UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

BreathBank

Presentado por Eduardo García de Leániz Juarros
en Universidad de Burgos — 4 de mayo de 2025
Tutor: Ana Serrano Mamolar

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Eduardo García de Leániz Juarros, con DNI 71482319K, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado BreathBank.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 4 de mayo de 2025

Vº. Bº. del Tutor:

D. nombre tutor

Vº. Bº. del co-tutor:

D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice general iii

Índice de figuras v

Índice de tablas vi

1. Introducción 1
- 1.1. Estructura de la memoria . . . 2
- 1.2. Estructura de los anexos . . . 2

2. Objetivos del proyecto 3
- 2.1. Objetivos generales . . . 3
- 2.2. Objetivos técnicos . . . 3
- 2.3. Objetivos personales . . . 4

3. Conceptos teóricos 5
- 3.1. Conceptos sobre el tema de la aplicación . . . 5
- 3.2. Conceptos sobre el software . . . 7

4. Técnicas y herramientas 13
- 4.1. Metodologías . . . 13
- 4.2. Patrones de diseño aplicados en el desarrollo . . . 17
- 4.3. Hosting del repositorio – *GitHub* . . . 20
- 4.4. Gestión del proyecto – GitHub Projects . . . 21
- 4.5. Comunicación – Outlook y Microsoft Teams . . . 21
- 4.6. Entorno de Desarrollo Integrado (IDE) . . . 21
- 4.7. Servicios de Backend . . . 24

5. Aspectos relevantes del desarrollo del proyecto **29**
5.1. De donde surge? 30
5.2. Metodologias 30
5.3. Formacion, libros, recursos... 30
5.4. Diagramas 30

6. Trabajos relacionados **31**
6.1. Artículos 31
6.2. Apps 31

7. Conclusiones y Líneas de trabajo futuras **33**

Bibliografía **35**

Índice de figuras

Índice de tablas

4.1. Resumen de la aplicación de *Scrum* en el proyecto 16
4.2. BD Relacionales VS BD No Relacionales 26
4.3. Firestore DB VS Realtime DB 28

1. Introducción

Descripción del contenido del trabajo y del estructura de la memoria y del resto de materiales entregados.

La velocidad y el ritmo de vida que lleva la sociedad hoy en día, es incomparable al que podían llevar hace unas décadas. Este aumento exponencial en las responsabilidades y en la inmediatez, ha sido impulsado principalmente por el desarrollo de nuevas tecnologías, que no solo permite esta interconexión global constante sino que muchas veces la exigen para no quedarse atrás. Esto causa que cada vez más personas sufran de estrés, ansiedad y otras consecuencias asociadas.

Es evidente que es muy dificl reducir las obligaciones o el tiempo de conexión a ciertos medios digitales, tanto en el tiempo de trabajo como en el de ocio. Pero lo que sí se puede hacer, es transformar el uso de estos medios en benificio de nuestra salud física y mental. Este proyecto busca no solo proporcionar una herramienta práctica para cumplir con este objetivo, sino también demostrar el impacto positivo que la tecnología puede tener en el cuidado de la salud personal.

BretahBank es una aplicación para dispositivos móviles que pretende ser un instrumento al alcance de la mano, para que los usuarios, de una forma personalizada, puedan desconectar y relajarse por unos instantes de este mundo tan frenético que nos rodea. Ser conscientes de nuestra respiración y ejercitarla diariamente aporta no solo benificios a nivel físico, como aumentar la capacidad pulmonar, estabilizar el Sistema Nervioso Central o fortalecer el Sistema Inmunitario, sino también a nivel psicológico, como alcanzar una mayor capacidad de concentración, reducir la irascibilidad y aumentar la energía y la vitalidad.

1.1. Estructura de la memoria

Estructura de la memoria

- Introducción
- Objetivos del proyecto
- Conceptos teóricos
- Técnicas y herramientas
- Aspectos relevantes del desarrollo del proyecto
- Trabajos relacionados
- Conclusiones y líneas de trabajo futuras

1.2. Estructura de los anexos

Estructura del anexo

- Plan de Proyecto
- Requisitos
- Diseño
- Manual de Usuario
- Manual de Programador

2. Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se distinguen objetivos técnicos marcados por los requisitos del software a construir y los objetivos de carácter más personal que motivaron la realización de este TFG.

2.1. Objetivos generales

- Desarrollar una aplicación multiplataforma para ayudar a los usuarios a adquirir el hábito de realizar respiraciones conscientes y sus beneficios asociados.
- Ofrecer al usuario un seguimiento personalizado y ejercicios para mejorar en sus habilidades respiratorias.
- Almacenar los datos de los usuarios de manera organizada y responsable, para representar su progreso y otras estadísticas.

2.2. Objetivos técnicos

- Aplicar la metodología ágil *Scrum* (4.1) durante el desarrollo del proyecto para su organización.
- Utilizar *GitHub* (4.3) como plataforma de control de versiones y colaboración, junto con *GitHub Projects* (4.4) para organizar y dar seguimiento al desarrollo del proyecto mediante tableros, facilitando una gestión ágil y ordenada de tareas.

- Llevar a cabo el desarrollo de una aplicación *full-stack*, teniendo en cuenta todos sus componentes: *backend*, *frontend*, infraestructura y lógica de negocio.
- Implementar la arquitectura MVC (Modelo-Vista-Controlador) en el desarrollo de la aplicación, con el fin de estructurar el código de forma modular, facilitar el mantenimiento y separar claramente la lógica de negocio de la interfaz de usuario.
- Integrar un sistema de *backend* en la aplicación, a través de *Firebase* (4.7), para gestionar el sistema de autenticación de usuarios y almacenar datos de forma segura y en tiempo real.
- Utilizar *Flutter* (4.6) como *framework* de desarrollo multiplataforma para crear una única base de código que permita desplegar la aplicación en dispositivos *Android* e *iOS*, optimizando tiempos y recursos de desarrollo.
- Desarrollar una interfaz intuitiva y cómoda para que el usuario se centre únicamente en los contenidos y no en cómo funciona la misma.

2.3. Objetivos personales

- Fomentar mi crecimiento personal al trabajar en un proyecto que promueva el bienestar de las personas, desarrollando herramientas que ayuden a mejorar su salud mental y física.
- Desarrollar mi capacidad para estructurar y organizar proyectos de software, mejorando la modularidad y mantenimiento del código.
- Mejorar mi capacidad para resolver problemas complejos durante el proceso de desarrollo de software, enfrentando desafíos técnicos y encontrando soluciones eficaces.
- Adquirir experiencia práctica en la gestión de la privacidad y seguridad de los datos de los usuarios, especialmente en el contexto de aplicaciones que requieren el almacenamiento de información sensible.
- Aumentar mi comprensión de la importancia de la usabilidad y la experiencia del usuario en el diseño de aplicaciones móviles, garantizando que la interfaz sea intuitiva y fácil de usar.

3. Conceptos teóricos

3.1. Conceptos sobre el tema de la aplicación

Inversión

Se denomina ***inversión*** al ejecicio o prueba que realiza el usuario de la aplicación con el objetivo de mejorar sus capacidades pulmonares y de obtener los beneficios de la respiración consciente. El término hace referencia a la inversión de un bien, como el tiempo personal, para recoger en un futuro resultados favorables a largo plazo en el ámbito personal.

Inversor

Recibe el nombre de ***inversor*** cada usuario de la aplicación *BreathBank* que destina su tiempo a la práctica de respiraciones conscientes con el objetivo de obtener beneficios en su bienestar físico y mental, respaldado por estudios científicos.

Respiración consciente

Es aquella en la que nuestra mente y nuestros pensamientos están enfocados en la propia respiración. La intención puede ir desde únicamente ser consciente de nuestra respiración a modificarla también conscientemente aumentando o disminuyendo la intensidad y la duración de la inspiración y de la espiración o la duración de las pausas entre ellas.

Ciclos y series respiratorias

Ciclo respiratorio (C.R): realización de 3 ***respiraciones conscientes*** consecutivas.

Serie respiratoria (S.R): realización de 3 ***ciclos respiratorio (C.R)*** consecutivos.

Tipos de respiraciones según musculos implicados

- ***Respiración abdominal***: Solo se contrae el músculo diafragma y se expande principalmente la mitad inferior o zona abdominal. Es la respiración más habitual y que hacemos de manera involuntaria.

- ***Respiración torácica***: Además del diafragma, se contraen otros músculos accesorios que provocan una mayor expansión de la caja torácica, quedando la zona abdominal sin apenas expansión o incluso retrayéndose.

- ***Respiración global o integral***: Resulta de la unión de los dos tipos de respiración anteriores. Como resultado, se produce una expansión de ambas cavidades, abdominal y torácica, ya sea de forma simultánea o secuencial.

Niveles de práctica

Cada nivel viene determinado por la capacidad del ***inversor*** de respirar de forma más lenta, buscando aumentar el tiempo que puede inspirar y espirar de forma continuada sin que aparezca tensión en el cuerpo ni en la mente. Estos están diseñados para adaptarse progresivamente a las capacidades respiratorias de cada ***inversor***, que irá subiendo de nivel en base a la experiencia que vaya adquiriendo y a la mejora en su práctica diaria. El pertenecer a un nivel u otro afecta a la duración de las respiraciones durante las ***inversiones***, adaptándose a las capacidades del usuario.

Listón de la inversión

Dentro de cada nivel se estipulan unos ***listones de inversión***. Estos listones pueden ser elegidos por el usuario para progresar dentro de cada nivel y prepararse para pasar al siguiente. Cada listón modifica las ***inversiones*** con pruebas un poco más exigentes o periodos de respiraciones más largos.

Tipo de inversiones

Se ofrecen dos formas de realizar inversiones dentro de la aplicación. En ambos casos la duración de las respiraciones vendrán marcadas por el ***nivel de inversor*** y por el ***listón de la inversión*** elegido.

- ***Manual:*** en la inversión manual...
- ***Guiada:*** en la inversión guiada...

3.2. Conceptos sobre el software

Aplicación Híbrida

Una ***aplicación híbrida*** es un tipo de *software* móvil que combina elementos de aplicaciones nativas y aplicaciones web. Se desarrolla utilizando tecnologías web estándar pero se empaqueta dentro de un contenedor nativo que le permite acceder a funcionalidades del dispositivo. Esto permite que la aplicación se ejecute en múltiples plataformas como *Android* e *iOS* sin necesidad de escribir el código desde cero para cada una.

Emulador

Un ***emulador*** es un *software* que permite simular el entorno de otro sistema operativo o dispositivo. En el desarrollo de aplicaciones móviles, los emuladores son utilizados para ejecutar y probar aplicaciones sin necesidad de un dispositivo físico, imitando las características de un teléfono o tablet. Esto facilita las pruebas en diferentes dispositivos y versiones del sistema operativo.

Interfaz / Frontend

La ***interfaz*** es el medio o mecanismo a través del cual los usuarios o sistemas interactúan con un *software* o *hardware*. A través de ella se produce el intercambio de información entre la persona que la utiliza y el programa que esté utilizando.

Backend

El ***backend*** es la parte de la aplicación que gestiona el procesamiento de datos y la lógica de negocio. Aunque no es visible para el usuario, es

fundamental para que la aplicación funcione correctamente. El *backend* se encarga de tareas como el manejo de bases de datos, la autenticación de usuarios y la integración con otros servicios.

Base de datos no relacional

Una ***base de datos no relacional*** *(NoSQL)* es un tipo de base de datos que no sigue el modelo de tablas y relaciones de las bases de datos tradicionales. Son útiles cuando se manejan grandes volúmenes de datos no estructurados o cuando se necesita escalabilidad horizontal. Este tipo de bases de datos permiten almacenar datos en formatos más flexibles como documentos, clave-valor o grafos, lo cual es común en aplicaciones móviles que manejan datos dinámicos.

Persistencia

La ***persistencia*** en el desarrollo de aplicaciones móviles se refiere a la capacidad de guardar datos de manera permanente o a largo plazo, de modo que sobrevivan a la ejecución de la aplicación. Esto es crucial para aplicaciones que requieren almacenamiento de información, como configuraciones de usuario, historial o datos entre sesiones. Se puede lograr mediante bases de datos locales o servicios de almacenamiento en la nube.

Colecciones

En el contexto de bases de datos, una ***colección*** es una estructura que agrupa un conjunto de documentos o registros. En bases de datos *NoSQL* las colecciones son similares a las tablas en bases de datos relacionales, pero no requieren un esquema fijo. Se utilizan para almacenar y organizar datos de forma flexible, lo que se ajusta muy bien para aplicaciones móviles que necesitan manejar datos dinámicos y cambiantes.

Concurrencia

La ***concurrencia*** en este contexto hace referencia a la capacidad de la aplicación, de ejecutar múltiples tareas al mismo tiempo. No tienen porqué ejecutarse exactamente en el mismo instante, pero si producirse en momentos simultáneos. Este concepto aparece por ejemplo cuando el usuario quiere acceder, y está utilizando la interfaz a la vez que el sistema está verificando sus credenciales, o cuando se está guardando datos del usuario en la base de datos y mientras se puede cambiar de pantalla. Si esta funcionalidad no

está bien implementada puede perjudicar de gran manera al rendimiento de la aplicación.

Autenticación de usuarios

La ***autenticación de usuarios*** es el proceso que garantiza que solo los usuarios autorizados puedan acceder a ciertos recursos dentro de la aplicación. Esto se logra normalmente mediante un sistema de credenciales, como un nombre de usuario y una contraseña, aunque también se pueden utilizar métodos más avanzados, como la autenticación multifactor o biométrica.

Programación reactiva

La ***programación reactiva*** es un enfoque que se centra en gestionar flujos de datos y eventos de manera eficiente. En aplicaciones móviles, la *programación reactiva* permite que la interfaz de usuario reaccione automáticamente a los cambios en los datos, como la actualización de información en tiempo real. Patrones de diseño como el Estado o el Observador (4.2), son de mucha utilidad a la hora de implementar estas características.

API

Una ***API*** (Interfaz de Programación de Aplicaciones) es un conjunto de reglas y protocolos que permite que diferentes programas se comuniquen entre sí. Las *APIs* son fundamentales para integrar servicios externos, como sistemas de autenticación, sistemas gestores de bases de datos, etc. Permiten que la aplicación se conecte a otras plataformas de manera eficiente y controlada.

Debugging

El ***debugging*** es el proceso de identificar y corregir errores en el código de una aplicación. Normalmente, los editores de código (como *Visual Studio Code* [16]) siempre incorporan esta opción para facilitar el desarrollo del software. Estas herramientas de depuración sirven para revisar el flujo de la aplicación, inspeccionar variables y ver cómo se comporta el código durante la ejecución.

Hot Reload

El ***Hot Reload*** es una característica que permite ver los cambios del código de la aplicación de manera instantánea en un *emulador*, sin necesidad de reiniciar la aplicación. Esto mejora la productividad de los desarrolladores al reducir el tiempo de espera entre modificaciones en el código y su visualización en el dispositivo. En plataformas como *Flutter* (4.6), el *Hot Reload* facilita la iteración rápida en el proceso de desarrollo de aplicaciones móviles.

Dependencias

Se define ***dependencia*** a cualquier componente externo necesario para que funcione correctamente una aplicación. Normalmente suelen ser librerías, módulos o servicios. Para incluir estos elementos suele ser necesario modificar la configuración de arranque de la aplicación y descargar ciertos archivos en los que estén contenidas las herramientas que se quieren importar.

Arquitectura MVC

La ***arquitectura MVC (Modelo-Vista-Controlador)*** es un patrón de diseño que divide las aplicaciones en tres componentes principales:

- el Modelo, que gestiona los datos y la lógica de negocio;
- la Vista, que es responsable de la interfaz de usuario;
- y el Controlador, que maneja las interacciones del usuario y actualiza tanto el Modelo como la Vista.

Este patrón es útil para organizar el código de forma modular y mejorar la mantenibilidad de la aplicación móvil.

Inyecciones SQL

Las ***inyecciones SQL*** [5] son un tipo de vulnerabilidad de seguridad que ocurre cuando un usuario introduce código malicioso en una aplicación a través de ciertos puntos posiblemente más vulnerables, como campos de formularios, cajas de texto o similares, realmente cualquier punto donde el usuario pueda introducir información es un potencial punto de peligro. Este ataque puede tener la intención de acceder a la base de datos, modificarla o robar información de la misma. Es fundamental proteger las aplicaciones

móviles contra estas inyecciones utilizando medidas de seguridad como la validación adecuada de entradas y el uso de consultas parametrizadas.

4. Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

4.1. Metodologías

Scrum [2]

Scrum es una metodología de trabajo ágil utilizado principalmente para el desarrollo de software, aunque también se aplica en otros contextos. Su propósito es mejorar la colaboración, la flexibilidad y la entrega continua de valor en equipos de trabajo. En el caso de este proyecto no se ha seguido esta metodología tal cual, sino que se ha adaptado al contexto y características del mismo.

Roles

Para llevar a cabo esta metodología, se identifican distintos roles sus tareas asociadas:

- **Product Owner:** Es el encargado de representar al cliente y asegurar que se cumplen sus intereses. En este proyecto ese papel ha sido tomado por el ideador de *BreathBank* junto con la tutora de este TFG, los cuales han velado porque las necesidades que querían cubrir con esta aplicación de verdad se cumpliese.
- **Scrum Master:** Es la persona que dirige y coordina los eventos *Scrum* (que se detallan más adelante). En este caso esa figura la toma totalmente la tutora de este TFG.
- **Equipo Scrum o equipo de trabajo:** Formado por el conjunto de desarrolladores, ingenieros, técnicos, etc. que llevan a cabo el producto final. En el proyecto de *BreathBank* no ha sido un equipo, sino que todo el trabajo de desarrollo, pruebas, implementación, servicios ... ha caído sobre el alumno

Elementos *Scrum*

Son componentes fundamentales para la aplicación de este marco de trabajo. Se consideran artefactos a los objetos que se generan y utilizan como medios para abordar y cubrir los desafíos y requisitos del proyecto.

- ***Product Backlog:*** Lista de trabajo que recoge todas las tareas a finalizar para obtener el producto final y finalizar el proyecto. Estas tareas deben estar supervisadas por el Product Owner, pero en este proyecto fueron añadidas en su gran mayoría al principio por el alumno, aunque otras ciertas tareas tuvieron que ir añadiéndose según avanzaba el proyecto para cubrir imprevistos y errores que iban surgiendo.
- ***Sprint Backlog:*** Subconjunto de tareas del Product Backlog elegidas para completar durante el Sprint actual. Este Backlog deberá definirse teniendo en cuenta la cantidad de trabajo que se puede asumir en función del tiempo y de los recursos de los que se dispone. En este proyecto concreto, el Sprint Backlog se consensuaba entre la tutora y el alumno en cada reunión de Sprint.
- ***Incremento:*** Resultado del producto al finalizar cada Sprint. En el entorno profesional se requiere que este resultado sea funcional para enseñar los avances al cliente. En este proyecto, no siempre terminaba una versión funcional de la aplicación después de cada Sprint, puesto que había momentos más enfocados al desarrollo software y sin embargo otros más enfocados a la documentación o a la investigación, aunque se procuraba equilibrar ambas antes de cada reunión.

Eventos

Los eventos que se dan en esta metodología de planificación pueden variar en función del entorno donde se va a llevar a cabo, cultura de la empresa, proporción del proyecto... Estos son los más habituales:

- ***Sprint:*** Ciclo de trabajo regular y periódico durante el que se llavará a cabo las distintas tareas definidas. En el caso de este proyecto se establecieron Sprints con una duración de 14 días.
- ***Sprint Planning/Review:*** En la metodología *Scrum* tradicional se suelen diferenciar dos días distintos para la *Sprint Planning* y la *Sprint Review*. En este proyecto se aprovecha la misma reunión, la primera parte se destinará a revisar los avances y las tareas que deberían estar finalizadas en el anterior *Sprint*, y en base a esto se pasará a discutir qué tareas de las que quedan en el *Product Backlog* se añadirán al *Sprint* que va a comenzar. Al comienzo del proyecto, cuando las tareas eran más enfocadas a investigación y preparación de herramientas, las reuniones fueron telemáticas. Sin embargo, cuando las tareas fueron enfocandose más en la aplicación en sí y en su comporetamiento, el alumno y la tutora vieron más beneficioso llevar a cabo las *Sprint Reviews* presencialmente con el objetivo de solventar mejor las incidencias y de manera más rápida.
- ***Daily Scrum:*** Reunión breve diaria donde el equipo de desarrollo se junta para revisar el avance y las tareas que quedan por completar antes de finalizar el *Sprint*. En este caso el alumno, antes de comenzar a ponerse a desarrollar, dedica unos 10 - 15 minutos a revisar las tareas pendientes en el *Sprint Backlog* y a decidir cuales abordará en ese mismo día.

Beneficios de usar *Scrum* en este proyecto de desarrollo

La adaptación de la metodología *Scrum* al contexto específico de este proyecto ha supuesto una serie de beneficios significativos que han contribuido positivamente al desarrollo y a la organización del trabajo:

- **Mejor organización y planificación del trabajo:** El uso del *Product Backlog* y del *Sprint Backlog* permitió estructurar y priorizar las tareas de forma clara, ayudando a mantener el foco en lo más relevante en cada momento según pasaban las semanas.

Elemento	**Descripción**
Product Owner	Ideador del proyecto + Tutora
Scrum Master	Tutora del TFG
Equipo de desarrollo	Alumno (desarrollador único)
Duración del Sprint	14 días
Sprint Meeting	Reunión conjunta (30' revisión + 30' planificación)
Daily Scrum	Revisión individual diaria (10' - 15')
Product Backlog	Ajustado durante el desarrollo
Sprint Backlog	Consensuado en cada Sprint con la tutora
Incremento	Depende del enfoque del Sprint (desarrollo o documentación)
Colaboración	Constante entre alumno, tutora e ideador

Tabla 4.1: Resumen de la aplicación de *Scrum* en el proyecto

- **Seguimiento continuo del progreso:** Gracias a los *Sprints* quincenales y a las reuniones de planificación y revisión, se pudo hacer un seguimiento constante de los avances y ajustar la dirección del proyecto según las necesidades e inconvenientes que iban surgiendo.

- **Mayor capacidad de adaptación:** Al no tratarse de un entorno con requisitos totalmente cerrados desde el inicio, el enfoque ágil permitió incorporar nuevas tareas y resolver imprevistos de forma flexible, sin que esto supusiera una disrupción grave en el desarrollo.

- **Refuerzo de la autoevaluación y la autogestión:** La práctica diaria de revisar el estado del Sprint Backlog permitió al alumno mejorar su capacidad de autogestión, planificar sus jornadas de trabajo con mayor eficiencia y tomar decisiones informadas en función del estado del proyecto.

- **Equilibrio entre desarrollo técnico y documentación:** La estructura cíclica de *Scrum* facilitó encontrar un balance adecuado entre las distintas fases del proyecto (desarrollo, investigación, documentación), asegurando que ninguna de ellas quedara desatendida, y fuesen avanzando todas de manera equilibrada.

4.2. Patrones de diseño aplicados en el desarrollo

A continuación se presentan algunos patrones de diseño [6] que se suelen utilizar en el desarrollo de software y han sido implementados en el código de la aplicación.

Plantilla

Descripción del patrón

Define el esqueleto de un algoritmo en una clase base, pero permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura general.

Aplicación práctica

Por definir...

Ventajas

Permite reducir la cantidad de código duplicado o redundante, y seguir un mismo criterio para realizar ciertas tareas. Además, facilita la extensión de algoritmos sin modificarlos (principio de Abierto/Cerrado [13]) y aplica el principio de Inversión de Dependencias [14]

Estrategia

Descripción del patrón

Permite definir una familia de algoritmos, encapsularlos y hacerlos intercambiables. El algoritmo puede variar independientemente del cliente que lo usa.

Aplicación práctica

Aunque no está implementado cambiar tema de la aplicacion (claro/oscuro). Por definir...

Ventajas

Cambiar dinámicamente el comportamiento de un objeto, reduce la cantidad de código necesaria para manejar condiciones de conducta (múltiples if/else) y fomenta el princpio de Abierto/Cerrado

Comando

Descripción del patrón

Encapsula una solicitud como un objeto, permitiendo parametrizar clientes con diferentes solicitudes, encolar o registrar operaciones, y soportar deshacer operaciones.

Aplicación práctica

Navegación entre pantallas de la aplicación. Por definir...

Ventajas

Facilita implementar funcionalidades de hacer/rehacer/deshacer o de navegación adelante/atrás, desacopla al emisor del receptor de una acción y permite registrar, parametrizar o encolar tareas o acciones.

Singleton

Descripción del patrón

Asegura que una clase tenga una única instancia y proporciona un punto de acceso global a ella.

Aplicación práctica

Instancia única a la base de datos y servicio de autenticación, minimizando el gasto de recursos y problemas de concurrencia 3.2

Ventajas

Asegura manejar una única instancia de un objeto y proporciona un punto de acceso global a ella, evitando la cantidad de código a modificar si se desea remplazar esa instancia por otra de otro tipo. Controla el acceso a recursos compartidos y evita la duplicación innecesaria de servicios reduciendo costes.

Observer

Descripción del patrón

Define una relación de dependencia uno-a-muchos entre objetos, de modo que cuando uno cambia su estado, todos sus dependientes son notificados automáticamente.

Aplicación práctica

Por definir...

Ventajas

Fomenta una arquitectura reactiva la cuál es una de las características del framewrok de desarrollo que he utilizado (3.2), permite actualizar automáticamenbte la interfaz de usuario cuando se producen cambio y mostrarselos en tiempo real al usuario.

Builder

Descripción del patrón

Separa la construcción de un objeto complejo de su representación, permitiendo construir diferentes representaciones con el mismo proceso.

Aplicación práctica

Crear widgets de la interfaz similares como botones, appbars, textfields...

Ventajas

Permite crear diferentes representaciones del mismo objeto lo cual es muy útil a la hora de construir los elementos de la interfaz al ser tan personalizable con *Flutter*. Además mejora la legibilidad al instanciar objetos con múltiples parámetros.

Repository

Descripción del patrón

Actúa como una capa intermedia entre la lógica de la aplicación y la fuente de datos. Encapsula la lógica de acceso a datos, de modo que el resto de la app no necesita saber de dónde vienen ni cómo se obtienen los datos.

Aplicación práctica

Por definir...

Ventajas

Centraliza la lógica de acceso a datos y la desacopla del resto de la app, facilita el mantenimiento y extensión del código, promueve el principio de Única Responsabilidad [15]

Fachada

Descripción del patrón

Proporciona una interfaz unificada y simplificada para un conjunto de interfaces en un subsistema. El patrón Fachada define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.

Aplicación práctica

Por definir...

Ventajas

Simplifica la complejidad de un sistema al ocultar los detalles de implementación de sus subsistemas, reduce el acoplamiento entre el cliente y los componentes internos, y promueve una separación clara de responsabilidades. También facilita la mantenibilidad y escalabilidad del sistema, ya que los cambios internos en el subsistema no afectan directamente al cliente.

4.3. Hosting del repositorio – *GitHub*

El repositorio del proyecto se ha alojado en la plataforma *GitHub*, una de las herramientas más utilizadas a nivel mundial para el control de versiones y colaboración en proyectos de software. Esta plataforma es sencilla, fácil de utilizar y muy integrable con distintos editores de código, como *Visual Studio Code*, para según vas desarrollando poder almacenar los cambios y tenerlo todo siempre actualizado. Solo con vincularlo con su herramienta *Github Desktop*, se pueden visualizar los cambios de todos los archivos y confirmarlos o revertirlos con apenas dos clicks.

Gracias a su interfaz intuitiva y funcionalidades avanzadas, *GitHub* ha permitido mantener el proyecto bien estructurado, facilitando la gestión del código fuente, la documentación y otros archivos asociados.

4.4. Gestión del proyecto – GitHub Projects

Para la planificación y seguimiento de las tareas del proyecto, se ha utilizado la funcionalidad de *GitHub Projects*, una herramienta integrada en GitHub que permite crear tableros *Kanban* personalizados. Ha sido realmente útil para llevar a cabo la metodología *Scrum* (4.1), clasificar las tareas según su estado (por hacer, en progreso, finalizadas) y vincular directamente las tareas con *issues* y *milestones*, para una mejor organización de las mismas.

El uso de esta herramienta ha contribuido a mantener una visión clara del progreso general del proyecto, facilitando tanto la planificación de los Sprints como el seguimiento de tareas pendientes o en curso. Además, su integración directa con el repositorio ha simplificado la trazabilidad de las decisiones técnicas y los avances conseguidos.

4.5. Comunicación – Outlook y Microsoft Teams

La comunicación, como en cualquier proyecto, siempre es una parte fundamental. En este caso los canales utilizados han sido dos. Las dudas, respuestas y avisos entre alumno y tutora se realizaban a través de *Outlook*, mientras que las reuniones telemáticas se organizaban a través de *Microsoft Teams*. Posteriormente, las reuniones pasaron a ser presenciales reduciendo el uso de *Teams* a partir de la mitad del desarrollo del proyecto.

4.6. Entorno de Desarrollo Integrado (IDE)

Flutter [9]

En el proceso de desarrollo de *BreathBank*, se evaluaron diversas opciones de *frameworks* que permitieran construir la aplicación para múltiples plataformas de manera eficiente. Sin embargo, se decidió optar por *Flutter* por una serie de características y ventajas que se comentan a continuación.

- **Desarrollo Multiplataforma con un solo código base:** Una de las principales razones para elegir *Flutter* es su capacidad para desarrollar aplicaciones para múltiples plataformas (Android, iOS) con una sola base de código. Esto no solo reduce significativamente el tiempo de desarrollo, ya que no es necesario duplicar esfuerzos para adaptarse a diferentes sistemas operativos.

  Flutter permite escribir el código una vez y ejecutarlo en ambas plataformas sin comprometer la funcionalidad o la experiencia del usuario, lo que es una ventaja clara frente a soluciones que requieren mantener bases de código separadas para cada plataforma.

- **Alto rendimiento en desarrollo y compilación:** A diferencia de otros frameworks que dependen de un puente o capas intermedias para comunicar el código con los sistemas operativos nativos, *Flutter* compila directamente a código nativo. Esto significa que las aplicaciones desarrolladas con *Flutter* tienen un rendimiento muy cercano al de aplicaciones nativas. Además, funciona de manera bastante rápida a la hora de mostrar los cambios en el dispositivo o en el emulador que se esté probando, aunque esto se hablará también en el apartado de Hot Reload.

- **Control sobre la UI y widgets personalizados:** *Flutter* se basa en un sistema de widgets para la creación de interfaces de usuario. Cada componente de la interfaz, desde botones hasta animaciones, es un widget, lo que ofrece un control totalmente personalizado sobre el diseño y el comportamiento de la aplicación.

- **Hot Reload: Desarrollo Ágil y Eficiente:** Una de las características más valoradas por los desarrolladores de *Flutter* es el *Hot Reload* (3.2). Esta funcionalidad permite a los programadores ver los cambios en el código en tiempo real sin necesidad de reiniciar la aplicación. Esto acelera enormemente el proceso de desarrollo y permite realizar pruebas e iteraciones rápidas sobre el diseño y la funcionalidad sin interrumpir el flujo de trabajo.

  El *Hot Reload* mejora también la productividad en el proceso de depuración, ya que facilita la corrección de errores y la optimización de la aplicación de forma inmediata. Este enfoque ágil es ideal para el desarrollo iterativo de características y mejora de la experiencia de usuario.

- **Fácil integración con Firebase:** *Flutter* destaca también por su fácil integración con *Firebase*, plataforma la cual he elegido para guardar

los datos de la aplicación. Firebase ofrece múltiples herramientas para gestionar la autenticación, bases de datos en tiempo real, almacenamiento de archivos y funciones en la nube, se verán en más detalle en el apartado de Bases de Datos. La integración con *Firebase* en *Flutter* se realiza de manera sencilla gracias a los paquetes oficiales proporcionados por el equipo de *Flutter*, lo que permite a los desarrolladores incorporar funcionalidades avanzadas sin tener que gestionar una infraestructura de *backend* compleja.

Dart [7]

Dart es un lenguaje de programación optimizado para aplicaciones móviles, web y de escritorio. Fue desarrollado por *Google* con el objetivo de proporcionar una plataforma eficiente para la creación de aplicaciones modernas. *Dart* es conocido por ser el lenguaje que utiliza el *framework Flutter* para el desarrollo de aplicaciones.

- **Sintaxis moderna y familiar:** *Dart* tiene una sintaxis que se asemeja a otros lenguajes como *JavaScript, C++ o Java*, lo que facilita su aprendizaje para programadores con experiencia en estos lenguajes.
- **Orientado a objetos:** *Dart* es un lenguaje orientado a objetos, lo que significa que utiliza clases y objetos para organizar el código y la lógica de las aplicaciones.
- **Asincronía y Concurrencia:** *Dart* proporciona soporte para programación asincrónica mediante *async y await*, facilitando la escritura de código no bloqueante y optimizando el rendimiento en tareas concurrentes (3.2). Esto es muy útil a la hora de implementar sistemas de autenticación o en operaciones de lectura o escritura en base de dastos.
- **Tipado estático:** *Dart* permite un tipado estático opcional, lo que ayuda a detectar errores en tiempo de compilación y mejora la mantenibilidad del código.

Visual Studio Code y extensiones

La elección del editor de código fuente ha cambiado a lo largo del desarrollo de este proyecto. Primero comencé utilizando Android Studio ya que pensé que quizá estaría un poco más integrado con el trabajo con aplicaciones móviles. Sin embargo, el no tener experiencia utilizándolo y

el bajo rendimiento que daba en mi equipo con el *emulador* (3.2) y a la hora de hacer *debug*, me hizo cambiar de opinión e inclinarme por **Visual Studio Code**.

Este editor me funcionaba mucho mejor, además de que ya tenía experiencia trabajando con él en proyectos anteriores. Además, para trabajar con *Flutter* y *Firebase* solo eran necesarias instalar 3 extensiones: **Flutterr, Dart y Awesome Flutter Snippets**. Otras partes de la configuración como es la instalación de *Flutter* y *Firebase*, y la integración del proyecto de una en otra si que fueron bastante más complejas, pero eso se detallará en su apartado correspondiente.

Latex Overleaf

Respecto a la redacción de la memoria y sus anexos, *Overleaf* resultó ser una herramienta muy cómoda. Hay múltiples editores de archivos de LaTex, ya que es un lenguaje cada vez más utilizado en documentos oficiales gracias a su control preciso de formato, modularidad, gestión automática de referencias y bibliografía, etc.

Algunos de los motivos por los que me incliné por *Overleaf* son: acceso desde cualquier lugar y sin necesidad de instalación, colaboración en tiempo real y compilación automática mientras editas.

Aunque el hosting del repositorio se llevó a través de *Github* (4.3), el control de cambios y versiones de la memoria y demás documentación se llevo a través de *Overleaf* por comodidad y rápidez en las correcciones.

4.7. Servicios de Backend

Falta añadir la parte de Cloud Messaging que estoy todavía implementando.

Uno de los pilares fundamentales dentro de el desarrollo de aplicaciones es **la gestión y el guardado de los datos** que vaya a utilizar la misma. Por ello es importante estudiar las opciones que hay e identificar cual se adapta mejor a cada caso en función de sus ventajas e inconvenientes.

Comparación entre distintos SGBD

En el mundo del desarrollo de aplicaciones, elegir el sistema de gestión de bases de datos (SGDB) adecuado es crucial para garantizar un rendimiento

óptimo y una buena experiencia de usuario. A día de hoy existen múltiples opciones a valorar en función de su arquitectura, escalabilidad y otras características que se han tenido en cuenta para elegir el gestor adecuado.

Las **bases de datos relacionales** ofrecen muchas ventajas, tienen una estructura muy definida, permiten realizar consultas más complejas de manera eficiente y ofrecen una mayor integridad y consistencia de los datos. Sin embargo, estas son características que en el contexto de este proyecto no se iban a aprovechar al máximo. Por ello, ciertos SGDB como MySQL, PostgreSQL, Oracle, SQLite fueron descartados como opciones para crear la base de datos de *BreathBank*.

Por otro lado, las **bases de datos no relacionales** ofrecían algunas ventajas que sí podían ser de mayor provecho para el contexto de desarrollo de una aplicación para móviles: mayor flexibilidad en el modelo de datos y facilidad para manejarlo, mayor capacidad de adaptación a cambios en los datos, mejor manejo de operaciones de lectura y escritura en tiempo real, mayor tolerancia a fallos y disponibilidad, etc. Pero sobre todo, la mayor ventaja que ofrecían en este caso, era una mayor facilidad a la hora de integrarlas con la aplicación y de conectarlas con las herramientas de desarrollo que estaba utilizando.

En concreto **Firebase** es una plataforma que ofrecía todas las herramientas para cubrir las necesidades de backend que requería la aplicación: autenticación de usuarios, almacenamiento de datos, comunicación entre frontend y backend, notificaciones, etc. y además está muy integarada con Flutter, el entorno de desarrollo que estoy utilizando

Firebase [8]

Firebase es una plataforma de desarrollo de aplicaciones móviles y web ofrecida por Google, que proporciona una variedad de servicios backend para ayudar a los desarrolladores a construir, gestionar y escalar aplicaciones de manera eficiente.

Servicios de Firebase utilizados en el proyecto

- **Firestore DB y Realtime DB:** Bases de datos NoSQL escalables que permiten almacenar y sincronizar datos en tiempo real entre todos los usuarios de la aplicación.

Característica	**BD Relacionales**	**BD No Relaciona**
Escalabilidad	↓	↑
Consistencia de datos (ACID)	↑	↓
Flexibilidad de esquema	↓	↑
Rendimiento en consultas simples	↓	↑
Consultas complejas (JOIN, GROUP BY, etc.)	↑	↓
Soporte para datos jerárquicos	↓	↑
Transacciones complejas	↑	↓
Manejo de datos offline (sin conexión)	↓	↑
Rendimiento en grandes volúmenes de datos	↓	↑
Manejo de relaciones complejas	↑	↓
Fácil de usar para desarrolladores	↑	↑
Manejo de datos en tiempo real	↓	↑
Costo de infraestructura	↓	↑

Tabla 4.2: BD Relacionales VS BD No Relacionales

- **Autenticación:** Servicio sencillo para gestionar el registro, inicio de sesión y autenticación de usuarios mediante múltiples proveedores como correo electrónico, Google, Facebook, etc.
- **Cloud Messaging:** Por completar...
- **Firebase Analytics:** Herramienta de análisis para comprender el comportamiento de los usuarios dentro de la aplicación.

Firestore DB vs Realtime DB

Tanto *Realtime Database* como *Firestore Database* son **bases de datos NoSQL en tiempo real** proporcionadas por **Firebase**, lo que significa que ambas permiten una sincronización instantánea de datos entre el cliente y el servidor. Ambas están optimizadas para aplicaciones móviles y web, con una infraestructura gestionada que facilita la **escalabilidad**. En cuanto a su estructura, ambas ofrecen almacenamiento flexible de datos (JSON en *Realtime Database* y documentos en *Firestore Database*), y permiten la **actualización en tiempo real de los datos**, lo que es fundamental para aplicaciones de este tipo. Además, las dos bases de datos están integradas dentro del ecosistema de *Firebase*, lo que facilita su utilización con otros servicios de la plataforma, como la autenticación y las notificaciones.

Las **principales diferencias** entre ambas radican en varias áreas clave. *Realtime Database* organiza la información en forma de un **árbol JSON**, mientras que *Firestore Database* emplea una estructura de datos basada en **colecciones y documentos**, lo que proporciona una organización más flexible y escalable. Respecto a las consultas, *Firestore Database* permite realizar búsquedas complejas y optimizadas a través de índices, lo que permite mayor eficiencia, mientras que *Realtime Database* dispone de un sistema más básico y limitado para este tipo de operaciones. En términos de actualización en tiempo real, aunque ambos servicios soportan la sincronización instantánea de datos, *Firestore Database* maneja de manera más eficiente conexiones en segundo plano y operaciones con grandes volúmenes de datos, aunque esto no es de gran importancia en este caso ya que la aplicación no va a llegar a un volumen de datos tan grande como para que esto sea un factor relevante. Además, en cuanto a la **capacidad de funcionar sin conexión**, *Firestore Database* ofrece un manejo más robusto de datos cuando el dispositivo está sin conexión, permitiendo que la aplicación se **sincronice automáticamente cuando se restablezca la conexión**. Por último, *Firestore Database* ofrece un control de acceso más detallado, permitiendo **reglas de seguridad específicas** para cada documento y colección, mientras que *Realtime Database* trabaja con **reglas de acceso más generales**.

Teniendo en cuenta todo esto, he considerado que *Firestore Database* era más adecuado para este tipo de aplicación debido a la flexibilidad que me ofrecía en el modelo de datos, la fácil organización de los mismos, y la mayor capacidad de realizar consultas y devolver datos de manera más rápida. Además, su capacidad de trabajar de manera eficiente en modo offline permitiendo que la aplicación siga funcionando sin conexión y se sincronice de manera automática cuando se restablezca la conexión me parecía de gran utilidad.

Ventajas de usar Firebase

- Servicios de autenticación, base de datos y notificaciones integradaos en una única plataforma.
- Servicio de autenticación ya implementado listo para integrarse directamente en la aplicación.
- Bases de datos flexibles y en tiempo real.
- Fácil integración con Flutter.

Aspecto	**Firestore DB**	**Realtime DB**
Arquitectura	Único árbol JSON	Colecciones y documentos
Consultas	Simples, sin índices complejos	Avanzadas con índices automát
Escalabilidad	Menor	Mejor rendimiento a gran esca
Sinc. en Tiempo Real	Menos eficiente	Más eficiente
Manejo de Datos Offline	Soporte básico	Sinc. automática al reconecta
Latencia	Mayor (con vol. grandes)	Menor
Seguridad	Reglas globales	Reglas por documento y colecc
Rendimiento general	Medio	Alto

Tabla 4.3: Firestore DB VS Realtime DB

- Permite trabajar con múltiples plataformas de desarrollo.
- Escalabilidad automática.

5. Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales

como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

5.1. De donde surge?

5.2. Metodologias

5.3. Formacion, libros, recursos...

5.4. Diagramas

6. Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

6.1. Artículos

6.2. Apps

- **Calm** [4] es una aplicación para dormir y meditar muy popular entre los usuarios. Incluye meditaciones guiadas de distintos temáticas, historias para dormir y música relajante. Está disponible para multiples plataformas como Android, IOS, web...
- **Headspace** [10] también tiene soporte multiplataforma, centrado en cursos de meditación aunque también incluye historias para ayudar a conciliar el sueño. Parte de la aplicación es bajo suscripción de pago.
- **Breathwrk** [3] ofrece múltiples ejercicios de respiración con una interfaz muy elaborada y visual. Además presenta al usuario los resultados de estos ejercicios con gráficas de progreso y logros. También ofrece al usuario crear o suscribirse a programas personalizados. Está disponible en múltiples plataformas.
- **RespiRelax+** [12] es una aplicación gratuita para móviles que ofrece la posibilidad de utilizarse offline. Respecto al contenido de la misma, ha desarrollado una interfaz intuitiva e inmersiva logrando muy buenos

resultados. Ofrece programas de respiración según la experiencia del usuario, aunque también permite personalizarlos en su duración y efectos visuales y sonoros.

- **Insight Timer** [11] no está enfocada en ejercicios propios de respiración, sino que es más como una plataforma con múltiples contenidos sobre meditación. Música, historias para dormir incluso eventos en vivo sobre referentes del mindfulness ayudarán al usuario a obtener los beneficios de la relajación.

- **Prana Breath** [1] es la aplicación más similar a lo que se ha intentado desarrollar en *BreathBank*. Se centra principalmente en ejercicios de respiración guiados con una interfaz muy interesante. Estos ejercicios son muy personalizables por parte del usuario, incluso en modo dinámico lo que permite variar la estructura del ejercicio una vez ya iniciado. También presenta el progreso con múltiples estadísticas, y ofrece la posibilidad de crear alarmas y recordatorios para una experiencia más adaptada.

7. Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Conclusiones

Lineas de trabajo futuras

Añadir idiomas, interfaz oscuro/claro, nuevos ejercicios de respiración, más metodos de acceso/registro de cuentas, dispositivos de medición, música relajante, podcast de relajacion, alarmas/recordatorios

Bibliografía

[1] Abdula. Prana breath: Respiración consciente. https://play.google.com/store/apps/details?id=com.abdula.pranabreath&hl=es_419, 2025. Disponible en Google Play, consultado el 11 de abril de 2025.

[2] Asana. ¿qué es scrum? https://asana.com/es/resources/what-is-scrum, 2024. Consultado el 24 de abril de 2025.

[3] Breathwrk Inc. Breathwrk: Ejercicios de respiración. https://play.google.com/store/apps/details?id=com.breathwrk.android, 2025. Disponible en Google Play, consultado el 11 de abril de 2025.

[4] Calm. Calm: Meditación y sueño. https://play.google.com/store/apps/details?id=com.calm.android&hl=es_419, 2025. Accedido el 11 de abril de 2025.

[5] Fortinet. ¿qué es una inyección sql (sql injection)? https://www.fortinet.com/lat/resources/cyberglossary/sql-injection, n.d. Accedido el 30 de abril de 2025.

[6] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley, 2012.

[7] Google. Dart. https://dart.dev/, 2025. Último acceso: 29 de abril de 2025.

[8] Google. Firebase. https://firebase.google.com/?hl=es-419, 2025. Último acceso: 29 de abril de 2025.

[9] Google. Flutter. https://flutter.dev/, 2025. Último acceso: 29 de abril de 2025.

[10] Headspace Inc. Headspace: Meditación y sueño. https://play.google.com/store/apps/details?id=com.getsomeheadspace.android&hl=es_419, 2025. Disponible en Google Play Store.

[11] Insight Network Inc. Insight timer - meditation app. https://play.google.com/store/apps/details?id=com.spotlightsix.zentimerlite2&hl=es, 2025. Accedido: 2025-04-14.

[12] Thermes d'Allevard. Respirelax+. https://play.google.com/store/apps/details?id=com.thermesallevard.respi_relax&hl=es, 2025. Disponible en Google Play, consultado el 11 de abril de 2025.

[13] Thorben. Solid design principles explained: The open/closed principle with code examples. https://stackify.com/solid-design-open-closed-principle/, 2018. Accedido: 2025-04-14.

[14] Thorben. Solid design principles explained: The dependency inversion principle with code examples. https://stackify.com/dependency-inversion-principle/, 2023. Accedido: 2025-04-14.

[15] Thorben. Solid design principles explained: The single responsibility principle. https://stackify.com/solid-design-principles/, 2023. Accedido: 2025-04-14.

[16] Visual Studio Code. Debugging in visual studio code. https://code.visualstudio.com/docs/debugtest/debugging, n.d. Accessed: 2025-04-29.