran capacidad adaptativa, nos permitirán responder a las posibles modificaciones en los requisitos y a los posibles cambios y errores en el uso de las tecnologías.

1.3.2. Introducción a la metodología Kanban.

La metodología *Kanban* [2] surgió de la mano de la empresa japonesa *Toyota*. Su nombre proviene del término japones *'Kanban'*, que significa *'tablero visual'*.

Como su propio nombre indica, la metodología *Kanban* está basada en el uso de un tablero visual dividido en columnas en el cual las diferentes tareas, a modo de tarjeta, se van situando en cada una de las columnas en función de su estado.

A pesar de no existir un estándar acerca del número y tipo de columnas que poseen los tableros *Kanban*, la gran mayoría de proyectos que aplican esta metodología presentan cuatro estados indispensables:

- *To do* (*backlog*): Este estado agrupa todas aquellas tareas pendientes que no han sido asociadas a ningún miembro del equipo.
- *Work in progress* (WIP): En esta columna se encuentran las tareas que ya han sido asignadas a uno o varios miembros del proyecto. Tareas en las que se está trabajando actualmente.
- *Blocked*: El estado *blocked* recoge las tareas que se encuentran bloqueadas, ya sea por motivos relativos a la propia tarea o por posibles dependencias con el resto de tareas que se están desarrollando.
- *Done*: Estado final, en él se encuentran todas las tareas que han sido finalizadas.

Estos cuatro estados serán los que se incorporarán al tablero que se usará durante el desarrollo de este proyecto.

1.3.3. Aproximación personal a la metodología Kanban.

A diferencia de otras metodologías *Agile* como *SCRUM* [3], *Kanban* carece de unos estándares concisos. La metodología *Kanban* proporciona herramientas, prácticas y principios para orientar el desarrollo ágil de un proyecto, pero no establece unos pasos claros y definidos sobre cómo debe manejarse. Es por esto que, en el presente proyecto, se ha decidido aplicar esta metodología mediante el uso de ciclos similares a los *sprint* de *SCRUM*. Durante estos ciclos, de una semana de duración, se podrán distinguir las siguientes etapas:

- Se dedicará el primer día de cada nuevo ciclo para seleccionar las tareas que se desean completar durante dicho ciclo. Cada una de las tareas seleccionadas deberá corresponder con una tarjeta homónima en la columna *'To do'*.
- Durante el resto de días previos al último día del ciclo se realizarán las tareas que se han seleccionado. Siempre actualizando el estado de dichas tareas mediante las tarjetas del tablero visual.

- El último día del ciclo se evaluarán las tarjetas que no se encuentren en el estado *'Done'*. Todas las tareas no finalizadas deberán actualizarse y añadirse en la columna *'To Do'* para ser terminadas durante el siguiente ciclo.

1.4 Estructura

El resto de la memoria se ha dividido en los siguientes seis capítulos:

2. Estado del arte: En este apartado se realiza un estudio de soluciones similares a nuestra propuesta, lo que nos permite contextualizar y definir mejor las características de nuestra aplicación en comparación con los productos ya existentes.

3. Análisis del problema: Recoge el análisis de los requisitos que debe cumplir el *software* que se pretende desarrollar. En este apartado también se presenta un diagrama UML para nuestro sistema de almacenamiento de datos persistentes.

4. Diseño de la solución: En este capítulo se recoge todas las decisiones de diseño que deben tomarse antes de iniciar la implementación del código. En esta parte se aborda el diseño de los test y preguntas, las pautas para crear nuestra interfaz de usuario, la arquitectura de *software* que se utiliza y se describen todas las tecnología que se utilizarán durante el proyecto.

5. Desarrollo de la solución: Este capítulo describe cómo se ha realizado la implementación de cada una de las capas de nuestra aplicación y cómo han sido utilizados los servicios externos de los que se hace uso. También se expone el resultado final de la implementación y como sería una sesión habitual de uso de la aplicación.

6. Pruebas: Este apartado recoge las diferentes pruebas llevadas a cabo para testear el correcto funcionamiento de la aplicación.

7. Conclusión: En este apartado se realiza un ejercicio de retrospectiva y se determina si los objetivos expuestos al principio de la memoria han sido logrados con éxito. También se relaciona el trabajo realizado con todos los conocimientos obtenidos durante los estudios cursados. Por último, se exponen las diversas propuestas que pretenden llevar a cabo en futuras versiones de la aplicación.

Además de los capítulos mencionados, esta memoria contiene tres apéndices:

- El apéndice A recoge varios ejemplos de código para mejorar el entendimiento de como se ha desarrollado las diversas partes de la aplicación.

- El apéndice B contiene los *mockups* generados durante nuestra etapa de diseño de la interfaz.

- El apéndice C relaciona este trabajo con los Objetivos de Desarrollo Sostenible (ODS).

CAPÍTULO 2
Estado del arte

Uno de los primeros pasos para el desarrollo de un proyecto *software* es la investigación del contexto actual. En este apartado, trataremos de explorar aplicaciones similares a la que se va a desarrollar, con el objetivo de presentar una propuesta que destaque y se enmarque correctamente dentro de la situación actual del mundo de las aplicaciones móviles.

2.1 Papel de las aplicaciones móviles en el problema de las enfermedades neurodegenerativas.

La llegada e instauración de los teléfonos inteligentes ha supuesto un gran cambio en la manera de relacionarnos, compartir información e incluso en nuestros hábitos diarios. La enorme gama de aplicaciones existentes, sumado a la naturaleza portátil de los teléfonos inteligentes, facilita enormemente el seguimiento de la evolución de diversos ámbitos de nuestra vida. Todo esto sitúa a los teléfonos inteligentes como una plataforma perfecta para el desarrollo de aplicaciones que ayuden a pacientes afectados por las enfermedades neurodegenerativas (EN).

En la actualidad, existe gran cantidad de aplicaciones destinadas a ayudar a personas con enfermedades neurodegenerativas. Como resultaría inabarcable destacar todas y cada una de ellas, en este apartado se tratará de realizar una breve descripción de aquellas que se han considerado de especial interés. Para realizar esta selección se han tenido en cuenta dos criterios:

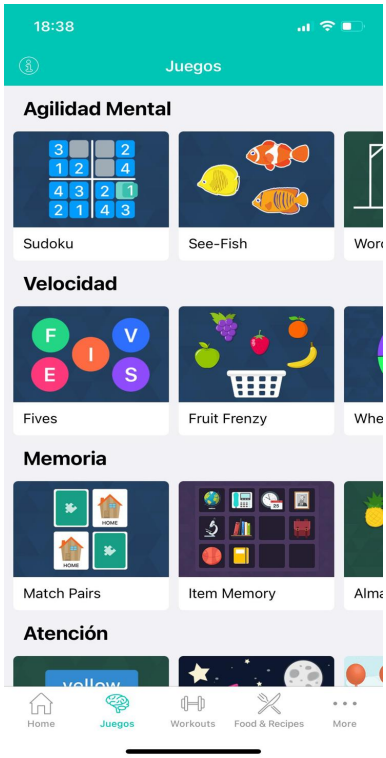
I. Nivel de popularidad en la *app Store* de *iOS*.

II. Aplicaciones destacadas dentro del articulo *'Aplicaciones móviles en la enfermedad de Alzheimer. Una revisión sistemática de la literatura'* [4] publicado en 2019.

2.1.1. MindMate

MindMate [5] es una aplicación popular diseñada para personas con demencia, Alzheimer y otras enfermedades neurodegenerativas. Al registrarse la aplicación realiza preguntas sobre la edad y condición médica del usuario, así como un breve y sencillo test de memoria (Figura 2.1a).

(a) MindMate: test y preguntas al iniciar la aplicación por primera vez.



(b) MindMate: sección con juegos interactivos.

Figura 2.1: Imágenes de MindMate

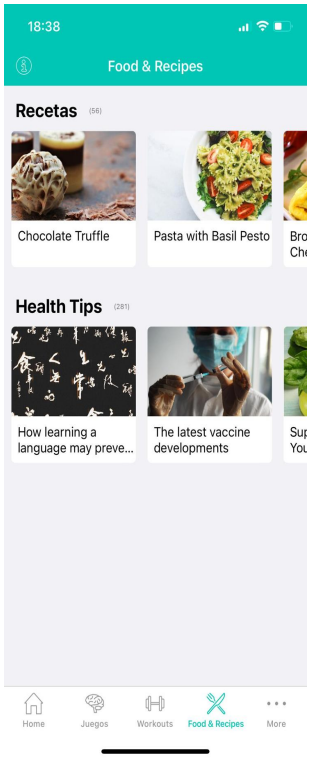
La aplicación cuenta con varias secciones, entre las cuales se encuentran:

- Sección con varios juegos interactivos para mejorar las capacidades cognitivas. Dichos juegos se agrupan en las categorías: Agilidad mental, velocidad, memoria y atención (Figura 2.1b).
- Acceso a información sobre las últimas investigaciones en el campo de la demencia y otras enfermedades neurodegenerativas.
- Recetario saludable con una amplia gama de platos (Figura 2.2a).
- Rutinas de ejercicio físico centradas en ejercicios de cardio, yoga y aeróbic (Figura 2.2b).

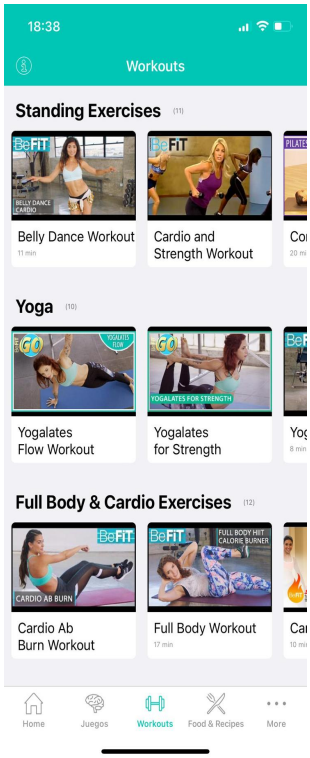
En la pantalla principal de la aplicación se proponen 5 actividades diarias: un juego, una receta, un consejo de salud, un video informativo sobre las EN y una rutina de ejercicio (Figura 2.3).

La aplicación incluye un apartado de estadísticas que permite medir el cumplimiento de los objetivos diarios a lo largo de la semana. Además, este apartado permite vincular los datos de la aplicación salud de un dispositivo *iOS* para mostrar métricas de actividad física y calidad de sueño.

Por último, la aplicación permite la creación de un banco de recuerdos, el cual engloba varios álbumes de fotos que el usuario puede consultar en cualquier momento.



(a) MindMate: sección con recetario saludable.



(b) MindMate: sección con rutinas de ejercicio.

Figura 2.2: Imágenes de la aplicación MindMate

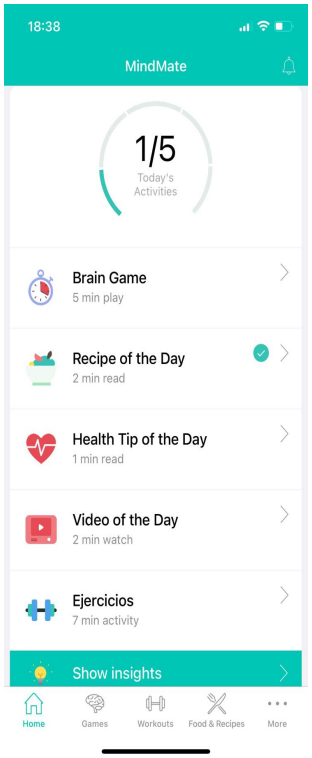
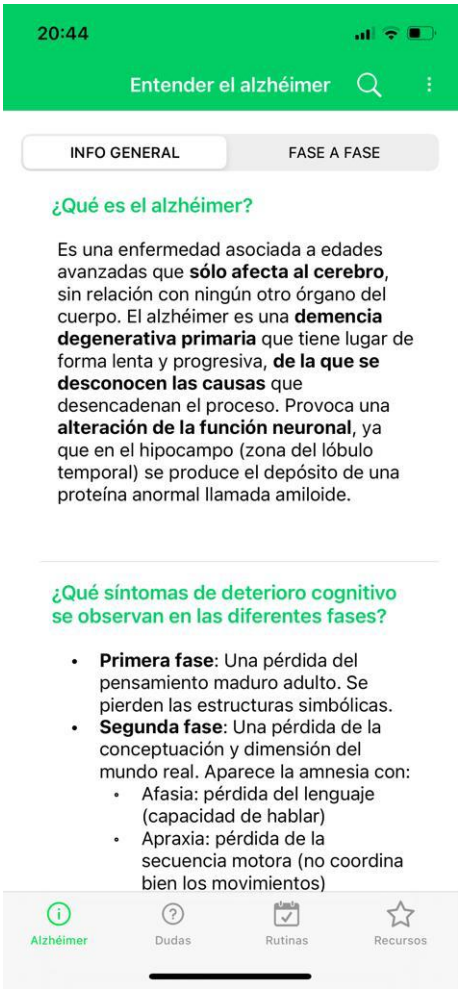


Figura 2.3: MindMate: pantalla con las 5 actividades diarias propuestas por la aplicación.

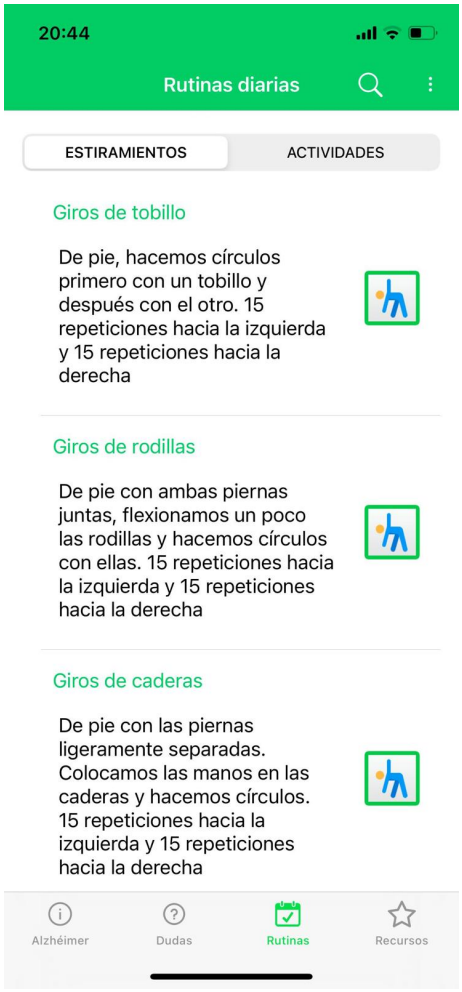
2.1.2. YoTeCuido Alzheimer

YoTeCuido Alzheimer [6] es una aplicación desarrollada por *Lapisoft Projects SL* en colaboración con AFACO (Asociación de Familiares de enfermos de Alzheimer de A Coruña). Está destinada a proporcionar información relevante sobre la enfermedad del Alzheimer. La aplicación está diseñada para ayudar tanto a afectados como a cuidadores de personas con esta enfermedad.

La aplicación proporciona información sobre las diferentes fases de la enfermedad (Figura 2.4a), así como sobre los problemas diarios que puede causar y las rutinas que los afectados pueden realizar para mejorar su condición (Figura 2.4b). Además de todo esto, la aplicación posee una sección de dudas ordenadas por múltiples categorías (Figura 2.5) y otra con contactos de interés.



(a) YoTeCuido Alzheimer: sección de información relevante sobre la enfermedad del Alzheimer.



(b) YoTeCuido Alzheimer: sección con rutinas de ejercicio diarias.

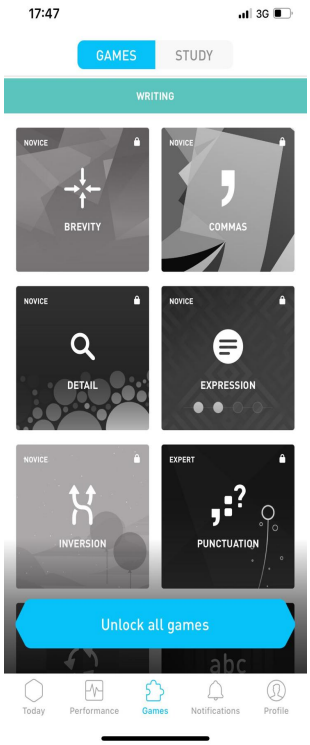
Figura 2.4: Imágenes de la aplicación YoTeCuido Alzheimer

Figura 2.5: YoTeCuido Alzheimer: sección de resolución de dudas sobre diversas categorías.

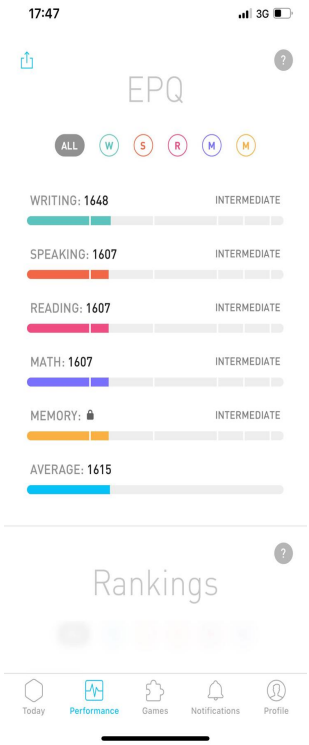
2.1.3. Elevate

Elevate [7] es una aplicación diseñada para mejorar la expresión oral, la atención, la memorización y el cálculo mental. A pesar de no ser una aplicación enteramente dedicada a personas con EN, puede servir como ejercicio cognitivo para aquellas personas afectadas.

La aplicación cuenta con un surtido de hasta 40 juegos distintos destinados al entrenamiento cognitivo (Figura 2.6a), una sección de estadísticas que muestra el desempeño en los juegos pertenecientes a cada una de las distintas categorías (Figura 2.6b) y un apartado con distintos retos para motivar el uso de la aplicación diariamente (Figura 2.7).



(a) Elevate: sección para poder seleccionar uno de entre los 40 juegos cognitivos disponibles en la aplicación.



(b) Elevate: sección con métricas y estadísticas del desempeño del usuario.

Figura 2.6: Imágenes de la aplicación Elevate

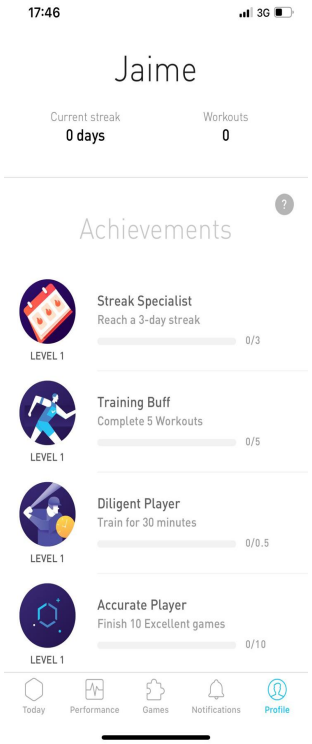


Figura 2.7: Elevate: sección de logros con diversos desafíos y un contador de la racha de días que el usuario lleva usando la aplicación.

2.1.4. Stimulus Home y Pro

Stimulus [8] es una aplicación multiplataforma de estimulación cognitiva que cuenta con más de 50 actividades distintas con diversos niveles de dificultad. Sus actividades se engloban en 10 capacidades funcionales que van desde el cálculo y el razonamiento hasta la memoria a largo plazo.

Actualmente, *Stimulus* cuenta con dos versiones distintas de su aplicación: *Stimulus Home* y *Stimulus Pro*. Cada una de estas atiende a un ámbito específico.

Stimulus Pro se concibe para un uso por profesionales. Esta versión posibilita a los terapeutas llevar el seguimiento de varios pacientes. La aplicación permite diseñar sesiones de un conjunto de actividades específicas para cada paciente. Además de lo ya mencionado, la aplicación también cuenta con un apartado detallado de estadísticas que muestra el desempeño del paciente en las diversas sesiones e incluso permite generar informes con dichos datos en diversos formatos como *Excel* o PDF.

Por otro lado, *Stimulus Home* es la versión doméstica de la aplicación. Esta versión permite a los pacientes sanos realizar sus propias sesiones de actividades de manera autónoma (Figura 2.8). Además, esta versión puede usarse para tratamientos de manera remota, ya que se puede vincular a una cuenta de la versión Pro para llevar un seguimiento remoto. Por lo demás, la versión Home cuenta con las mismas actividades que la aplicación Pro y también posee un apartado donde observar las métricas de los desempeños en las distintas sesiones (Figura 2.9).

Cabe destacar que el equipo de *Stimulus* ha realizado diversos análisis de estudios científicos para poder determinar la eficacia de la estimulación cognitiva por medio de programas y poder determinar empíricamente qué actividades y características debería presentar su aplicación.

Figura 2.8: Stimulus Home: Sección con las diversas actividades de estimulación cognitiva que el usuario puede realizar en la aplicación.

Figura 2.9: Stimulus Home: Sección de estadísticas en las diversas capacidades funcionales.

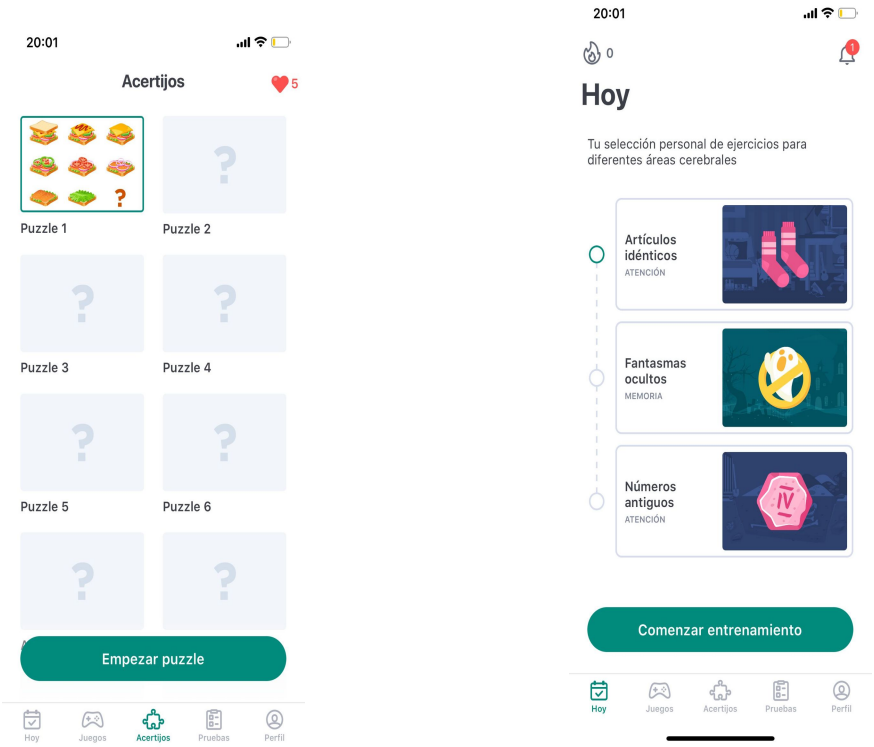
2.1.5. Impulse

Impulse [9] es una aplicación desarrollada por la compañía GMRD APPS LIMITED. A pesar de ser una aplicación centrada en la estimulación cognitiva, *Impulse* no está enteramente dedicada a personas que padezcan EN. La aplicación favorece la mejora cognitiva mediante el uso de juegos y acertijos enfocados en diversas áreas (Figura 2.10a).

Al abrir la aplicación por primera vez, *Impulse* realiza un pequeño test con preguntas personales sobre la edad, género y capacidades que le interesa ejercitar al usuario. Con todos estos datos, *Impulse* es capaz de generar diariamente una selección personal de actividades para cada uno de sus usuarios (Figura 2.10b).

Impulse también cuenta con una sección de retos y un apartado que calcula la racha de días seguidos que el usuario lleva realizando sus actividades diarias (Figura 2.11).

Al igual que muchas de las aplicaciones mencionadas, *Impulse* cuenta con un apartado de métricas y estadísticas que permiten monitorizar la evolución de las puntuaciones del usuario en los diversos apartados y observar las variaciones de estas a lo largo del tiempo.



(a) Impulse: sección de acertijos.

(b) Impulse: sección personal con ejercicios diarios propuestos por la aplicación.

Figura 2.10: Imágenes de la aplicación Impulse

Figura 2.11: Impulse: sección de logros con diversos desafíos y un contador de la racha de días que el usuario lleva usando la aplicación.

2.2 Crítica al estado del arte

2.2.1. Propuesta

Tras realizar un breve estudio de la situación actual de las aplicaciones móviles para pacientes con enfermedades neurodegenerativas, en este apartado se pretende exponer nuestra propuesta.

MindTest ha sido el nombre escogido para bautizar a la aplicación. *MindTest* se concibe como una aplicación móvil multiplataforma que podrá ejecutarse en dispositivos *iOS* y *Android*. A continuación, se describirán las principales funciones que poseerá la aplicación:

- Capacidad para realizar tests cognitivos: La estimulación cognitiva es uno de los principales propósitos perseguidos en el desarrollo de esta aplicación. La capacidad para realizar tests que estimulen y evalúen diversas capacidades del usuario es una funcionalidad imprescindible en esta propuesta.
- Consulta de estadísticas: Al igual que la estimulación cognitiva, la detección precoz de síntomas es otro de los objetivos perseguidos por la aplicación. Por esta razón, se considera de vital importancia que la aplicación permita consultar métricas y estadísticas que pongan de manifiesto posibles deficiencias o deterioros en alguna de las capacidades evaluadas.
- Consulta de información y recursos útiles y contrastados sobre las EN: Brindar información acerca de las enfermedades neurodegenerativas puede ser sumamente útil para una mejor comprensión de estas por parte de los usuarios.
- Generación de informes de estadísticas sobre el usuario: La capacidad para generar informes que reflejen la evolución de estadísticas del usuario, permite facilitar el seguimiento de su estado. Además, estos informes podrán ser enviados a terceros configurando correctamente un correo electrónico de contacto.
- Configuración de un filtro de inactividad: Con el objetivo de favorecer el uso de la aplicación, esta contará con la posibilidad de establecer un filtro de inactividad que notifique a un supervisor, vía correo electrónico, cuando el usuario lleva más de un determinado número de días sin realizar tests.

Además de las funcionalidades descritas, se ha buscado añadir características diferenciadoras que permitan dotar al proyecto de originalidad. Esta diferenciación surge desde la capacidad de establecer un familiar o persona cercana como usuario "supervisor". Este supervisor será capaz de recibir los informes periódicos y avisos de inactividad que genere la aplicación.

Cabe destacar que muchas otras funcionalidades presentes en el conjunto de aplicaciones estudiadas han quedado fuera de nuestra propuesta. Entre ellas destacamos :

- Consejos de alimentación saludables: No se cuenta con los conocimientos necesarios y se considera que queda fuera del propósito de la aplicación
- Rutinas de ejercicios: Similar a la funcionalidad anterior. No se posee conocimiento en este ámbito y queda fuera del enfoque del proyecto.
- Creación de un banco de recuerdos: Supone un mayor desafío técnico. Esta funcionalidad se podría contemplar como un trabajo a futuro.

2.2.2. Tabla comparativa

La Tabla 2.1 compara las aplicaciones revisadas en el apartado anterior en base a sus funcionalidades y las plataformas en las que se encuentran disponibles.

	MindMate	YoTeCuido Alzheimer	Elevate	Stimulus	Impulse	MindTest
Capacidad para realizar test/juegos	sí	no	sí	sí	sí	sí
Información relevante sobre las EN	sí	sí	no	?	no	sí
Estadísticas sobre el desempeño del usuario	sí	no	sí	sí	sí	sí
Consejos de alimentación saludable	sí	sí	no	no	no	no
Rutinas de ejercicio físico	sí	sí	no	no	no	no
Creación de banco de recuerdos	sí	no	no	no	no	no
Envío de informes periódicos para monitorizar la evolución del usuario	no	no	no	Disponible en su versión Pro	no	sí
Notificaciones que recuerden la necesidad de realizar el test diario	no	no	no	no	sí	Trabajo a futuro
Capacidad para establecer un contacto cercano que reciba los informes y avisos de inactividad	no	no	no	Su versión Pro es utilizada por un cuidador	no	sí
Alerta de alto periodo de inactividad	no	no	no	no	no	sí
Plataformas Disponibles	iOS PC	Android iOS	Android iOS	Android iOS PC	iOS	Android iOS

Tabla 2.1: Tabla comparativa de las diferentes aplicaciones de interés y sus diferentes funcionalidades

CAPÍTULO 3
Análisis del problema

Tras el estudio e investigación de las diversas propuestas realizado en el capítulo anterior, es necesario realizar un análisis del problema que nos permita detectar posibles oportunidades de innovación y establecer unos objetivos claros sobre las funciones que la aplicación debe realizar. En este caso, se ha realizado un análisis de requisitos siguiendo el estándar *IEEE 830* y se ha diseñado el diagrama UML que define el modelo de datos de nuestra base de datos.

3.1 Especificación de requisitos

Esta especificación de requisitos se ha realizado basándose en las directrices que se recogen en el estándar *IEEE 830* [10].

3.1.1. Propósito

La presente sección tiene como principal propósito definir los requisitos funcionales y no funcionales de una aplicación destinada a la detección temprana de síntomas de EN.

3.1.2. Ámbito del sistema

Esta especificación definirá las distintas funcionalidades que la aplicación podrá realizar. En ningún caso se debe asumir una validez médica de los posibles resultados. Esta aplicación no tiene como propósito realizar un diagnóstico de rigor médico sobre el estado del usuario.

3.1.3. Personal involucrado

Nombre	Jaime Fernández Plaza
Rol	Diseñador y programador
Categoría	Estudiante de grado de ingeniería
Responsabilidad	Elicitación de requisitos, elección de metodología, diseño de la aplicación, programación, despliegue y testeo
Información de contacto	jferpla@inf.upv.es

Tabla 3.1: Datos del personal involucrado

3.1.4. Características de los usuarios

Esta aplicación cuenta con dos principales perfiles de usuario:

El primero es una persona de edad avanzada que utiliza la aplicación para realizar los diversos test. Por lo general estos usuarios carecen de grandes aptitudes para la utilización de dispositivos electrónicos.

El segundo perfil se refiere a un familiar o persona cercana que pueda realizar cierto seguimiento de los resultados de los test.

3.1.5. Restricciones del Sistema

Existen posibles restricciones que puedan impedir al usuario el uso de la aplicación. Estas restricciones son:

- Necesidad de conexión a internet.
- Necesidad de un dispositivo móvil con las características necesarias para el correcto funcionamiento de la aplicación.

3.1.6. Requisitos Funcionales

Número de requisito	RF1
Nombre	Registrar Usuario
Características	El sistema permitirá a los usuarios que no se hayan registrado previamente hacerlo.
Descripción	El sistema permitirá al usuario registrarse. Para ello, el usuario deberá proporcionar los siguientes datos: nombre y apellidos, nombre de usuario, correo electrónico y contraseña.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta

Tabla 3.2: Requisito funcional: Registrar Usuario

Número de requisito	RF2
Nombre	Autenticación de Usuario
Características	Los usuarios podrán autenticarse para entrar a la aplicación.
Descripción	Los usuarios que hayan sido registrados previamente podrán autenticarse para entrar a la aplicación.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta

Tabla 3.3: Requisito funcional: Autenticación de Usuario

Número de requisito	RF3
Nombre	Seleccionar test por categoría
Características	El sistema permitirá al usuario seleccionar una de las distintas categorías que se evalúan en los test completos para poder realizar un test de esa habilidad específica.
Descripción	El sistema mostrará las diferentes habilidades que se evalúan en un test completo (memoria, razonamiento numérico, etc.), para que el usuario pueda seleccionar aquella categoría en la que quiere realizar el test.
Requisitos no funcionales	RNF1, RNF2, RNF3
Prioridad	Alta

Tabla 3.4: Requisito funcional: Seleccionar test por categoría

Número de requisito	RF4
Nombre	Seleccionar test completo
Características	El usuario seleccionará realizar un test completo que presenta preguntas de todas las categorías. Dicho test evaluará todas las habilidades del usuario.
Descripción	El usuario podrá seleccionar realizar un test completo, el cual constará, al menos, de una pregunta de cada categoría.
Requisitos no funcionales	RNF1, RNF2, RNF3
Prioridad	Alta

Tabla 3.5: Requisito funcional: Seleccionar test completo

Número de requisito	RF5
Nombre	Realizar Test.
Características	El usuario podrá realizar el test que haya seleccionado previamente.
Descripción	El sistema realizará una llamada a la base de datos donde se almacenan los test y las preguntas. Una vez reciba las preguntas necesarias para el test a realizar, será capaz de mostrárselas al usuario. La mayoría de preguntas del test serán tipo test con 4 posibles respuestas, de las cuales solo una será correcta. También existirán preguntas en las que el usuario podrá introducir la respuesta manualmente.
Requisitos no funcionales	RNF1, RNF2, RNF3
Prioridad	Alta

Tabla 3.6: Requisito funcional: Realizar Test

Número de requisito	RF6
Nombre	Guardar resultados de los tests.
Características	El sistema almacenará la información referente a los resultados que el usuario obtenga en los distintos tests.
Descripción	Una vez finalizado el test, el sistema deberá almacenar automáticamente la puntuación, así como la información relativa al resultado del test realizado.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta

Tabla 3.7: Requisito funcional: Guardar resultados de los tests

Número de requisito	RF7
Nombre	Mostrar seguimiento de los resultados.
Características	El usuario podrá realizar un seguimiento de los resultados de los diversos test que ha realizado.
Descripción	El sistema mostrará el resultado de los test realizados. Los datos podrán visualizarse en diversos formatos (gráficas, tablas, etc.). El usuario podrá filtrar los datos que se le muestran en función de la fecha de realización del test y de las distintas categorías disponibles.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta

Tabla 3.8: Requisito funcional: Mostrar seguimiento de los resultados

Número de requisito	RF8
Nombre	Configurar el seguimiento de los test realizados por el usuario.
Características	El usuario podrá ingresar el correo electrónico de un familiar o persona cercana para que, en caso de no realizar los test durante un determinado periodo de tiempo, el sistema lo notifique enviando un aviso al correo electrónico indicado.
Descripción	El sistema permitirá al usuario ingresar el correo electrónico de un familiar o persona cercana, así como seleccionar el margen de tiempo del que dispone el usuario para realizar los test antes de que se le envíe una notificación.
Requisitos no funcionales	RNF1, RNF2, RNF3, RNF4
Prioridad	Alta

Tabla 3.9: Requisito funcional: Configurar el seguimiento de los test realizados por el usuario

Número de requisito	RF9
Nombre	Mostrar información relevante sobre EN.
Características	El usuario podrá visualizar artículos e información sobre distintas enfermedades neurodegenerativas.
Descripción	El sistema mostrará información sobre enfermedades neurodegenerativas (síntomas, recomendaciones, artículos relacionados, etc.).
Requisitos no funcionales	RNF1, RNF2, RNF3
Prioridad	Alta

Tabla 3.10: Requisito funcional: Mostrar información relevante sobre EN

Número de requisito	RF10
Nombre	Envío de informes periódicos con estadísticas sobre la evolución del usuario.
Características	El usuario podrá ingresar el correo electrónico de un familiar o persona cercana para que el sistema envíe periódicamente informes sobre el desempeño y evolución del usuario en las diversas áreas.
Descripción	El sistema permitirá al usuario ingresar el correo electrónico de un familiar o persona cercana para que esta reciba los informes. En el caso de que no se introduzca ninguna información de un contacto cercano, los informes se enviarán al correo electrónico usado al registrarse.
Requisitos no funcionales	RNF1 RNF2 RNF3 RNF4
Prioridad	Alta

Tabla 3.11: Requisito funcional: Envío de informes periódicos con estadísticas sobre la evolución del usuario.

3.1.7. Requisitos No Funcionales

Número de requisito	RNF1
Descripción	El sistema debe tener una interfaz intuitiva y sencilla que facilite su uso por parte de usuarios poco familiarizados con los teléfonos inteligentes.
Prioridad	Alta

Tabla 3.12: Requisito no funcional RNF1

Número de requisito	RNF2
Descripción	El sistema debe estar diseñado para implantarse en dispositivos móviles y tablets.
Prioridad	Alta

Tabla 3.13: Requisito no funcional RNF2

Número de requisito	RNF3
Descripción	La interfaz del sistema debe ser capaz de adaptarse a las dimensiones de los diferentes dispositivos.
Prioridad	Alta

Tabla 3.14: Requisito no funcional RNF3

Número de requisito	RNF4
Descripción	El sistema debe garantizar un correcto uso de los datos personales.
Prioridad	Alta

Tabla 3.15: Requisito no funcional RNF4

3.2 Diagrama UML

El uso de un almacenamiento persistente de cierta información conlleva la necesidad de diseñar e implementar una base de datos que cumpla dicho propósito. La Figura 3.1 muestra el diagrama UML [11] que representa el modelo de este sistema de información junto con sus relaciones y restricciones. A continuación, se describirán las distintas clases que conforman el diagrama y el propósito de cada uno de sus atributos:

- Usuario: Clase que representa los datos del usuario.
  - userUid: Cadena de texto que identifica al usuario. Es único para cada usuario registrado.
  - email: Cuenta de correo electrónico que el usuario usará para autenticarse.
  - contraseña: Contraseña que el usuario usará para autenticarse.
  - nombre: Nombre del usuario.
  - emailContacto: Correo electrónico de persona cercana al usuario que desee recibir información periódica sobre los resultados obtenidos y notificaciones en caso de que los tests no se realicen.
  - filtro: Número máximo de días que pueden pasar sin el usuario realice algún test antes de que se envié un aviso de inactividad.
- Puntuación: Representa las puntuaciones que obtiene un usuario al realizar un test.
  - userUid: Cadena de texto que identifica al usuario que ha realizado el test.
  - testUid: Cadena de texto que identifica el test realizado.
  - puntuación: Número que indica la cantidad de preguntas acertadas por el usuario.

- fecha: Fecha de realización del test.

- Test: Clase que representa la información de un test.
  - testId: Cadena de texto que identifica al test. Es único para cada test almacenado.
  - categoría: Categoría a la que pertenece el test.
- Pertenece: Clase de asociación entre la clase Test y Pregunta.
  - testUid: Cadena de texto que identifica al test.
  - preguntaUid: Cadena de texto que identifica a la pregunta.
- Pregunta: Clase que representa la información de cada pregunta.
  - preguntaUid: Cadena de texto que identifica la pregunta. Es único para cada pregunta almacenada.
  - categoría: Categoría a la que pertenece la pregunta.
  - opciones: Lista de las cuatro posibles respuestas que puede tener la pregunta.
  - respuesta: Respuesta correcta.
  - url: dirección online donde esta almacenada la imagen asociada a esa pregunta.

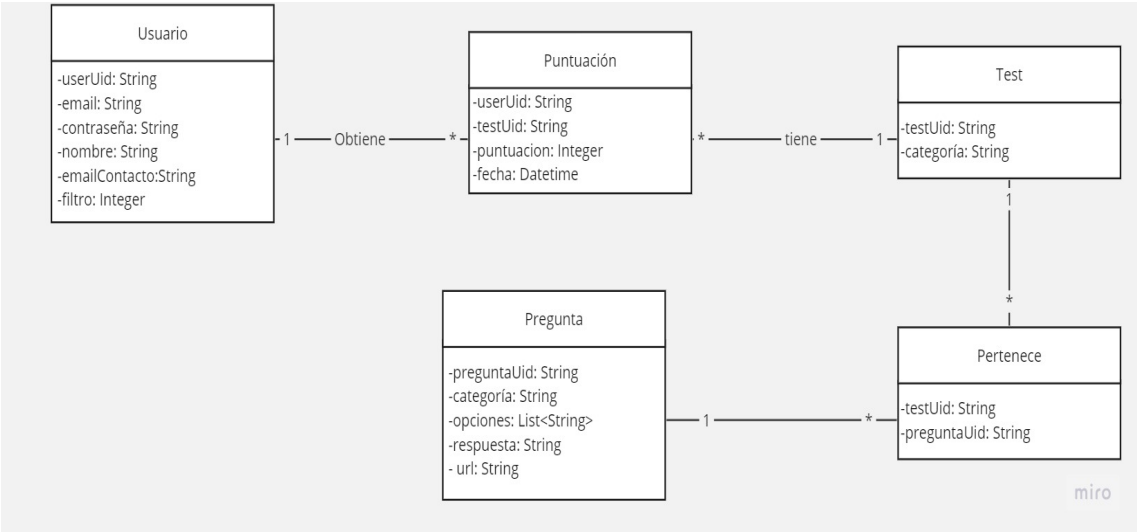


Figura 3.1: Diagrama UML de la base de datos utilizada para almacenar usuarios, puntuaciones, test y preguntas.

CAPÍTULO 4
Diseño de la solución

Tras la elicitación de requisitos, comienza una de las etapas más determinantes en el mundo de los proyectos de *software*. La etapa de diseño es fundamental para estructurar un correcto desarrollo de la aplicación, así como para definir claramente los objetivos que este desarrollo persigue. En este apartado, se abordarán las decisiones tomadas para definir cómo se plantea la implementación de los diversos aspectos de nuestra aplicación.

4.1 Diseño de los tests y elección de preguntas

La degradación neuronal conlleva un deterioro en diversas funciones cognitivas. Dentro de esta sintomatología [12], se destacan:

- Amnesia: pérdida de memoria que dificulta la consolidación de nuevos aprendizajes y el recuerdo de información ya aprendida.
- Apraxia: deterioro de las capacidades motrices y alteración en el control voluntario de gestos y movimientos.
- Pérdida del lenguaje: dificultad para recordar palabras e incapacidad para formar frases coherentes.
- Pérdida o disminución de las capacidades de cálculo y razonamiento.
- Agnosia: Alteraciones en la percepción del entorno que les rodea, dificultades para reconocer objetos de su entorno o elementos en una imagen.

Entre los síntomas expuestos, debemos destacar la apraxia debido a la complejidad de evaluar el deterioro de las habilidades motrices y gestuales mediante un dispositivo móvil. Prácticamente la totalidad de los tests existentes para detectar la aparición de este síntoma se basan en la replicación de figuras o símbolos, y requieren la implementación de un sistema que permita al dispositivo móvil reconocer las figuras realizadas por el usuario y compararlas con las que se le han expuesto como objetivo a replicar.

Este aumento de la complejidad en el diseño de tests que evalúen la apraxia nos ha llevado a tomar la decisión de prescindir de este tipo de tests y centrarnos en preguntas que puedan detectar posibles apariciones del resto de síntomas restantes.

Es por todo lo expuesto que las preguntas diseñadas tienen como objetivo evaluar las siguientes categorías:

- Memoria

- Lenguaje
- Cálculo
- Reconocimiento de imágenes y rostros

Cabe destacar que, a falta de una supervisión médica especializada, todas las preguntas han sido diseñadas basándose en el manual *"Volver a Empezar. Ejercicios prácticos de estimulación cognitiva para pacientes de Alzheimer"* [13], publicado por la Fundación ACE (*Alzheimer Centre Educacional*) de Barcelona.

También es importante resaltar que, en el caso hipotético de que la aplicación se comercializara, sería necesario contactar con los autores del manual y negociar el uso de los derechos de autor.

A continuación, se explicará el modelo de preguntas para cada categoría:

4.1.1. Memoria

Cuando hablamos de memoria, podemos distinguir tres tipos:

- Memoria inmediata: encargada de recordar información recién percibida.
- Memoria reciente: encargada de retener información y nuevos recuerdos.
- Memoria remota: encargada de almacenar recuerdos de todas nuestras experiencias.

Los afectados por las enfermedades neurodegenerativas presentan un deterioro notable en la memoria reciente y, por lo tanto, en los procesos de consolidación de nuevo material aprendido. Con el desarrollo de la enfermedad, los afectados comienzan a tener dificultades para evocar recuerdos pertenecientes a la memoria remota.

Nuestras preguntas se han enfocado en estimular y evaluar las memorias inmediata y reciente, ya que la degradación de la memoria remota suele estar asociada a experiencias individuales de cada afectado, como la dificultad para reconocer a personas cercanas o la incapacidad de recordar eventos o vivencias personales. Esto hace que la creación de tests objetivos para evaluar esta degradación sea realmente difícil y se necesitaría adaptar las preguntas a cada individuo.

Teniendo en cuenta lo anterior, las preguntas de esta categoría obedecen al siguiente modelo:

- En primer lugar, se le muestra al usuario una imagen con un conjunto de palabras (Figura 4.1) durante 10 segundos. El usuario tratará de memorizar todas las palabras de la imagen.
- Una vez pasados los 10 segundos aparecerá un test de opción múltiple con 4 palabras de entre las cuales el usuario deberá seleccionar cuál no estaba presente en la imagen anterior.

Como ya hemos mencionado este test es una variación de uno de los test expuestos en el manual, pero ha sido adaptado para presentarse como un test de opción múltiple.

Figura 4.1: Ejemplo de imagen utilizada en las preguntas de memoria. Contiene un conjunto de palabras para memorizar.

4.1.2. Lenguaje

Dentro de las distintas alteraciones del lenguaje se pueden distinguir tres casos:

- Afasia: alteración del lenguaje oral que genera dificultades para la expresión y comprensión oral
- Alexias: dificultad en la comprensión lectora o directamente incapacidad para leer.
- Agrafias: dificultad en el lenguaje escrito.

Se ha considerado que las preguntas de esta categoría tengan como principal objetivo evaluar las degradaciones producidas por alexias y agrafias. La implementación de preguntas dedicadas a la detección de afasias requieren de algún sistema que permita reconocer los mensajes orales del usuario y que sea capaz de evaluar si estos son correctos o presentan síntomas de alguna alteración en el lenguaje. Es por todo esto que en los dos modelos de preguntas propuestos se hace especial hincapié en la degradación del lenguaje escrito.

- En el primer modelo de pregunta se mostrará una breve descripción. El usuario deberá seleccionar cual de las 4 opciones del test concuerda con la definición mostrada.
- En el segundo modelo la aplicación mostrará una frase incompleta. El usuario deberá seleccionar cuál de las cuatro opciones posibles es la correcta para completar la frase.

Ambos modelos están enfocados en intentar estimular el uso de la comprensión lectora y de la escritura.

4.1.3. Cálculo

En los primeros compases de las enfermedades neurodegenerativas se observa un empeoramiento en la capacidad de resolución de problemas matemáticos diarios, como podrían ser las compras o ventas y los cambios de divisa. Posteriormente, en etapas más avanzadas, se producen deterioros en el cálculo mental simple y los números comienzan a perder su valor simbólico. En los momentos más avanzados de la enfermedad se puede llegar a dar trastornos de la organización espacial, en la que no se mantiene el orden ni la posición de los dígitos, estos son denominados como acalculia espacial.

Dentro de esta categoría se han seleccionado dos modelos de preguntas:

- El primero consiste en un problema sencillo de actividades de compra o venta. Se mostrará el enunciado del problema y el usuario deberá introducir la solución correcta. Este modelo pretende evaluar los deterioros en las capacidades resolutivas de problemas matemáticos simples que se presentan en el día a día.

- En el segundo modelo la aplicación mostrará una operación combinada. El usuario deberá escoger cual de los cuatro resultados disponibles concuerda con el resultado de la operación. Debido a la mayor complejidad que pueden presentar las operaciones combinadas, se ha optado por un test de opción múltiple. Este modelo tiene como principal objetivo la estimulación de las habilidades de cálculo mental.

4.1.4. Reconocimiento de imágenes y rostros

La agnosia consiste en una alteración en el reconocimiento de las características físicas del entorno. Este síntoma puede darse a nivel visual, auditivo, táctil y olfativo. Los afectados por enfermedades neurodegenerativas pueden comenzar a experimentar agnosias visuales en las fases más leves de la enfermedad. En fases más avanzadas el paciente puede comenzar a experimentar agnosias olfativas y táctiles. A su vez, las agnosias visuales se incrementan llegando a causar dificultad en el reconocimiento de rostros de familiares cercanos. En los momentos más graves, las agnosias visuales son extremas, llegando a dificultar que el paciente sea capaz de reconocer su propio rostro.

Nuestros modelos se centran en la ejercitación del reconocimiento de estímulos visuales, ya que las agnosias visuales son las más presentes desde los primeros momentos de estas enfermedades.

- En el primer modelo la aplicación mostrará una imagen de una personalidad conocida dentro de la cultura popular. El usuario deberá elegir cuál de los 4 nombres dados concuerda con la persona que aparece en la imagen. Este modelo nos permite evaluar si existe algún tipo de alteración en el reconocimiento de rostros por parte del usuario.

- El segundo modelo se centrará en el reconocimiento de objetos. La aplicación mostrará una imagen de un objeto cotidiano. Al igual que en el anterior modelo, el usuario deberá seleccionar el nombre correcto de entre las 4 opciones disponibles.

4.1.5. Diseño de los tests

Una vez seleccionado el modelo de las preguntas se han de determinar los diversos tipos de test que la aplicación permitirá realizar. Nuestra aplicación contará con cinco tipos de tests distintos:

- Test de memoria
- Test de lenguaje
- Tets de cálculo
- Tests de reconocimiento
- Test completo

Cada uno de los test cuenta con seis preguntas, excepto el test completo que únicamente cuenta con cuatro (una de cada categoría).

Los tests de categorías tienen como principal objetivo una evaluación más profunda del estado de una de las habilidades del usuario en concreto. Por otro lado, los test completos están ideados para realizarse diariamente y así poder tener una perspectiva general del estado del usuario. Su mayor brevedad y el cambio entre preguntas de diferentes ámbitos contribuye a que su realización sea más amena para los usuarios.

4.2 Diseño de la interfaz de usuario

El diseño de la interfaz de usuario es una etapa sumamente importante a la hora de determinar qué experiencia obtendrá el usuario cuando utilice nuestra aplicación. La gran implantación de los teléfonos inteligentes ha permitido que gran parte de la sociedad esté familiarizada con el uso de aplicaciones para estos dispositivos. Sin embargo, nuestra aplicación está enfocada a un nicho de personas adultas de edad avanzada que tiende a presentar problemas al desenvolverse con el uso de nuevas tecnologías.

La falta de familiaridad con dispositivos móviles, sumado a la disminución de las capacidades cognitivas que pueden experimentar las personas de edad avanzada, provoca que el diseño de la interfaz de usuario sea un elemento vital para decidir si el usuario hará uso de la aplicación o no.

Para el diseño de la interfaz de nuestra aplicación se ha tenido como fuente principal las recomendaciones recogidas en el artículo *"Recomendaciones de Diseño para Mejorar la Experiencia de los Usuarios Adultos Mayores con Facebook en Dispositivos Tablet"* [14]. Las recomendaciones de este artículo se basan en la aplicación de las 10 reglas heurísticas de Nielsen [15] y en un extenso estudio de las guías *"UX Design for Seniors (Ages 65 and older)"* [16] con recomendaciones para el diseño de interfaces destinadas a personas mayores y *"User Experience for Mobile Applications and Websites"* [17] con consejos de diseño en interfaces de aplicaciones móviles. Junto con estas recomendaciones, también se han seguido consejos de los artículos *"Considerations in Designing Human-Computer Interfaces for Elderly People"* [18] y *"Designing User Interfaces for the Elderly: A Systematic Literature Review"* [19] ambos enfocados en las buenas prácticas a la hora de diseñar interfaces para adultos de edad avanzada.

Las pautas y recomendaciones que se han tomado en consideración para el diseño de la interfaz son las siguientes:

- Simplicidad: la disminución de la capacidad de concentración es algo muy presente en las personas de edad avanzada. La simplicidad y sustracción de elementos innecesarios facilita que el usuario sea capaz de centrarse en los apartados relevantes sin dificultad para perder el foco de atención. Se recomienda que no existan en pantalla más de 10 elementos interactuables, en nuestra aplicación; únicamente la pantalla principal supera este número (11 elementos interactuables).

- Dificultad en el manejo del teclado: La pérdida de habilidades motrices puede causar dificultades en el uso del teclado. Es por eso que se ha intentado disminuir el uso de teclados en todos los apartados de la aplicación. Los usuarios solo tendrán que interactuar con el teclado en los procesos de registro o autenticación y en determinados test. La implantación de test de opción múltiple en la mayoría de preguntas permite evitar las dificultades presentes en el uso de teclados.
- La ya mencionada dificultad en el manejo del teclado conlleva cierta inseguridad a la hora de introducir datos. Es por eso que se recomienda que los campos para introducir contraseñas en los formularios no están enmascarados por defecto.
- Los mensajes emergentes pueden desorientar a este tipo de usuarios, es por eso que se recomienda mostrar los mensajes de error en la misma pantalla que se está visualizando. Nuestra aplicación muestra los mensajes de error en la propia pantalla excepto en el caso de intentar enviar un formulario de registro o inicio de sesión vacío.
- La disminución de la capacidad visual puede causar grandes problemas a estos usuarios, sobre todo al intentar leer los textos presentes en la aplicación. Los diferentes manuales recomiendan el empleo de tipografía con un tamaño estándar de 14 puntos o superior. Otra de las recomendaciones es asegurar que existe un fuerte contraste entre el color de los textos y el fondo, facilitando de este modo la lectura.
- El posible deterioro de la memoria y la capacidad de concentración hace recomendable la posibilidad de poder volver a la pantalla de inicio desde cualquier pantalla de la aplicación, ya sea mediante botones o iconos táctiles.
- La incorporación de imágenes en los botones de navegación favorece la experiencia del usuario. Añadir imágenes permite que el usuario pueda interpretar y asociar la imagen con cierta acción o pantalla.

El apéndice B recoge los diferentes *mockups* que se han generado siguiendo estas pautas y recomendaciones.

4.3 Patrón de diseño

La elección de un correcto modelo o patrón de arquitectura *software* permite que un proyecto pueda adecuarse a requisitos de coste, tiempo y número de personas implicadas. Estos patrones, a pesar de ser abstracciones teóricas, nos permiten reconocer las diversas partes de un *software* y cómo estas interactúan entre sí.

4.3.1. Arquitectura por capas

Para la implementación de este proyecto se ha optado por un Patrón de Arquitectura por capas [20]. Este patrón se basa en la separación de la aplicación en distintas capas, cada una de las cuales cumple un rol específico y está encargada de una tarea definida. La estructuración por capas de la aplicación permite cumplir el principio de separación de responsabilidades. De esta manera, cada capa es únicamente responsable de las tareas de las que se ocupa y, aunque presente dependencias con el resto de capas, no se preocupa por cómo estas están implementadas. Esta división de responsabilidades facilita enormemente el testeo de la aplicación, ya que las pruebas de cada capa solo deben comprobar que se cumplan los objetivos específicos de esta. No existe un número determinado de

capas en el que se deban dividir las aplicaciones que usan esta arquitectura. En el caso de este proyecto se ha decidido estructurar la aplicación en las siguientes cuatro capas (Figura 4.2):

- Capa de presentación: Es la capa encargada de la interacción directa con el usuario, esta capa constituye lo que se conoce como interfaz de usuario. Se encarga de la representación visual que percibe el usuario, la navegación y la gestión de eventos e interacciones que el usuario realiza con la aplicación.

- Capa de lógica de negocio: En esta capa subyace la mayor parte de la lógica necesaria en nuestra aplicación. Entre las tareas que cumple esta capa en nuestra aplicación encontramos:

  - determinar cómo se comporta la aplicación ante las distintas interacciones y eventos;

  - actuar como intermediaria entre la capa de presentación y la de acceso a datos, procesando los datos de la capa inferior y adecuandolos al formato necesario para poder ser mostrados en la capa de presentación;

  - gestionar los flujos de trabajo encargados de organizar procesos que involucran la llamada a varios servicios externos

- Capa de acceso a datos: Su principal función es la interacción con el sistema de almacenamiento de datos. Dentro de nuestra aplicación esta capa es la encargada de:

  - establecer la conexión y realizar consultas CRUD (*Creation, Read, Update and Delete*) con la base de datos;

  - traducir las estructuras de datos presentes en nuestra base de datos en objetos del lenguaje de programación que se está usando. Este "mapeo de objetos"permite facilitar el manejo de los datos y simplificar las interacciones con la base de datos.

- Capa de base de datos: Esta capa es externa a nuestra aplicación. Está constituida por nuestra base de datos y su principal función es el almacenamiento persistente de datos. Permite el almacenamiento, recuperación y actualización de los datos necesarios por nuestra aplicación.

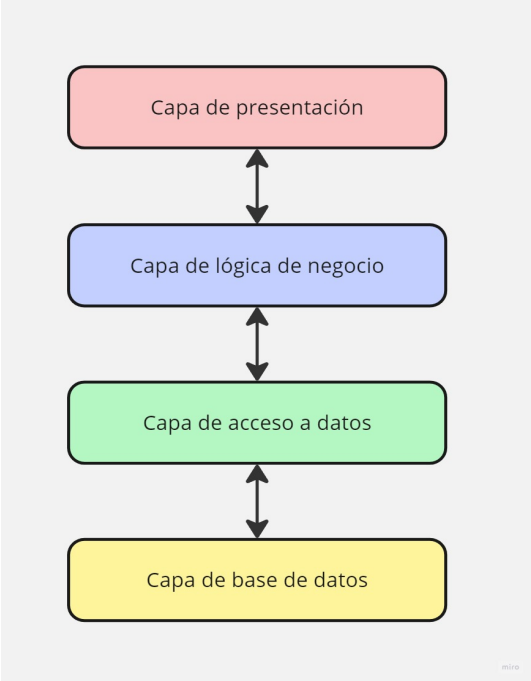


Figura 4.2: Diagrama del patrón de diseño por capas.

Un concepto importante de la arquitectura por capas es que cada capa únicamente debe comunicarse con la capa que está justo por debajo. De esta manera, si una capa que se encuentra en la parte más alta de la arquitectura necesita comunicarse con la capa más baja la comunicación no será directa y el resto de capas actuarán como intermediarias. Esta restricción permite mantener un código mucho más estructurado y definir flujos de trabajo más precisos.

4.4 Tecnologías utilizadas

La naturaleza multiplataforma de esta aplicación junto a la experiencia previa del desarrollador con el *framework* seleccionado, ha motivado la decisión de implementar nuestra aplicación utilizando el *framework React Native* creado por el grupo *Meta*. Este *framework* permite la creación de aplicaciones nativas para *Android* e *iOS* utilizando código compartido escrito en *Javascript*.

4.4.1. Lenguaje de programación JavaScript y TypeScript

JavaScript [21] es uno de los lenguajes de programación más instaurados y demandados en el mundo del desarrollo *software*. Este lenguaje surge con el propósito de ejecutar *scripts* de código en el *frontEnd* de páginas web (navegador del cliente). Su popularización permite que convierta en un lenguaje multipropósito, actualmente *JavaScript* es ejecutable en entornos fuera del navegador, lo cual permite que se utilice en gran variedad de implementaciones como puede ser el desarrollo *backend* de una web (*node.js*) o el desarrollo de aplicaciones móviles.

Construido sobre *JavaScript*, *TypeScript* [22] es un superconjunto de *JavaScript*, es decir, es un lenguaje de programación basado en *JavaScript* que añade características adicionales. Su principal adición es el tipado estático, lo que nos permite definir el tipo de datos de variables, funciones y objetos. El tipado estático permite detectar fallos y errores de tipos antes de que el programa sea ejecutado.

El *framework* utilizado para el desarrollo de nuestra aplicación soporta el uso de estos dos lenguajes. En nuestro caso se ha optado por el desarrollo con *TypeScript* ya que este permite producir código más seguro y fácil de depurar.

4.4.2. React Native

"the biggest mistake that we made as a company is betting too much on HTML5 as opposed to native" - Mark Zuckerberg

En 2012 Mark Zuckerberg, CEO y fundador de *Facebook*, se arrepintió de desarrollar la aplicación de su red social como una *webApp* en lugar de apostar por el desarrollo nativo [23]. No fue hasta más adelante, cuando uno de los integrantes de la empresa descubrió cómo generar elementos de interfaz nativos para *iOS* utilizando código *JavaScript* en segundo plano, cuando *facebook* decidió organizar una *hackathon* para poder crear un *framework* que permitiera el desarrollo de aplicaciones nativas usando código *JavaScript*. Es así como en 2015 surge la primera versión de *React Native* [24].

La principal ventaja que presenta este *framework* es que las aplicaciones creadas con él no son aplicaciones web dentro de un navegador nativo del dispositivo, sino que utilizan componentes nativos de los distintos sistemas operativos. La capacidad para renderizar componentes nativos a partir de código *JavaScript* permite que el código generado pueda ser reutilizado para generar aplicaciones tanto en *iOS* como en *Android* sin la necesidad de codificar la aplicación dos veces en los lenguajes nativos de cada sistema operativo.

Arquitectura de React Native

Para entender mejor cómo este *framework* es capaz de traducir el código de *JavaScript* en componentes nativos revisaremos brevemente su arquitectura [25] así como su componente más llamativo, el *bridge*.

Cuando un usuario ejecuta una aplicación de *React Native* en su dispositivo se crean dos grandes hilos:

- Hilo nativo: Es el hilo principal en el que se ejecuta la aplicación nativa, y el responsable de renderizar los componentes nativos y de captar las interacciones del usuario.

- Hilo JavaScript: Este hilo, que se ejecuta en el motor *JavaScriptCore* del dispositivo, es el encargado de la ejecución del código *JavaScript*. Es el hilo en el que se ejecuta toda la lógica de la aplicación. Junto con el hilo principal de *JavaScript* se genera otro hilo adicional denominado *"shadow Thread"* cuyo propósito es calcular las posiciones de los diversos componentes nativos para que el hilo de *JavaScript* sea consciente de la disposición de estos en la pantalla.

Una vez comienzan a ejecutarse los hilos entra en juego una de las partes más relevantes de esta arquitectura: el componente *bridge*. Este componente es el encargado de permitir la comunicación entre la parte nativa y el código *JavaScript*.

Los lenguajes nativos (*java* y *object-C*) no pueden comunicarse directamente con *JavaScript* ya que son lenguajes diferentes. Es por eso que la comunicación entre ambas capas se da a través de objetos JSON.

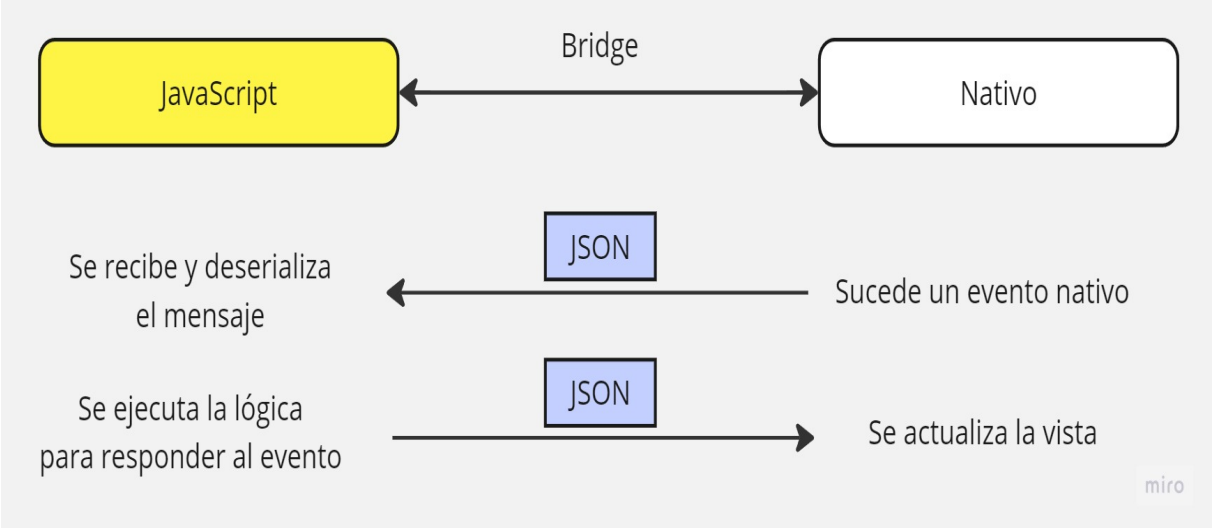


Figura 4.3: Arquitectura de React Native. Flujo de comunicación entre el hilo nativo y el hilo JavaScript.

La figura 4.3 describe cómo se realiza la comunicación y traducción entre el mundo nativo y el mundo *JavaScript*.

- Cuando ocurre un evento nativo (interacción del usuario con los componentes nativos). El hilo nativo envía un mensaje JSON a través del *bridge* con toda la información necesaria para que el hilo de *JavaScript* pueda comprender cómo actuar.
- El hilo *JavaScript* recibe el mensaje, lo deserializa y ejecuta el código con la lógica para responder a ese evento. Cuando el código ya se ha ejecutado, este hilo envía una respuesta en formato JSON al hilo nativo. Este nuevo mensaje, que también se transmite por el *bridge*, indica cómo deben actualizarse los componentes nativos para responder al evento.
- Finalmente, cuando el hilo nativo recibe este último mensaje, actualiza los componentes nativos de la vista para que cuadren con el evento ocurrido.

A pesar del gran avance que supone esta arquitectura, no está exenta de algunos problemas. La comunicación a través del *bridge* es asíncrona; esto, sumado a la gran cantidad de mensajes JSON transmitidos, puede llevar a la formación de cuellos de botella en este canal de comunicación. Este suceso repercute negativamente en el rendimiento de la aplicación. Es por eso que el equipo de *React Native* ha desarrollado una nueva arquitectura que prescinde del uso del *bridge* en favor de lo que se conoce como la JSI *JavaScript Interface*. Esta tecnología permite que el código *JavaScript* contenga referencias directas a objetos nativos y viceversa. Con esta nueva arquitectura un objeto en C++ puede pedir a un objeto *Javascript* que ejecute cierto método, y un objeto de *JavaScript* puede llamar a un método de un objeto C++ para que actualice un componente nativo. De esta manera, muchos de los inconvenientes de la arquitectura basada en el *bridge* desaparecen: ahora la comunicación es síncrona y ya no se forman cuellos de botella.

4.4.3. Expo

Expo [26] es un marco de código abierto que facilita el desarrollo de aplicaciones con *React Native*. Algunas de las herramientas ofrecidas por *Expo* y que han sido utilizadas en este proyectos son:

- *Expo Client*: Esta aplicación permite ejecutar el proyecto que se está desarrollando en nuestro dispositivo móvil sin la necesidad de generar la aplicación nativa.
- *Expo SDK*: Provee acceso a diferentes APIs nativas.
- *Expo CLI*: Interfaz de línea de comandos que permite la ejecución local del proyecto así como generar las aplicaciones nativas (.ipa en *iOS* y .apk en *Android*).

Cabe destacar que el uso del ecosistema *Expo* nos ha forzado a emplear la arquitectura antigua de *React Native*, ya que actualmente *Expo* no incluye soporte a esta nueva arquitectura. Aún así, la migración a esta nueva arquitectura se contempla como uno de los trabajos a futuro más relevantes en este proyecto.

4.4.4. Android Studio Virtual Device Manager

Android Studio es el IDE oficial para el desarrollo de aplicaciones *Android*. Entre sus múltiples herramientas destacamos el VDM *"Virtual Device Manager"* [27], que nos permite crear y configurar emuladores de dispositivos *Android* en nuestro ordenador. De esta manera, podemos probar las aplicaciones en estos dispositivos sin necesidad de disponer de ellos físicamente.

4.4.5. Firebase

Un BaaS *"Backend as a Service"*, como su propio nombre indica, es un servicio que permite externalizar el desarrollo y mantenimiento del *backend* para que sea un tercero el que se ocupe de estas labores [28]. Gracias a las distintas APIs y SDK los desarrolladores son capaces de añadir funciones propias del *backend* (como la gestión de base de datos, autenticación, etc) sin la necesidad de desarrollar un sistema de *backend* específico para su aplicación.

Firebase es un BaaS desarrollado por *Google*. Cuenta con gran variedad de herramientas, pero nosotros destacaremos a continuación las siguientes: *"Firebase Authentication"*, *"Cloud Storage"* y *"Cloud Firestore"*.

Firebase Authentication

Firebase Authentication [29] es una herramienta que proporciona servicios de autenticación seguros sin la necesidad de crear todo un sistema de autenticación desde cero. Esta herramienta permite la autenticación por medio de contraseña, teléfono móvil y otros proveedores de servicio como *Google*, *Facebook* o *Twitter*.

Cloud Storage

Cloud Storage [30] es una herramienta de almacenamiento de objetos. Permite el almacenamiento de imágenes, videos, audios y muchos otros archivos. Su principal propósito es el almacenamiento de contenido generado por el usuario. En este proyecto se ha utilizado *Cloud Storage* para almacenar todas las imágenes relacionadas con las preguntas de los tests.

Cloud Firestore

Cloud Firestore [31] es el servicio de base de datos que proporciona *Firebase*. Cabe destacar que, *Cloud Firestore* es una base de datos *NoSQL*. Esto dota a la base de datos de

una mayor flexibilidad a la hora de establecer el modelo de datos y de una gran capacidad para escalar, ya que estas pueden aumentar su volumen de datos añadiendo nuevos nodos o servidores.

Además, a pesar de ser una base de datos no relacional y de no contar con un lenguaje de consultas, como es SQL, *Firestore* permite la creación de consultas complejas que facilitan la recuperación de datos de manera más sencilla.

4.4.6. Visual Studio Code

Visual Studio Code [32] es un IDE desarrollado por *Microsoft*. Este editor de código cuenta con numerosas extensiones que facilitan el desarrollo. Se ha seleccionado este IDE debido a su familiaridad para el desarrollador y a la gran comunidad de desarrolladores en *TypeScript* que utilizan esta herramienta.

4.4.7. EmailJS

EmailJs [33] es un servicio que permite el envío de correos electrónicos utilizando tecnologías del lado del cliente, lo que significa que no es necesario un servidor *backend* que realice el envío de correos electrónicos. Este proyecto emplea el SDK de *EmailJS* para el envío automático de correos cuando se activa el filtro de inactividad o cuando el usuario quiere generar un informe con las estadísticas mensuales.

4.5 Contextualización de las tecnologías en capas

Después de exponer las tecnologías y la arquitectura de software que presentará nuestro proyecto, resulta un ejercicio sumamente interesante el poder localizar las tecnologías utilizadas dentro de las diversas capas presentes en la aplicación.

React Native es la tecnología principal utilizada en nuestro proyecto. Este *framework* constituye la columna vertebral de la aplicación. Las capas de presentación, lógica de negocio y acceso a datos utilizan esta tecnología, ya que todas ellas se enmarcan dentro de nuestro proyecto en *React Native*.

Del mismo modo, *TypeScript* está presente en estas mismas capas, ya que constituye el lenguaje de programación utilizado durante todo el desarrollo.

La capa de base de datos es la única que se localiza fuera de nuestro proyecto y, por lo tanto, queda fuera del dominio de *TypeScript* y *React Native*. Esta capa hace uso de *Firestore* para permitirnos implementar nuestra base de datos.

Finalmente cabe mencionar que el resto de servicios externos (*EmailJS, Firebase Authentication* y *Cloud Storage*) son difíciles de contextualizar dentro de esta arquitectura y no se considera que se les pueda localizar dentro de una capa determinada, por lo que no se ven representados en el diagrama de la Figura 4.4 .

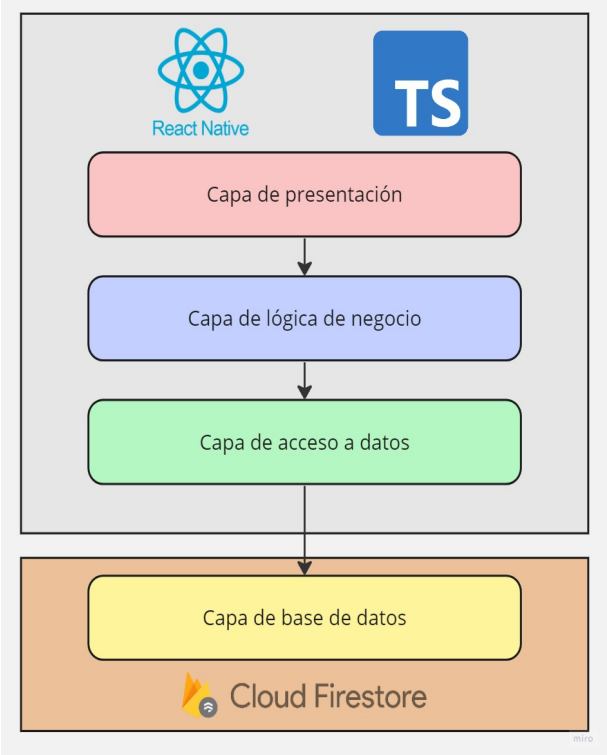


Figura 4.4: Localización de las diversas tecnologías dentro de las capas de la aplicación.

CAPÍTULO 5
Desarrollo de la solución

5.1 Estructura de carpetas

Antes de entrar en detalles sobre la implementación de cada capa de la arquitectura, vamos a hacer un repaso rápido a la estructura del proyecto (Figura 5.1). Revisar las diversas carpetas y ficheros principales de la aplicación nos permite tener un mejor contexto de cómo se ha estructurado el desarrollo del proyecto.

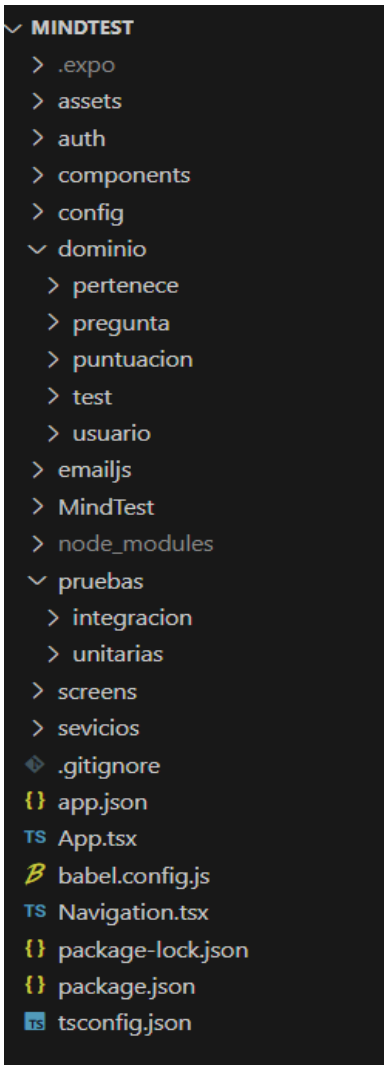


Figura 5.1: Directorio del proyecto.

- .expo: contiene los ficheros de configuración de la herramienta *Expo*.
- assets: contiene las imágenes usadas por la aplicación. Se debe mencionar que las imágenes relacionadas con los test no se almacenan aquí, sino que son almacenadas en el servicio *Cloud Storage* de *Firebase*.
- auth: incluye el código que gestiona el uso de *Firebase Authentication*, un servicio utilizado para la autenticación y registro de usuarios.
- components: contiene los distintos componentes reutilizables que se utilizan en las diferentes pantallas de la aplicación.
- config: guarda el archivo con la configuración necesaria para que la aplicación pueda hacer uso de los servicios de *Firebase*.
- dominio: posee una subcarpeta por cada colección presente en nuestra base de datos. Cada subcarpeta posee la clase entidad y el repositorio de su respectiva colección.
  - domino/pertenece: posee el repositorio y la clase entidad de la colección Pertenece.
  - domino/pregunta: incluye el repositorio y la clase entidad de la colección Pregunta.
  - domino/puntuación: contiene el repositorio y la clase entidad de la colección Puntuación.
  - domino/test: posee el repositorio y la clase entidad de la colección Test.
  - domino/usuario: incluye el repositorio y la clase entidad de la colección Usuario.
- emailjs: contiene el código que gestiona el uso de *EmailJs*, servicio dedicado al envió automático de correos electrónicos.
- node_modules: contiene todas las dependencias necesarias para nuestro proyecto. Todas las librerías y módulos externos que se utilicen se alojarán en esta carpeta.
- pruebas: contiene el código de los test creados para el testeo de la aplicación.
  - integración: aquí se ubica el código de las pruebas de integración.
  - unitarias: contiene el código de las pruebas unitarias.
- screens: contiene el código de las distintas pantallas de la aplicación.
- servicios: almacena los servicios, clases encargadas de la lógica de negocios.
- .gitignore: fichero que permite configurar qué ficheros serán incluidos o excluidos en el control de versiones de *git*.
- app.json: contiene configuraciones de nuestro proyecto: nombre,versión, plataformas soportadas, etc.
- App.tsx: fichero con el código que se muestra cuando se ejecuta la aplicación. App.tsx contiene la primera vista que verá el usuario cuando se inicie la aplicación.
- Navigation.tsx: código que configura la navegación entre las distintas pantallas de la aplicación.

- package-lock.json y package.json: contiene información sobre todas las dependencias que se usan en el proyecto. Estos archivos permiten que, cuando el proyecto se clone en otro equipo, todas las dependencias puedan ser instaladas correctamente con las versiones adecuadas.

- tsconfig.json: fichero de configuración de *TypeScript*.

5.2 Implementación de la capa de presentación

La capa de presentación es la encargada de la interfaz de usuario, así como de las interacciones directas con el usuario. Esta capa ha sido implementada por medio de las diversas pantallas, las cuales, a su vez, están formadas por un conjunto de componentes.

5.2.1. Componentes

Los componentes constituyen los diversos elementos presentes en la interfaz de nuestra aplicación. En *React Native* los componentes están conformados por uno o varios elementos JSX (elementos similares a los generados con etiquetas *html*). *React Native* nos permite generar componentes personalizados para que estos puedan adaptarse a las necesidades de nuestras interfaces y puedan ser reutilizados en el código de distintas pantallas.

Props

Cuando creamos un componente en *React Native*, procuramos generar código que pueda ser reutilizable en distintas partes de nuestra aplicación. Es por eso que, en ocasiones, surge la necesidad de pasar cierta información a nuestros componentes para que estos puedan adaptarse a la interfaz específica en la que son mostrados. Es aquí donde entran en juego las *props* [34], un mecanismo que permite pasar información de manera unidireccional desde un componente padre (pantalla) a los componentes hijos, para que así estos se puedan personalizar dependiendo de las necesidades de cada situación. Por ejemplo, si se crea un componente botón y se desea añadir dos botones a una pantalla con colores de fondo distintos, no es necesario implementar dos componentes. Simplemente se puede pasar el color de fondo como una de las *props* de ese componente.

Las *props* de *React Native* facilitan enormemente la reusabilidad de código y son una herramienta fundamental en todo desarrollo con esta tecnología.

Carpeta components

En nuestro proyecto, todos los componentes utilizados se localizan en la carpeta components (Figura 5.2). A continuación, se explicará brevemente cada uno de ellos.

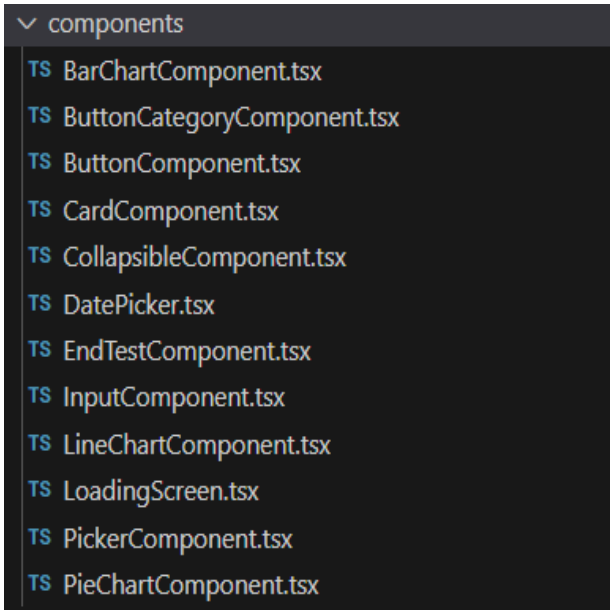


Figura 5.2: Carpeta con los componentes de la capa de presentación.

- BarChartComponent.tsx: Gráfico de barras utilizado en la pantalla de estadísticas para mostrar la media de puntuaciones de cada mes.
- ButtonCategoryComponent.tsx: Componente formado por varios botones. Es el componente utilizado en la pantalla principal para seleccionar la categoría de la que se quiere realizar el test.
- ButtonComponent.tsx: Botón utilizado en todas las pantallas de la aplicacion.
- CardComponent.tsx: Componente táctil que permite la navegación entre la pantalla principal y las pantallas de estadísticas e información.
- CollapsibleComponent.tsx: Desplegable que permite el pliegue y despliegue de un cuadro de texto cuando el usuario interactúa con el.
- DatePicker.tsx: Calendario que permite seleccionar una fecha específica.
- EndTestComponent.tsx: Componente formado por un texto y un botón. Es el encargado de mostrar la puntuación cuando se finaliza un test y de permitir la navegación de vuelta a la pantalla principal.
- InputComponent.tsx: Campo de entrada de texto.
- LineChartComponent.tsx: Gráfico de líneas. Muestra la evolución de las puntuaciones en un determinado lapso de tiempo.
- LoadingScreen.tsx: Icono de carga mostrado mientras se realizan los procesos de recolección de datos.
- PickerComponent.tsx: Componente que permite la selección entre un conjunto de opciones desplegadas.
- PieChartComponent.tsx: Gráfico circular, utilizado para mostrar el porcentaje de test realizados de cada categoría.

5.2.2. Pantallas

Todas las vistas de la aplicación se encuentran dentro de la carpeta screens (Figura 5.3). Dentro de este directorio encontramos las siguientes pantallas.

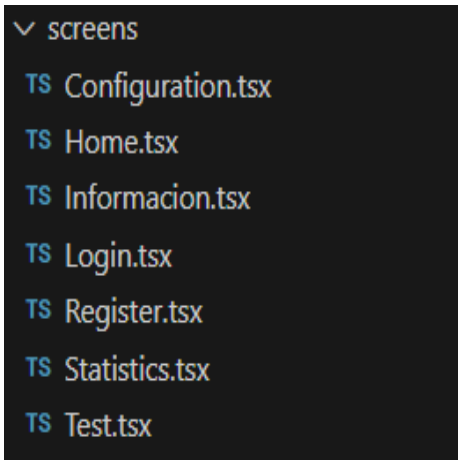


Figura 5.3: Carpeta con las pantallas de la capa de presentación.

- Configuration.tsx: Pantalla de configuración, en la que se pueden cambiar el correo electrónico de contacto, activar el filtro de inactividad y generar los informes mensuales.
- Home.tsx: Pantalla principal, desde la que se puede acceder a la mayoría de pantallas y comenzar un test de la categoría seleccionada.
- Informacion.tsx: Pantalla con información y contactos relevante sobre las enfermedades neurodegenerativas.
- Login: Es la primera pantalla al iniciar la aplicación. Permite a los usuarios autenticarse con sus credenciales.
- Register.tsx: Pantalla de registro; permite a los usuarios que no disponen de cuenta registrarse en la aplicación.
- Statistics.tsx: Pantalla de estadísticas; muestra diversas gráficas con las estadísticas del usuario.
- Test.tsx: Pantalla de los test; muestra las diversas preguntas que conforman el test adaptándose al formato de estas. Cuando el test concluye muestra la puntuación y permite volver a la pantalla principal.

5.2.3. React Navigation

En *React Native* la navegación entre pantallas se realiza gracias a los componentes que nos proporciona la librería *React Navigation* [35], una librería extremadamente popular que facilita las operaciones de navegación entre pantallas. *React Navigation* nos ha permitido configurar una navegación declarativa, en la cual hemos podido establecer nuestros propios flujos de navegación y decidir qué pantallas se muestran en función de las interacciones que realice el usuario.

5.3 Implementación de la capa de lógica de negocio

La capa de lógica de negocio es una de las partes más importantes de la aplicación. En esta se determina qué acciones debe realizar la aplicación para reaccionar a los distintos eventos de la capa de presentación. Entre otras tareas, esta capa también es la encargada de dar un formato correcto a los datos provenientes de la capa de acceso a datos para que estos puedan ser mostrados por la capa de presentación.

En nuestra aplicación, la lógica de negocio ha sido implementada y encapsulada en las clases de servicios. Cada una de estas clases posee la lógica necesaria en las funciones que desempeña nuestra aplicación. La Figura 5.4 muestra el directorio con las distintas clases de servicio que posee la aplicación.

Figura 5.4: Carpeta con los servicios de la capa de lógica de negocio.

A continuación pasaremos a explicar brevemente qué operaciones llevan a cabo cada uno de estos servicios.

5.3.1. EmailService

Gestiona el envío automático de correos electrónicos. Este servicio desempeña dos grandes funciones:

- Comprueba si la última puntuación guardada en la base de datos es anterior al filtro de inactividad activo. En el caso de que esta condición se cumpla, llama al repositorio encargado de utilizar el servicio de envío de correos *EmailJS*.
- Realiza la llamada al repositorio encargado de generar el envío de un correo con las estadísticas del mes que el usuario ha seleccionado en la pantalla de configuración.

5.3.2. EstadisticasService

Su principal función es adaptar el formato de los datos que le llegan de la capa de presentación para poder mostrarlos en la pantalla de estadísticas. Se encarga de:

- Estructurar los datos de la manera necesaria para mostrarlos en el gráfico de barras, gráfico de líneas y gráfico circular.
- Adaptar los datos al formato necesario para que el EmailService pueda enviar el correo con el informe de estadísticas.

5.3.3. TestService

Servicio encargado de la lógica subyacente a los test. Sus funciones son:

- Dada una categoría, selecciona al azar uno de los test de esa categoría; además, recoge y da el formato correcto a las preguntas del test seleccionado para que estas se muestren en pantalla.
- Cuando un test finaliza, se encarga de llamar a la capa de acceso a datos para que esta guarde la puntuación del test.

5.3.4. UsuarioService

Contiene la lógica asociada a los usuarios. Desempeña las funciones de:

- Gestionar los procesos de autenticación:
  - Comprueba la validez de los datos introducidos por el usuario en los procesos de inicio de sesión y registro.
  - Realiza las llamadas necesarias al repositorio encargado de gestionar el servicio *Firebase Authentication*.
  - Comunica a la capa de acceso a datos la necesidad de crear un nuevo usuario en la base de datos cuando este es registrado correctamente.
- Permite actualizar los campos de correo electrónico de contacto y filtro de inactividad del usuario.

5.4 Implementación de la capa de acceso a datos

Esta capa es la encargada de la comunicación con el sistema de almacenamiento de datos. En nuestra aplicación, la capa de acceso a datos se estructura en cinco directorios, uno por cada colección presente en nuestra base de datos (tablas en la Figura 3.1).

Cada uno de estos directorios contiene una clase entidad, una clase repositorio y la interfaz de esta última (Figura 5.5).

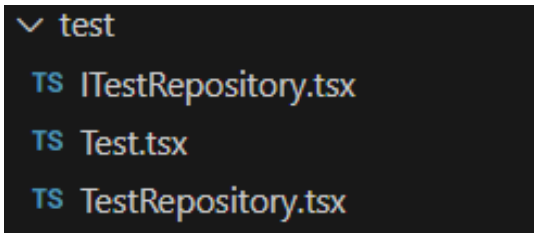


Figura 5.5: Ejemplo de uno de los directorios de la capa de acceso a datos.

5.4.1. Entidades y mapeo de objetos

El mapeo de objetos es un proceso que consiste en traducir las estructuras de datos presentes en nuestra base de datos a objetos del lenguaje de programación que se está utilizando. A pesar de ser una práctica utilizada en bases de datos relacionales, el mapeo de objetos también puede resultar útil en nuestro contexto, ya que este permite una mayor facilidad a la hora de operar con estos datos y consigue detectar errores más fácilmente.

Las entidades son clases de *TypeScript* que permiten traducir los datos de *Firestore* en objetos de *Typescript*, para así facilitar el acceso o actualización de sus atributos y facilitar la depuración de código.

5.4.2. Repositorios

Los repositorios son las clases encargadas de realizar las consultas a la base de datos. Cada clase repositorio es la encargada de realizar las operaciones CRUD sobre su respectiva colección en *Firestore*. Las distintas consultas se encapsulan dentro de los distintos métodos de la clase, de manera que, cada método es el encargado de una única operación CRUD. Estos métodos también se encargan de mapear los datos recibidos y devolver los datos transformados en la entidad correspondiente. El diagrama de la figura 5.6 ejemplifica los pasos que realiza una clase repositorio cada vez que se llama a uno de sus métodos.

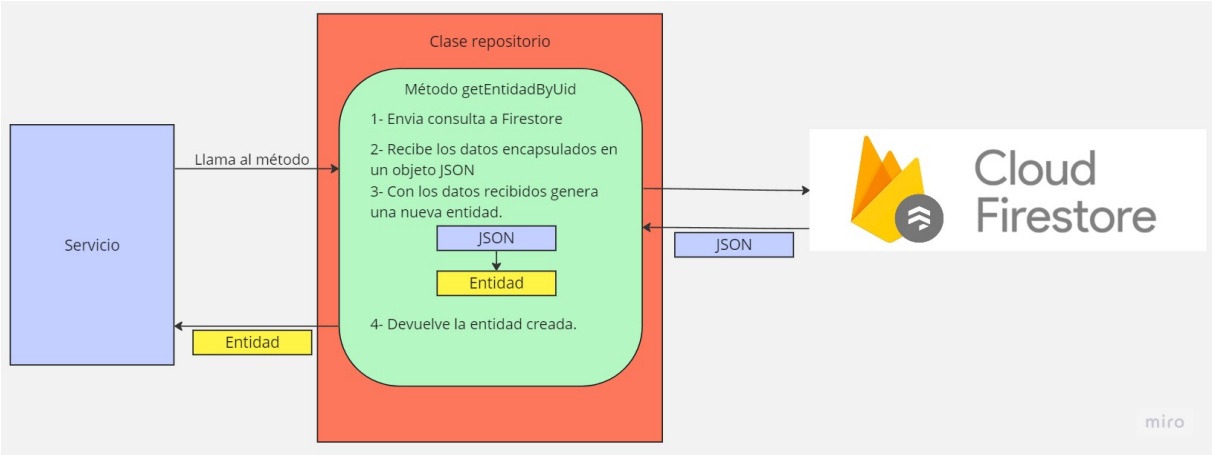


Figura 5.6: Diagrama de las funciones que desempeñan las clases repositorio.

5.5 Implementación de la capa de base de datos

Como ya se ha mencionado en el apartado 4.4.5, se ha optado por utilizar los servicios de *Cloud Firestore* como base de datos.

5.5.1. Colecciones y documentos

Firestore es una base de datos no relacional, lo que le lleva a sustituir los conceptos de tablas y registro por los de colecciones y documentos [36].

Las colecciones son agrupaciones de documentos que están relacionados de alguna manera. En nuestro caso, cada una de las tablas del diagrama UML expuesto en la figura 3.1 se corresponde con una colección en *Firestore*.

Los documentos son unidades de almacenamiento de datos. Dentro de cada documento se pueden declarar datos con el formato clave-valor. A pesar de poder seguir una cierta estructura, el modelo no relacional permite que cada documento tenga sus propias parejas clave-valor, incluso si estos pertenecen a la misma colección. Los documentos poseen un identificador único que permite diferenciarlos. En nuestra aplicación, cada documento correspondería a un registro de una tabla en el modelo relacional.

La Figura 5.7 muestra un ejemplo de organización de una colección y sus documentos en *Firestore*.

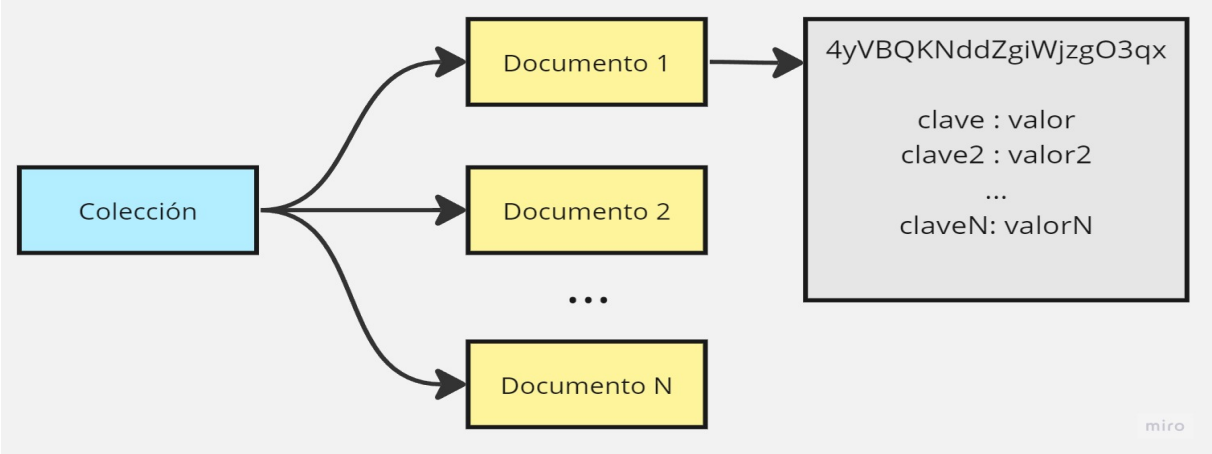


Figura 5.7: Ejemplo de colección y sus documentos en Firestore.

Cabe destacar que todas las colecciones y documentos han sido creados manualmente utilizando la herramienta web que proporciona *Google Firebase* en su página oficial (Figura 5.8).

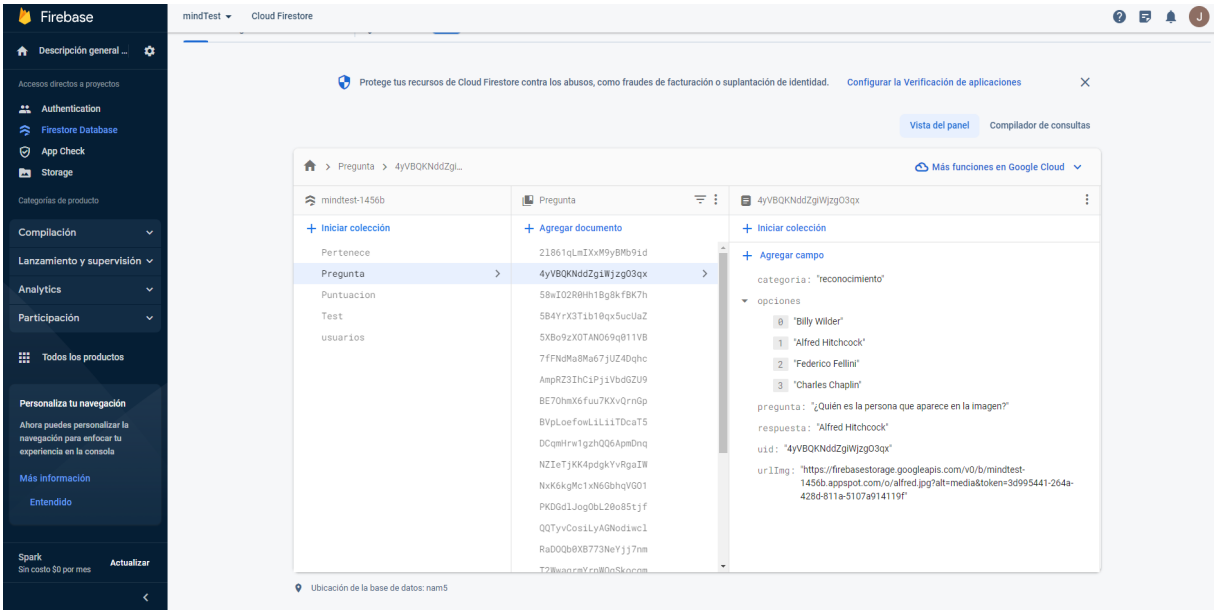


Figura 5.8: Herramienta para la creación de colecciones y documentos en la web oficial de Google Firebase.

5.6 Otros servicios utilizados

Ya se ha abordado la implementación de las distintas capas de nuestra aplicación. Sin embargo, hay servicios externos, como pueden ser *emailJs*, *Firebase Authentication* o *Cloud Storage*, que no pueden incluirse en una determinada capa de la arquitectura ya que su uso e implementación conlleva pasos que están fuera de los propósitos de cada una de las capas. Es por eso que, en esta sección, trataremos de explicar cómo se han implementado estos servicios dentro de nuestra aplicación.

5.6.1. Firebase Authentication y Cloud Storage

Además de sus servicios de almacenamiento de datos, el Baas de *Google Firebase*, contiene muchas más herramientas que permiten gestionar las funciones necesarias en el *backend* de nuestra aplicación. *Firebase* pone a nuestra disposición un SDK que permite el acceso a todos sus servicios desde la aplicación que estemos desarrollando.

Firebase Authentication

Firebase Authentication nos proporciona herramientas para facilitar los procesos de autenticación de usuarios.

En nuestra aplicación, la comunicación con esta herramienta se realiza mediante la clase repositorio AuthRepository. Este repositorio, a diferencia de los de la capa de acceso a datos, únicamente contiene las llamadas necesarias para el registro e inicio de sesión con *Firebase Authentication*. AuthRepository no está asociado a ninguna de las colecciones presentes en nuestra base de datos y, por tanto, no pertenece a la capa de acceso a datos.

Otro detalle a mencionar es que, a pesar de estar presente en el diagrama de la Figura 3.1, el uso de este servicio hace innecesario el almacenamiento del campo contraseña en nuestra colección de usuarios. Por lo tanto, aunque aparezca en la figura, este campo no ha sido añadido en nuestra base de datos.

Cloud Storage

Cloud Storage permite almacenar contenido multimedia generado por usuarios. En nuestro caso, este servicio ha sido utilizado de manera más modesta para almacenar las imágenes de nuestras preguntas.

Hemos optado por almacenar las imágenes en *Cloud Storage* en lugar de ubicarlas en la carpeta assets del proyecto. Esta decisión permite que todas las imágenes de las preguntas puedan ser actualizadas sin la necesidad de una actualización del proyecto. Tener las imágenes almacenadas en la nube de *Cloud Storage* permite desvincularlas de nuestra aplicación y que su gestión y actualización no obligue a los usuarios a descargar actualizaciones.

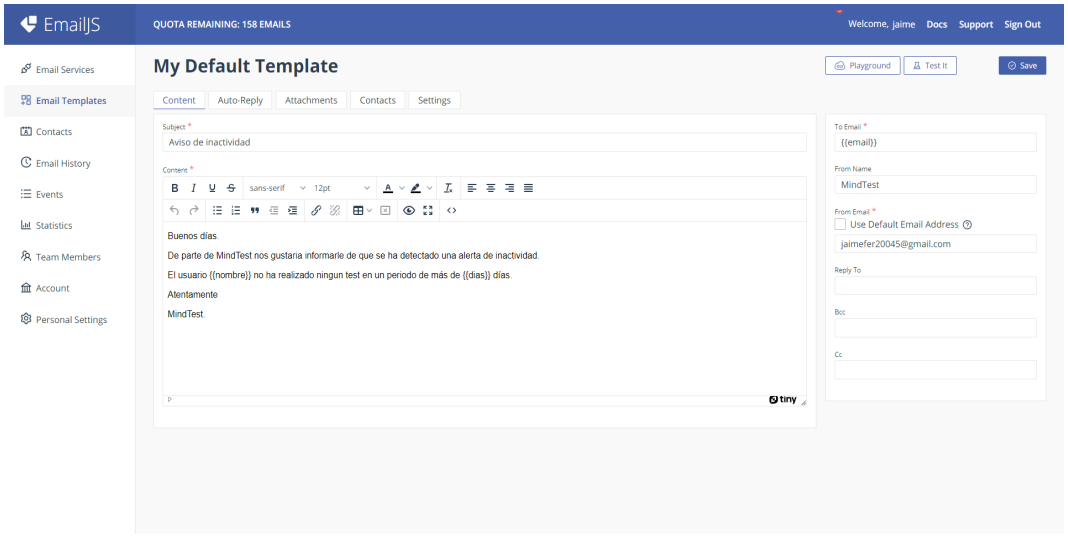
Para acceder a estas imágenes se ha añadido el campo imgUrl dentro de la colección de Preguntas (se puede ver en la Figura 5.8). Este campo almacena la url de acceso a la imagen asociada con la pregunta.

5.6.2. EmailJS

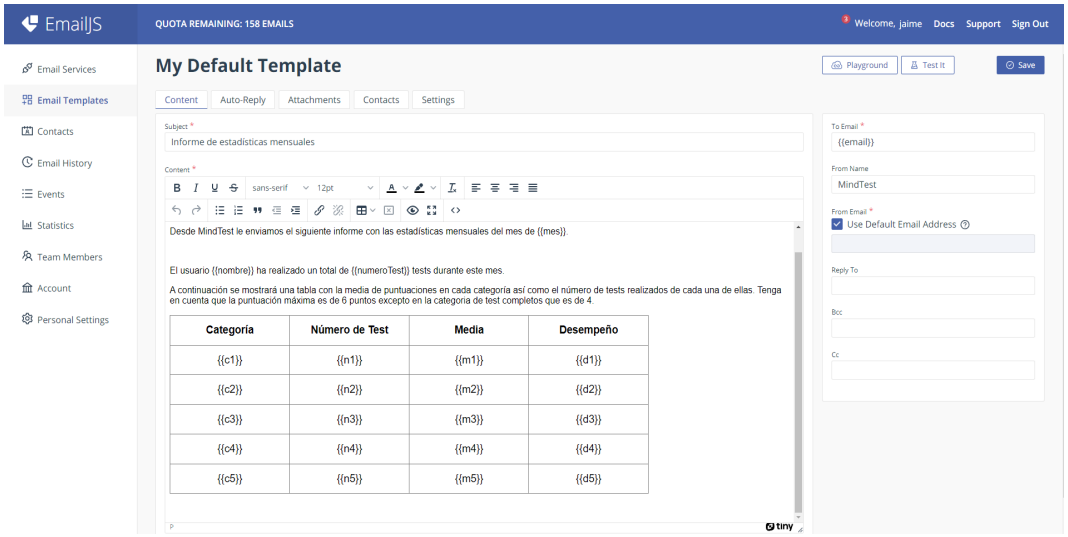
Emailjs es el servicio empleado para la automatización del envío de correos electrónicos. Para hacer uso de este servicio hemos necesitado realizar los siguientes pasos:

- En primer lugar, desde el apartado de servicios en nuestra cuenta de *EmailJS* hemos creado un nuevo servicio de envío de correos via *Gmail*.

- El siguiente paso es crear un *template*. Los *templates* definen la estructura del correo que se va a enviar. En nuestra aplicación, se han definido dos *templates* uno para el aviso de inactividad (Figura 5.9a) y otro para los informes de puntuaciones (Figura 5.9b).



(a) Template de aviso de inactividad en EmailJS.



(b) Template de informe de estadísticas en EmailJS.

Figura 5.9: Templates de Emails

Al igual que con *Firebase Authentication* todas las llamadas al servicio *EmailJS* se realizan a través de un repositorio emailJSRepository. La comunicación con este servicio se realiza gracias al SDK que *EmailS* pone a disposición de los desarrolladores.

Debe mencionarse que todos los campos dentro de llaves en las *templates* representan datos que se deben pasar en las llamadas del SDK. Esto nos permite personalizar nuestros correos. Por ejemplo, cada vez que se envía un informe, el método del repositorio encargado de comunicarse con *EmailJs* le envía como parámetro todas las estadísticas necesarias para rellenar el *template*.

5.7 Flujo de trabajo y comunicación entre capas

Una vez definido cómo se han implementado las diversas capas de la arquitectura, resulta necesario ejemplificar cómo estas se comunican y responden de manera conjunta ante los eventos del usuario El diagrama de la Figura 5.10 muestra cómo se produce esa comunicación.

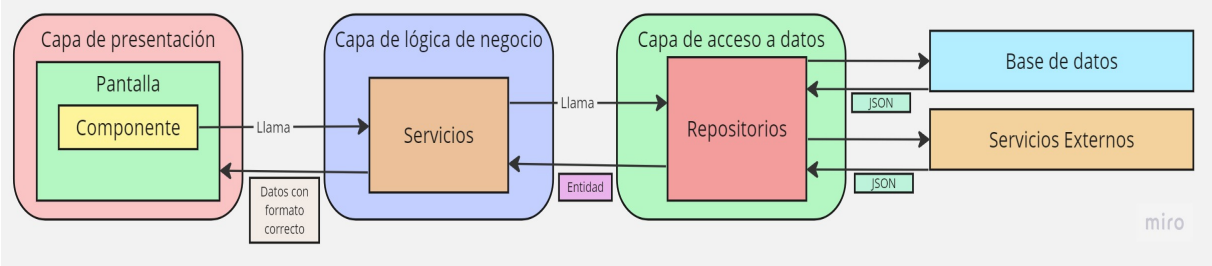


Figura 5.10: Diagrama de comunicación entre capas.

- Cuando un usuario interactúa con un componente de la capa de presentación este genera una reacción al evento.

- Dicha reacción es, en la mayor parte de casos, una llamada a alguno de los servicios de la capa de lógica de negocio.

- Cuando el servicio recibe la llamada, realiza la lógica necesaria que en la mayor parte de casos pasa por utilizar alguno de los métodos de las clases repositorio.

- Las clases repositorio son las que finalmente se comunican con los servicios externos necesarios para realizar la operación que se desea. Estas clases esperan hasta recibir una respuesta JSON de los servicios externos que les indique si la operación se ha realizado correctamente.

- Cuando el repositorio recibe la respuesta la devuelve a la clase servicio (muchas veces mapeada en un objeto entidad) y, posteriormente, este servicio adapta los datos recibidos al formato necesario para que la capa de presentación muestre los cambios al usuario.

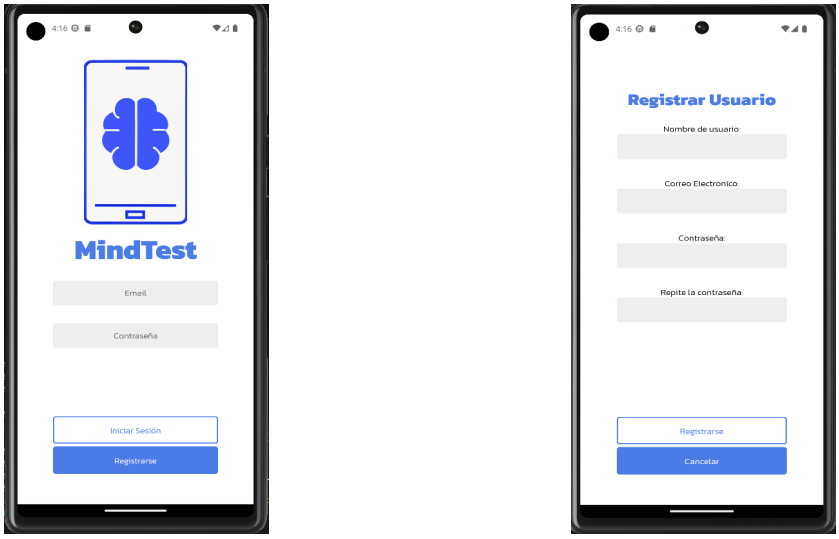
En el caso de que ocurra un error en cualquiera de las capas, este es propagado entre las capas hasta llegar a la capa de presentación. Es en esta capa, donde el error se maneja y la interfaz muestra la información correspondiente al usuario.

5.8 Resultado final

En esta sección se exhibirá el producto final con el objetivo de poder valorar cual es el resultado de todos los procesos descritos hasta este punto. Las diferentes imágenes son capturas extraídas desde un emulador *Android* en el que se está ejecutando nuestra aplicación. Este apartado describe cómo es una sesión de uso normal de la aplicación *MindTest*.

Al iniciar la aplicación se nos muestra una pantalla que permite iniciar sesión (Figura 5.11a). Si el usuario no posee una cuenta en *MindTest* puede pulsar en el botón de registrarse para navegar hasta la pantalla de la Figura 5.11b.

En esta nueva pantalla, el usuario puede introducir todos los datos necesarios para crear una nueva cuenta. También se incluye un botón *"Cancelar"* para que el usuario pueda volver a la pantalla de inicio de sesión si así lo desea.



(a) Pantalla de inicio de sesión. **(b)** Pantalla de registro.

Figura 5.11: Pantallas finales de inicio de sesión y registro.

Una vez realizados los procesos de autenticación se muestra la pantalla principal (Figura 5.12). Esta contiene dos grandes botones para navegar a las pantallas de información y estadísticas. Los dos botones inferiores permiten ir a la pantalla de configuraciones y cerrar sesión. Además, desde esta pantalla también se puede iniciar un nuevo test y seleccionar su categoría. Cuando el usuario pulse en el botón *"Realizar Test"* navegará hasta la pantalla de test.

Figura 5.12: Pantalla principal.

La pantalla de test es la encargada de mostrar las diferentes preguntas y opciones de respuesta (en el caso de ser una pregunta de opción múltiple). En este caso el usuario ha seleccionado un test completo. Las figuras 5.13, 5.14 y 5.15a muestran cada una de las preguntas de cada categoría. Cuando el usuario finaliza el test se muestra su puntuación junto con un botón para volver a la pantalla principal (Figura 5.15b).

(a) Pantalla de test con pregunta de la categoría de reconocimiento.

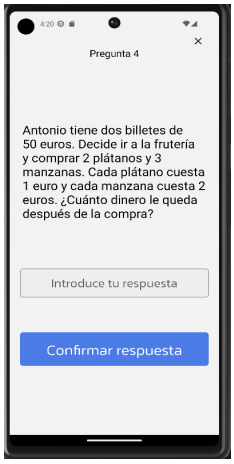
(b) Pantalla de test con pregunta de la categoría de lenguaje.

Figura 5.13: Pantallas con preguntas de las categorías reconocimiento y lenguaje.

(a) Pantalla que muestra las palabras a memorizar en la pregunta de memoria.

(b) Pantalla de test con pregunta de la categoría de memoria.

Figura 5.14: Pantallas con pregunta de la categoría memoria.



(a) Pantallas con pregunta de la categoría cálculo.

(b) Pantalla fin del test.

Figura 5.15: Pantalla con pregunta de la categoría cálculo y pantalla de fin del test.

Una vez de vuelta en la pantalla principal, el usuario navega hasta la pantalla de estadísticas (Figuras 5.16a y 5.16b). Esta contiene tres gráficas; una circular donde el usuario puede ver cuantos test ha realizado de cada categoría, otra gráfica de líneas donde puede consultar sus puntuaciones, y una gráfica de barras con la media de puntuaciones de cada mes. Cuando el usuario termina de repasar los datos vuelve de nuevo a la pantalla principal.

(a) Pantalla de estadísticas parte superior .

(b) Pantalla de estadísticas parte inferior.

Figura 5.16: Pantalla de estadísticas.

De nuevo en la pantalla principal, el usuario navega hasta la pantalla de información (Figura 5.17). En esta pantalla el usuario puede revisar información interesante sobre las enfermedades neurodegenerativas y sus síntomas.

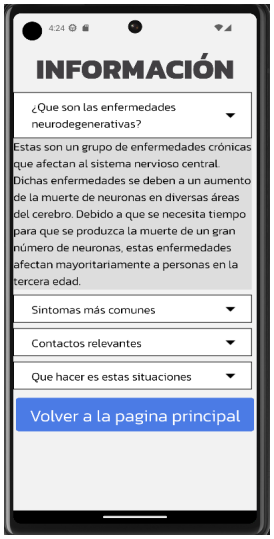
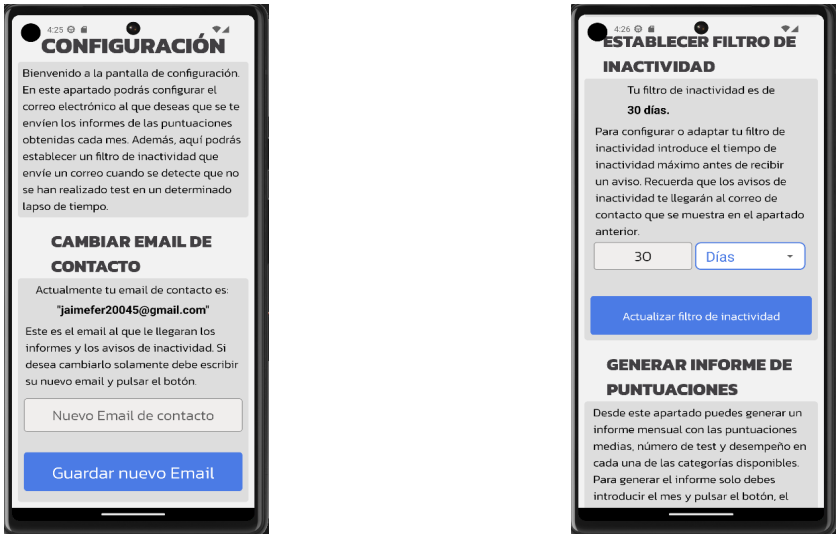


Figura 5.17: Pantalla de información.

Por último, el usuario vuelve a la pantalla principal y navega hasta la pantalla de configuración. Una vez en esta, el usuario puede cambiar el correo electrónico de contacto (Figura 5.18a), establecer un nuevo filtro de inactividad que en este caso es de 30 días (Figura 5.18b) y generar un informe con las puntuaciones, que en este ejemplo es el informe del mes de agosto (Figura 5.19).



(a) Pantalla de configuración, opciones para cambiar el correo electrónico de contacto.

(b) Pantalla de configuración, opciones para establecer un filtro de inactividad.

Figura 5.18: Pantalla de configuración, opciones para cambiar el correo electrónico de contacto y para establecer un filtro de inactividad.

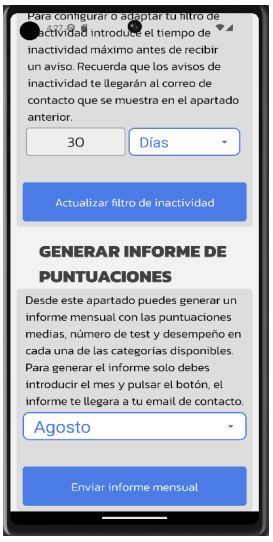


Figura 5.19: Pantalla de configuración, opciones para generar el informe de puntuaciones.

CAPÍTULO 6
Pruebas

El testeo de nuestro *software* es un proceso imprescindible para poder generar código de calidad que pueda ser llevado a producción. La creación de pruebas que aborden las distintas casuísticas a las que se puede ver sometido nuestro código nos permiten detectar errores que, de otra manera, pasarían completamente inadvertidos.

Dentro de la gran variedad de pruebas existentes en el mundo del desarrollo *software*, nuestro proyecto ha sido sometido a dos tipos de testeo en concreto: pruebas unitarias y pruebas de integración.

6.1 Herramientas de testeo Jest y React Native Testing Library

Existe una amplia gama de herramientas disponibles para el testeo de aplicaciones desarrolladas con *JavaScript* y *Typescript*. De entre todas ellas, se ha decidido seleccionar *Jest* y *React Native Testing Library* debido a su gran popularidad dentro del mundo del desarrollo de aplicaciones con *React Native*.

Jest [37] es un *framework* de pruebas diseñado para realizar pruebas unitarias y pruebas de integración en proyectos desarrollados con *JavaScript*. *Jest* contiene una gran variedad de herramientas entre ellas podemos destacar la capacidad para realizar *mocks* de objetos o funciones y la capacidad para medir la cantidad de código que ha sido testeado (cobertura).

Por otro lado, la librería *React Native Testing Library* [38] permite facilitar la creación de pruebas enfocadas a evaluar cómo reaccionan nuestros componentes ante las interacciones de los usuarios.

6.2 Pruebas unitarias

Las pruebas unitarias se definen como aquellas encargadas de aislar y evaluar el perfecto funcionamiento de unidades aisladas de código. Estas pruebas nos permiten validar que cada unidad de código cumpla correctamente con sus tareas asignadas de forma independiente al resto de partes del proyecto.

En nuestra aplicación hemos generado pruebas unitarias que comprueban el correcto funcionamiento de nuestros componentes (Figura 6.1). Las pruebas se han centrado en:

- Testear el renderizado de componentes: La librería *React Native Testing Library* permite acceder a los distintos elementos JSX que conforman nuestros componentes y comprobar que todos estos se renderizan correctamente.

- Testear el comportamiento de los componentes ante los diversos eventos: *React Native Testing Library* también permite la simulación de diversos eventos que pueden ser activados por los usuarios (como es el caso de *press*, *touch*, etc). Esta capacidad nos ha permitido comprobar si el comportamiento de nuestro componente ante esos eventos es el adecuado.

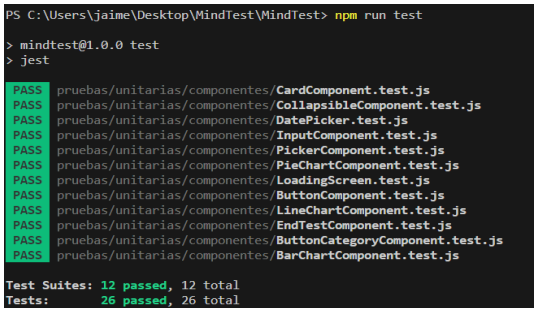


Figura 6.1: Ejecución de las pruebas unitarias de los diversos componentes.

6.3 Pruebas de integración

A diferencia de las unitarias, las pruebas de interacción tienen como objetivo evaluar la interacción entre las distintas partes o componentes que constituyen un proyecto de *software*. Estas nos permiten detectar fallos cuando varios componentes individuales trabajan en conjunto para realizar acciones más complejas.

Hemos creado pruebas de integración para comprobar la correcta interacción de los componentes que conforman las diversas pantallas. Estas pruebas nos han permitido comprobar que:

- Las pantallas renderizan todos los componentes que las conforman.

- Los eventos que se activan en alguno de los componentes son capaces de modificar el estado de las pantallas que los contienen.

6.4 Ampliación de la cobertura y trabajos a futuro

La cobertura de testeo es la medida utilizada para reflejar qué porcentaje de código ha sido testeado. Durante el transcurso de este proyecto no se ha establecido un desarrollo siguiendo una metodología TDD (*Test Driven Development*) [39] consistente en realizar primero las pruebas y luego generar y refinar el código necesario para superarlas. No establecer este enfoque ha derivado en una cobertura francamente mejorable. Aún así, en este apartado se pretende poner de manifiesto los próximos pasos que se pueden llevar a cabo en materia de testeo y que constituyen uno de los trabajos a futuro más prioritarios.

- Pruebas unitarias: Actualmente nuestras pruebas unitarias cubren únicamente los componentes de la capa de presentación. Por ello, se pretenden realizar los siguientes trabajos:

  - Generar las pruebas unitarias que permitan testear el funcionamiento de los métodos de nuestras clases servicio.
  - Generar pruebas unitarias que permitan evaluar los métodos de las clases de repositorio. Cabe mencionar que el comportamiento de estas clases está supeditado a las respuestas recibidas por los servicios externos con los que se comunica. Por ello, se considera necesario utilizar procesos de mocking que permitan aislar nuestras pruebas unitarias de dependencias externas (comportamiento de *Firebase* u otros servicios).

- Pruebas de integración: La comprobación de una correcta interacción entre las diversas capas de la aplicación es otro de los ámbitos más prioritarios. Las pruebas de integración presentes solo comprueban la integración de los componentes en las diversas vistas. Es de especial interés la creación de nuevas pruebas que evalúen la comunicación entre capas cuando se realizan tareas complejas.

CAPÍTULO 7
Conclusiones

En este apartado de conclusiones se pretende realizar un ejercicio de retrospectiva y determinar si el trabajo expuesto en esta memoria cumple con los objetivos expuestos en el apartado 1.2. Con este propósito en mente, vamos a realizar un pequeño repaso de las acciones y decisiones tomadas durante el transcurso del proyecto.

En primer lugar, se ha realizado una breve investigación de los diversos síntomas presentes en las enfermedades neurodegenerativas y, a raíz de lo estudiado, se han revisado, seleccionado y adaptado diversas actividades extraídas de un manual dedicado a la estimulación cognitiva de pacientes con Alzheimer. Todos estos procesos nos han permitido generar una batería de preguntas que tiene cierto respaldo en documentación científica.

Por otro lado, se ha utilizado el *framework React Native* para generar un aplicación móvil nativa multiplataforma utilizando código *TypeScript*. La selección de un patrón de diseño por capas nos ha permitido estructurar y dividir nuestro código para facilitar el testeo y la división de responsabilidades. Además la implementación de la interfaz (capa de presentación) por medio de componentes ha permitido la reutilización de estos en diversas pantallas de la aplicación haciendo innecesario la creación de nuevo código para los componentes específicos de cada pantalla.

Hemos seleccionado las tecnologías proporcionadas por *Google Firebase* para implementar un sistema *backend* que nos permita el almacenaje de datos persistentes en una base de datos y una correcta gestión de los procesos de autenticación de usuarios.

A todas estas acciones se les debe sumar el estudio de guías y artículos que nos han permitido establecer pautas para diseñar una interfaz amigable para personas de edad avanzada.

Es por todo lo expuesto que se considera que los objetivos del trabajo se han cumplido.

7.1 Relación del trabajo desarrollado con los estudios cursados

La puesta en práctica de muchos de los conocimientos adquiridos durante el Grado de Ingeniería Informática han sido totalmente indispensables para conseguir cumplir con los objetivos fijados.

Los contenidos estudiados en asignaturas como *"Interfaces Persona Computador"* nos han permitido contar con una base durante la etapa de diseño de nuestra interfaz. Por otro lado, asignaturas como *"Ingeniería del software"* nos han ayudado a poder seleccionar una arquitectura software y entender la necesidad del testeo de nuestra aplicación.

Una de las asignaturas más relevantes para este trabajo ha sido *"Design of Mobile Application"* cursada en la Universidad Politécnica de Milán durante una estancia Erasmus de seis meses. Esto me permitió familiarizarme con las tecnologías utilizadas en el mundo del desarrollo para dispositivos móviles.

Cabe destacar que el alumno que ha realizado este proyecto no pertenece a la rama de ingeniería de *software*; sin embargo, la realización de prácticas externas le ha permitido familiarizarse con los procesos involucrados en el desarrollo de *software*, así como con el uso de metodologías ágiles.

7.1.1. Competencias transversales

Dentro de la amplia lista de competencias transversales que se adquieren al cursar un grado de Ingeniería Informática, queremos destacar aquellas que han sido aplicadas en un mayor grado durante el desarrollo de este proyecto:

- Aplicación y pensamiento práctico: Una de las principales labores durante la elaboración de este trabajo ha sido la de aplicar gran parte de nuestros conocimientos teóricos y ponerlos en práctica dentro del contexto más cercano a los procesos de desarrollo *software* reales.
- Diseño y proyecto: Durante varias etapas del proyecto hemos evaluado las diversas ideas presentes hasta poder definir una propuesta de proyecto con objetivos fijos y alcanzables.
- Aprendizaje permanente: Algunas de las tecnologías empleadas en este trabajo nos eran desconocidas. Esta competencia nos ha facilitado poder aprenderlas de manera autodidacta.
- Análisis y resolución de problemas: Fundamental para la realización de cualquier proyecto *software*. La capacidad para analizar los problemas presentes en cualquiera de las etapas y poder resolverlos de la manera más eficaz es la competencia más necesaria en este tipo de trabajos.

7.2 Trabajos futuros

A pesar de haber alcanzado los objetivos propuestos al comienzo de este trabajo, quedan muchos ámbitos que pueden ser mejorados en versiones posteriores de la aplicación. En este apartado remarcaremos algunos de los más importantes en orden de prioridad:

- Aumento de la cobertura de testeo: Como ya se ha mencionado en el apartado 6.2, la creación de nuevas pruebas unitarias y de integración se considera el trabajo a futuro más prioritario actualmente.
- Implementación de notificaciones diarias que recuerden la necesidad de realizar test: Si revisamos la tabla comparativa 2.1 podremos observar que esta funcionalidad está incluida en nuestra aplicación. Sin embargo, tras revisar la complejidad que supone la implementación de esta funcionalidad se decidió contemplarla como un trabajo a futuro para así poder primar el cumplimiento de los objetivos estipulados al comienzo de la memoria.
- Adición de preguntas para evaluar la concentración: Durante nuestro estudio de la sintomatología de las EN se observó que algunas fuentes ponían de manifiesto

deterioros significativos en la capacidad de concentración de los pacientes con estas enfermedades. La tarea conocida como *"Test de Stroop"* [40] es una herramienta potente para estimular y evaluar las capacidades de concentración que poseen los pacientes. En esta versión se ha decidido prescindir de este tipo de preguntas debido a la complejidad que supondría su implementación. No obstante, su adición a una futura versión de la aplicación se contempla como una tarea especialmente interesante.

- Generación de los informes de puntuación en formato pdf: Otro de los cambios que se contemplan es cambiar los informes de puntuaciones para que se generen en formato pdf. Este formato resulta mucho más versátil que un correo electrónico. Con este cambio también se contempla revisar la información que se incluye en estos informes y añadir nuevas estadísticas.

- Consulta con especialistas: En el momento de la redacción de esta memoria, *MindTest* ha sido ideada y diseñada gracias al estudio de artículos y documentación sobre enfermedades neurodegenerativas. No obstante, la idea de que nuestra aplicación sea probada y analizada por un especialista sanitario que nos ayude a dotarla de un mayor rigor médico, se contempla como un trabajo de gran interés para el proyecto.

- Migración a la nueva arquitectura de *React Native*: La nueva arquitectura de *React Native* sustituye el concepto de *bridge* por la *JavaScript Interface*, lo que permite mejorar enormemente el rendimiento de las aplicaciones. La migración a esta nueva arquitectura se contempla como un trabajo interesante para futuras actualizaciones.

Bibliografía

[1] La matriz de Stacey para elegir proyecto "ágil" o "predictivo". Consultar a https://www.obsbusiness.school/blog/la-matriz-de-stacey-para-elegir-proyecto-agil-o-predictivo Última consulta 26/08/2023.

[2] What Is Kanban? Explained for Beginners. Consultar a https://kanbanize.com/kanban-resources/getting-started/what-is-kanban. Última consulta 26/08/2023.

[3] The 2020 Scrum Guide. Consultar a https://scrumguides.org/scrum-guide.html#purpose-of-the-scrum-guide. Última consulta 05/09/2023.

[4] Sánchez-Gutiérrez, C., Ortega-Bastidas, P., & Cano-de-la-Cuerda, R. Aplicaciones móviles en la enfermedad de Alzheimer. Una revisión sistemática de la literatura. *Rehabilitación*, 53(4), 247-275 (2019).

[5] Web oficial de la aplicación MindMate. Consultar a https://www.mindmate-app.com/. Última consulta 26/08/2023.

[6] Web oficial de la aplicación YoTeCuido Alzheimer. Consultar a http://www.yotecuidoalzheimer.com/. Última consulta 27/08/2023.

[7] Web oficial de la aplicación Elevate. Consultar a https://elevateapp.com/. Última consulta 27/08/2023.

[8] Web oficial de la aplicación Stimulus. Consultar a https://stimuluspro.com/. Última consulta 27/08/2023.

[9] Página de la aplicación Impulse en la App store de Ios. Consultar a https://apps.apple.com/es/app/impulse-juegos-mentales/id1451295827. Última consulta 27/08/2023.

[10] ÏEEE Recommended Practice for Software Requirements Specifications,ïn *IEEE Std 830-1998* , vol., no., pp.1-40, 20 Oct. 1998. Electronic ISBN:978-0-7381-0448-5

[11] UML Class and Object Diagrams Overview. Consultar a https://www.uml-diagrams.org/class-diagrams-overview.html. Última consulta 05/09/2023.

[12] Cortizo, A. M. *Enfermedades metabólicas hereditarias*. Libros de Cátedra. Editorial de la Universidad Nacional de La Plata (EDULP). ISBN: 978-950-34-1978-6 (2021).

[13] Tàrraga, L., & Rovira, M. B. I. *Volver a empezar: ejercicios prácticos de estimulación cognitiva para enfermos de Alzheimer*. Glosa Ediciones. ISBN: 84-7429-067-8 (2000).

[14] Cardozo, C., Martin, A., & Saldaño, V. Recomendaciones de diseño para mejorar la experiencia de los usuarios adultos mayores con Facebook en dispositivos tablet. *Informes Científicos Técnicos-UNPA*, 10(2), 1-32 (2018).

[15] Diez heurísticas de usabilidad para el diseño de interfaces de usuario. Consultar a https://www.nngroup.com/articles/ten-usability-heuristics/. Última consulta 28/08/2023..

[16] Kane, L., & Pernice, K. *UX design for seniors (Ages 65 and older)*. Nielsen Norman Group (2020).

[17] Budiu, R., & Nielsen, J. *User experience for mobile applications and websites: Design Guidelines for Improving the Usability of Mobile Sites and Apps*. Nielsen Norman Group (2015).

[18] Williams, D., Alam, M. A. U., Ahamed, S. I., & Chu, W. *Considerations in designing human-computer interfaces for elderly people.* In 2013 13th *International Conference on Quality Software* (pp. 372-377). IEEE. (2013, July).

[19] Dodd, C., Athauda, R., & Adam, M. *Designing user interfaces for the elderly: a systematic literature review* (2017). ACIS 2017 Proceedings. 61.

[20] Arquitectura en Capas. Consultar a https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/capas. Última consulta 30/08/2023.

[21] JavaScript. Consultar a https://developer.mozilla.org/es/docs/Web/JavaScript. Última consulta 30/08/2023.

[22] TypeScript: qué es, diferencias con JavaScript y por qué aprenderlo. Consultar a https://profile.es/blog/que-es-typescript-vs-javascript/. Última consulta 30/08/2023.

[23] React Native: Aprende todo sobre el Framework Javascript de Facebook. Consultar a https://ciberninjas.com/react-native/. Última consulta 30/08/2023.

[24] React Native official documentation. Consultar a https://reactnative.dev/docs/getting-started. Última consulta 30/08/2023..

[25] How does React Native work? Understanding the architecture. Consultar a https://medium.com/front-end-weekly/how-does-react-native-work-understanding-the-architecture-d9d714e402e0. Última consulta 30/08/2023.

[26] Expo official documentation. Consultar a https://docs.expo.dev/. Última consulta 31/08/2023.

[27] Android Studio official documentation. Cómo crear y administrar dispositivos virtuales. Consultar a https://developer.android.com/studio/run/managing-avds?hl=es-419. Última consulta 31/08/2023.

[28] Qué es Baas, Backend as a service. Consultar a https://www.tokioschool.com/noticias/backend-as-service/. Última consulta 31/08/2023.

[29] Firebase Authentication official documentation. Consultar a https://firebase.google.com/docs/auth?hl=es-419. Última consulta 31/08/2023.

[30] Cloud Storage official documentation. Consultar a https://firebase.google.com/docs/storage?hl=es-419. Última consulta 31/08/2023..

[31] Cloud Firestore official documentation. Consultar a https://firebase.google.com/docs/auth?hl=es-419. Última consulta 31/08/2023.

[32] Visual Studio Code official documentation. Consultar a https://code.visualstudio.com/docs. Última consulta 01/09/2023.

[33] EmailJS official documentation. Consultar a https://www.emailjs.com/docs/introduction/how-does-emailjs-work/. Última consulta 01/09/2023.

[34] Props. React Native official documentation. Consultar a https://reactnative.dev/docs/props.html. Última consulta 01/09/2023.

[35] React Navigation official documentation. Consultar a https://reactnavigation.org/docs/getting-started. Última consulta 01/09/2023.

[36] Modelo de datos de Cloud Firestore. Consultar a https://firebase.google.com/docs/firestore/data-model?hl=es-419. Última consulta 01/09/2023.

[37] Jest official documentation. Consultar a https://jestjs.io/docs/getting-started. Última consulta 02/09/2023.

[38] React Native Testing Library official documentation. Consultar a https://testing-library.com/docs/react-native-testing-library/intro/. Última consulta 02/09/2023.

[39] What is Test Driven Development (TDD)?. Consultar a https://www.browserstack.com/guide/what-is-test-driven-development. Última consulta 02/09/2023.

[40] Test de Stroop: así evalúa la capacidad atencional e inhibitoria. Consultar a https://psicologiaymente.com/psicologia/test-de-stroop. Última consulta 02/09/2023.

[41] Objetivos de desarrollo sostenible, pagina oficial de Naciones Unidas. Consultar a https://www.un.org/sustainabledevelopment/es/. Última consulta 05/09/2023.

APÉNDICE A
Ejemplos de código

A.1 Código del componente ButtonComponent

```
import React from "react";
import { Text, View, Pressable } from "react-native";
import { StyleSheet } from "react-native";
import { Dimensions } from "react-native";

type buttonProps = {
  text: string;
  function: Function;
  type: boolean;
  styleSize: any;
  fontSize: number;
};

export default function ButtonComponent(props: buttonProps) {
  return (
    <View style={props.styleSize}>
      <Pressable
        style={props.type ? styles.buttonType1 : styles.buttonType2}
        onPress={() => {
          props.function();
        }}
      >
        <Text
          style={
            props.type
              ? {
                  fontFamily: "Kanit_300Light",
                  fontSize: props.fontSize,
                  color: "#4B7BE5",
                }
              : {
                  fontFamily: "Kanit_300Light",
                  fontSize: props.fontSize,
                  color: "white",
                }
          }
        >
          {props.text}
        </Text>
      </Pressable>
    </View>
  );
}

```

```
const windowWidth = Dimensions.get("window").width;

const styles = StyleSheet.create({
  buttonType1: {
    borderWidth: 2,
    flex: 1,
    borderRadius: 5,
    borderColor: "#4B7BE5",
    backgroundColor: "white",
    justifyContent: "center",
    alignItems: "center",
  },
  buttonType2: {
    flex: 1,
    borderRadius: 5,
    backgroundColor: "#4B7BE5",
    justifyContent: "center",
    alignItems: "center",
  },
});
```

A.2 Código del servicio TestService

```
import { ITestRepository } from "../dominio/test/ITestRepository";
import { TestRepository } from "../dominio/test/TestRepository";
import { PerteneceRepository } from "../dominio/pertenece/PerteneceRepository";
import { IPerteneceRepository } from "../dominio/pertenece/IPerteneceRepository
    ";
import { IPreguntaRepository } from "../dominio/pregunta/IPreguntaRepository";
import { PreguntaRepository } from "../dominio/pregunta/PreguntaRepository";
import { PuntuacionRepository } from "../dominio/puntuacion/
    PuntuacionRepository";
import { AuthRepository } from "../auth/AuthRepository";
import { IAuthRepository } from "../auth/IAuthRepository";
import { IPuntuacionRespository } from "../dominio/puntuacion/
    IPuntuacionRepository";

export class TestService {
  constructor() {}
  private authRepository: IAuthRepository = new AuthRepository();
  private testRepository: ITestRepository = new TestRepository();
  private perteneceRepository: IPerteneceRepository = new PerteneceRepository()
      ;
  private preguntaRepository: IPreguntaRepository = new PreguntaRepository();
  private puntuacionRepository: IPuntuacionRespository =
    new PuntuacionRepository();

  async generateTest(categoria: string): Promise<any> {
    var tests = await this.testRepository
      .getTestByCategoria(categoria)
      .catch((error) => {
        throw new Error(error.message);
      });
    var random = Math.floor(Math.random() * tests.length);
    console.log(tests);
    var testUid = tests[random].getUid();
    var pertenece = await this.perteneceRepository
      .getPerteneceByUidTest(testUid)
      .catch((error) => {
        throw new Error(error.message);
      });
    console.log(pertenece);
```

```
    var preguntasUid: string[] = [];
    pertenece.forEach((data) => {
      preguntasUid.push(data.getUidPregunta());
    });
    var preguntas = await this.preguntaRepository
      .getPreguntasByUids(preguntasUid)
      .catch((error) => {
        throw new Error(error.message);
      });

    return [preguntas, testUid];
  }

  async addPuntuacion(testUid: string, puntuacion: number) {
    var currentUserUid = this.authRepository.getCurrentUserUid();
    await this.puntuacionRepository
      .addPuntuacion(testUid, puntuacion, currentUserUid)
      .catch((error) => {
        throw new Error(error.message);
      });
  }
}
```

A.3 Código de la entidad Puntuación

```
export class Puntuacion {
  constructor(
    private fecha: Date,
    private puntuacion: number,
    private testUid: string,
    private userUid: string
  ) {}
  public getUserUid(): string {
    return this.userUid;
  }
  public setUserUid(value: string) {
    this.userUid = value;
  }
  public getPuntuacion(): number {
    return this.puntuacion;
  }
  public setPuntuacion(value: number) {
    this.puntuacion = value;
  }
  public getFecha(): Date {
    return this.fecha;
  }
  public setFecha(value: Date) {
    this.fecha = value;
  }
  public getTestUid(): string {
    return this.testUid;
  }
  public setTestUid(value: string) {
    this.testUid = value;
  }
}
```

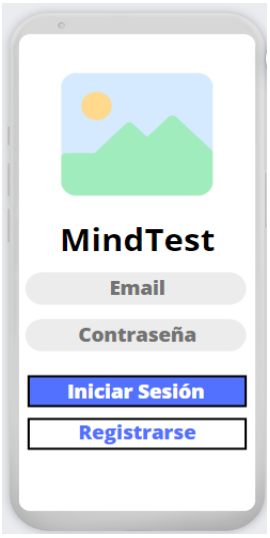
A.4 Código de la clase repositorio TestRepository

```
import { Firestore, getFirestore } from "firebase/firestore";
import { query, collection, where, getDocs } from "firebase/firestore";
import { ITest, Test } from "./Test";
import { ITestRepository } from "./ITestRepository";
import app from "../../config/firebase";
export class TestRepository implements ITestRepository {
  constructor() {}

  private db: Firestore = getFirestore(app);

  async getTestByCategoria(categoria: string): Promise<ITest[]> {
    const puntuacion = collection(this.db, "Test");
    const q = query(puntuacion, where("categoria", "==", categoria));
    const qRes = await getDocs(q).catch((error) => {
      throw new Error("Error en getTestByCategoria:" + error);
    });
    var res: ITest[] = [];
    qRes.forEach((doc) => {
      res.push(new Test(doc.data().categoria, doc.data().uid));
    });
    return res;
  }

  async getTestByUids(uids: string[]): Promise<ITest[]> {
    const test = collection(this.db, "Test");
    const q2 = query(test, where("uid", "in", uids));
    const q2Res = await getDocs(q2).catch((error) => {
      throw new Error("Error en getTestByUids:" + error);
    });
    var res: ITest[] = [];
    q2Res.forEach((doc) => {
      res.push(new Test(doc.data().categoria, doc.data().uid));
    });
    return res;
  }
}
```

A.4.1. Interfaz de la clase TestRepository

```
import { ITest } from "./Test";
export interface ITestRepository {
  getTestByCategoria(categoria: string): Promise<ITest[]>;
  getTestByUids(uids: string[]): Promise<ITest[]>;
}
```

APÉNDICE B
Mockups



(a) Mockup de la pantalla de inicio de sesión.

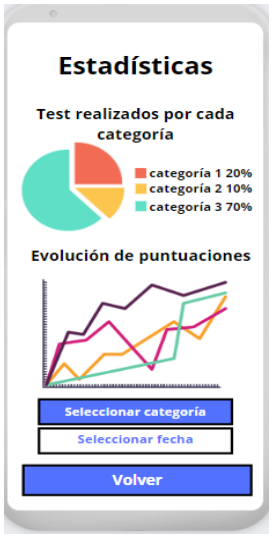
(b) Mockup de la pantalla de registro.

Figura B.1: Mockups de las pantallas de inicio de sesión y registro.

(a) Mockup de la pantalla principal.

(b) Mockup de la pantalla para realizar tests.

Figura B.2: Mockups de la pantalla principal y la pantalla para realizar tests.



(a) Mockup de la de estadísticas.

(b) Mockup de la pantalla de configuración.

Figura B.3: Mockups de la pantalla de estadísticas y de la pantalla de configuración.

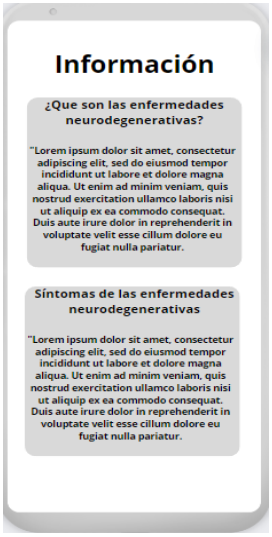


Figura B.4: Mockup de la pantalla con información y contactos relevantes.

APÉNDICE C
Objetivos de Desarrollo Sostenible

Los Objetivos de Desarrollo Sostenible ODS [41] son un conjunto de 17 objetivos (Figura C.1) aprobados por la ONU en 2015 con el propósito de lograr un futuro mejor y más sostenible para todos.

Figura C.1: Objetivos de desarrollo sostenible.

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**	x			
ODS 4. **Educación de calidad.**	x			
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. **Energía asequible y no contaminante.**				x
ODS 8. **Trabajo decente y crecimiento económico.**				x
ODS 9. **Industria, innovación e infraestructuras.**			x	
ODS 10. **Reducción de las desigualdades.**			x	
ODS 11. **Ciudades y comunidades sostenibles.**				x
ODS 12. **Producción y consumo responsables.**				x
ODS 13. **Acción por el clima.**				x
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Tabla C.1: Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

De entre todos estos objetivos existen cuatro que tienen relación con nuestra propuesta. Dos con una relación de nivel alto y otros dos con una relación más baja:

- Relación alta:
  - Garantizar una vida sana y promover el bienestar para todos en todas las edades: Como se ha mencionado en el apartado introductorio el aumento de la esperanza de vida conlleva una proliferación de las enfermedades neurodegenerativas. Este tipo de enfermedades suponen actualmente uno de los mayores riesgos para la salud de las personas de la tercera edad, así como, uno de los principales desafíos para la ciencia y la investigación médica en este momento. Las repercusiones de estas enfermedades no solo afectan a las personas que las padecen, sino también a su entorno familiar y social más próximo.La detección precoz de los síntomas ha demostrado ser de gran ayuda para establecer tratamientos y terapias antes de que el deterioro neurológico avance.

    Nuestra propuesta está completamente orientada a perseguir este objetivo. Este proyecto busca ayudar a reducir el impacto que las enfermedades neurodegenerativas causan en la sociedad y, de esta manera, mejorar el bienestar de las personas de edad avanzada.
  - Garantizar la educación de calidad: Nuestra propuesta busca combatir la desinformación, educar y concienciar sobre los síntomas y diferentes rasgos de estas enfermedades. Brindar información relevante y contrastada es vital para crear una sociedad más consciente de los rasgos que se dan en personas que comienzan a padecer estas enfermedades, para que así se puedan detectar cuanto antes y comenzar con el tratamiento necesario.
- Relación baja:
  - Reducción de las desigualdades: Nuestra aplicación se concibe como un producto gratuito. Su carácter multiplataforma permite el acceso a esta aplicación desde gran variedad de dispositivos. Estas dos decisiones pretenden facilitar el uso de esta herramienta por personas de cualquier estrato social.
  - Industria, innovación e infraestructura: *MindTest* busca promover el uso de nuevas tecnologías, como pueden ser los telefonos inteligentes, en un nicho de usuarios poco familiarizados con estas. Nuestra aplicación busca ser una propuesta innovadora que, además de cumplir con su propósito principal, permita a las personas de edad avanzada a sentirse más cómodos utilizando estos dispositivos.