MEMORIA DEL PROYECTO

Habitr - Aplicación para fomentar los hábitos saludables.

Trabajo de Fin de Grado

Ingeniería Informática

VNiVERSiDAD D SALAMANCA

Junio 2023

Autor:

Jorge Cruz García

Tutores:

André Filipe Sales Mendes

Gabriel Villarubia González

Certificado de los tutores TFG

D. André Sales Mendes, D. Gabriel Villarubia González, profesores del Departamento de Informática y Automática de la Universidad de Salamanca.

HACEN CONSTAR:

Que el trabajo titulado "Habitr - Aplicación para fomentar los hábitos saludables.", que se presenta, ha sido realizado por Jorge Cruz García, con DNI 21740040A y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Ingeniería Informática en esta Universidad.

Salamanca, 4 de Julio de 2023

Fdo.:

Firmado por SALES MENDES ANDRE FILIPE - ****1546* el día 04/07/2023 con un certificado emitido por AC FNMT Usuarios

VILLARRUBIA GONZALEZ GABRIEL - 71932845T

Firmado digitalmente por VILLARRUBIA GONZALEZ GABRIEL - 71932845T
Fecha: 2023.07.04 13:58:51 +02'00'

TABLA DE CONTENIDOS

1 RESUMEN

El proyecto realizado como Trabajo de Fin de Grado consiste en el diseño y desarrollo de una aplicación que ayude a sus usuarios a desarrollar rutinas saludables, usando la gamificación como aspecto diferenciante con otras aplicaciones que intentan conseguir el mismo propósito.

La aplicación permite a sus usuarios compartir sus rutinas con otra gente, hacer amigos y en general compartir la experiencia a través de una plataforma social.

A pesar de que inicialmente se ideó como una aplicación destinada a gente que quisiese mejorar su salud mental por razones personales, el factor de la gamificación hace que tenga una audiencia más general.

Para que el proyecto sea robusto, esté poco acoplado, y sea de fácil mantenimiento se han puesto en práctica estrategias de planificación y patrones de código aprendidos en la asignatura de Ingeniería del Software.

También se ha llevado a cabo la realización de la documentación necesaria para entender cómo se ha abordado el desarrollo del proyecto, con todas las tareas hechas y las herramientas utilizadas.

Palabras clave: Aplicación, salud mental, bienestar, rutina, gamificación, Flutter

2 SUMMARY

This project has been made as an End Grade Project. It consists of the design and development of an app that helps its users to pick up healthy routines, using gamification as the differential aspect between this app and others that try to achieve the same goal.

The app lets its users share routines, make friends and in general share the experience through a social platform.

Even though it was initially conceived as an app that helps people who tried to make their mental health better, gamification made the potential userbase more general.

To make the project more robust, less coupled and easy to maintain, we have used planning strategies and code patterns previously learnt in the Software Engineering subject.

The required documentation to understand how development of the project happened has been made, explaining all the tasks that have been done and all the tools that have been used.

KEY WORDS: App, mental health, wellbeing, routine, gamification, Flutter

3 INTRODUCCIÓN

En el mundo actual, y especialmente tras la situación vivida a escala global durante la pandemia del COVID-19, se está empezando a dar más importancia que nunca a la salud mental.

Está demostrado científicamente que tener rutinas saludables contribuye directamente a nuestro bienestar físico y mental (Brennan & WebMD, 2021). Sin embargo, durante la pandemia las rutinas diarias que todos teníamos se vieron interrumpidas forzosamente, llevando a una pérdida de una de las mayores herramientas que tenemos a la hora de mantener nuestro bienestar.

A pesar de que la situación de emergencia sanitaria ya ha sido subsanada, el impacto es visible, habiendo causado un gran pico de diagnósticos de cuadros de ansiedad y depresión en la población mundial. (OMS, 2022)

Este proyecto nace motivado por los hechos explicados previamente. Su objetivo es, por tanto, ayudar a las personas a recuperar una vida diaria saludable tras la disrupción causada en los años previos. El objetivo es mejorar el bienestar y la calidad de vida de las personas a través de la motivación diaria a mantener rutinas saludables. Se intenta dar dicha motivación a través de varios métodos: la gamificación, un factor social y recordatorios motivantes.

Otra de las motivaciones principales de la aplicación es el funcionamiento de las aplicaciones que pretenden conseguir el mismo propósito: muchas son de pago, o coleccionan información de índole personal sobre sus usuarios, que posteriormente es vendida para anunciantes. Estas prácticas ponen por delante los beneficios propios sobre el bienestar real de sus usuarios, y por tanto otro de los objetivos del proyecto es hacer una aplicación que sea gratuita y que obtenga la cantidad mínima posible de datos de sus usuarios.

En este documento se explican los aspectos más relevantes del desarrollo del proyecto. La estructura del documento es la siguiente:

- **Estado del arte y comparativa:** se mostrarán las aplicaciones actualmente disponibles, analizando sus ventajas y desventajas.
- **Objetivos:** se especificarán los objetivos que se intentan conseguir con la realización del proyecto.
- **Conceptos teóricos:** se explicarán los conceptos teóricos necesarios para la comprensión del trabajo.
- **Técnicas y herramientas:** se hará una documentación de todas las técnicas y herramientas usadas en el desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** se detallarán los aspectos más complejos del desarrollo de la aplicación.
- **Limitaciones de la aplicación:** se hablan de las limitaciones técnicas y temporales encontradas en la realización del proyecto.

- **Conclusiones y líneas de trabajo futuras:** se especifican las conclusiones obtenidas tras la realización del proyecto, y potenciales formas de continuar la investigación en este campo
- **Bibliografía**

Además, el documento está complementado con los siguientes anexos:

- **Anexo I – *Planificación temporal*:** explicación de la planificación temporal de las tareas del proyecto y los aspectos considerados para realizarla.
- **Anexo II – *Especificación de requisitos*:** expone la especificación de requisitos del sistema.
- **Anexo III – *Análisis de requisitos:*** documento que explica la fase de análisis de los requisitos.
- **Anexo IV – *Diseño del software:*** explicación de la documentación de la fase de diseño del sistema.
- **Anexo V – *Documentación técnica:*** documento que facilita la comprensión del código.
- **Anexo VI – *Manual de usuario:*** documento que explica detalladamente la funcionalidad de todos los sistemas de la aplicación para que un usuario pueda usarla.

4 ESTADO DEL ARTE

Actualmente existen muchas aplicaciones que intentan cumplir el mismo propósito que nuestra aplicación. A continuación, se explicarán algunas aplicaciones alternativas, su diseño, sus ventajas y desventajas.

4.1 APLICACIONES DE ESTABLECIMIENTO DE RUTINAS

4.1.1 Routinery

Routinery es una aplicación de establecimiento de rutinas desarrollada por la empresa coreana *Routinery Corp*. La aplicación destaca por su facilidad de uso y lo guíado que está todo, a través de tutoriales de uso y explicaciones interactivas. La aplicación permite empezar con rutinas pre-creadas por otros usuarios y los creadores de la aplicación, y usa un sistema de niveles para representar el progreso.

A pesar de todo, su mayor desventaja, que es la misma que en muchas otras aplicaciones es el modelo de negocio; sólo podemos añadir un número de rutinas muy limitado antes de que nos intenten vender la versión pagada, que cuesta 32 euros al año.

En la *figura 1*, se puede ver la interfaz de Routinery

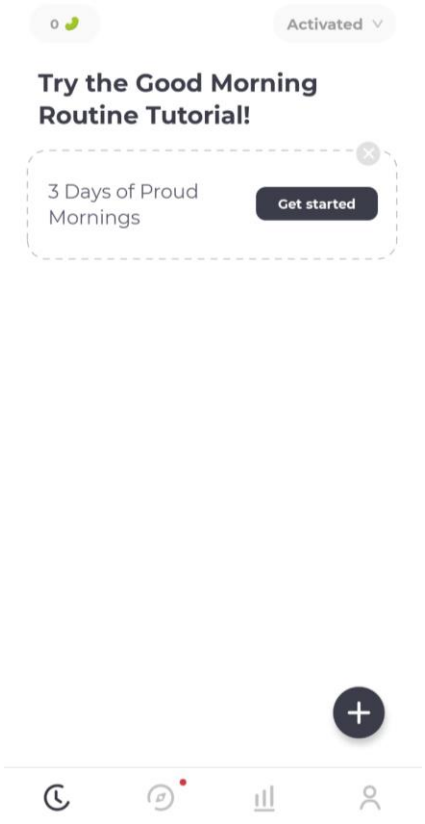

Figura 1: Interfaz Routinery

4.1.2 Proddy

Proddy es una aplicación desarrollada por el programador alemán *Thomas Wolfgang Menzel*. Se intenta diferenciar del resto de aplicaciones del mismo tipo añadiendo funcionalidades extra, cómo son una motivación por las mañanas y un tiempo de reflexión por las noches, además de lecciones sobre cómo funciona el cerebro humano y la creación de hábitos. Hace hincapié en la facilidad de uso y la motivación de sus usuarios.

A nivel de monetización, colecciona datos personales que son vendidos por Google, y su versión premium cuesta 10 euros. No está tan limitada como Routinery.

En la *figura 2* se puede ver la interfaz principal de Proddy

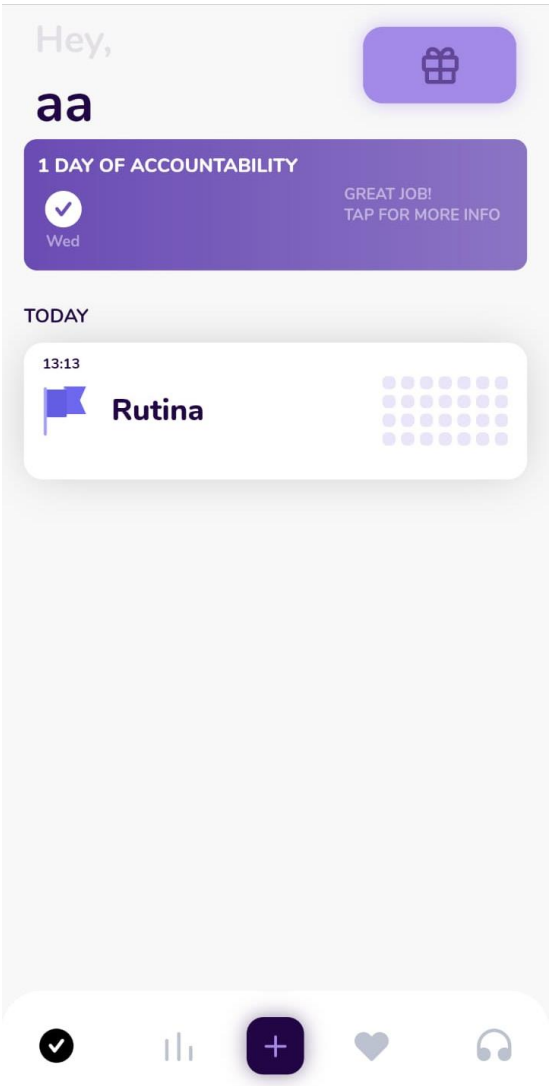

Figura 2: Interfaz Proddy

4.1.3 Productive

Productive es una aplicación que también intenta cumplir este propósito. Pone énfasis en el aspecto social creando *retos* en los que podemos tomar parte con un gran número de otros usuarios. Además, ofrece motivación al despertar y un tiempo para reflexionar por la noche.

La monetización de esta aplicación consiste en una subscripción de 16 euros por año, que desbloquea la mayor parte de la aplicación.

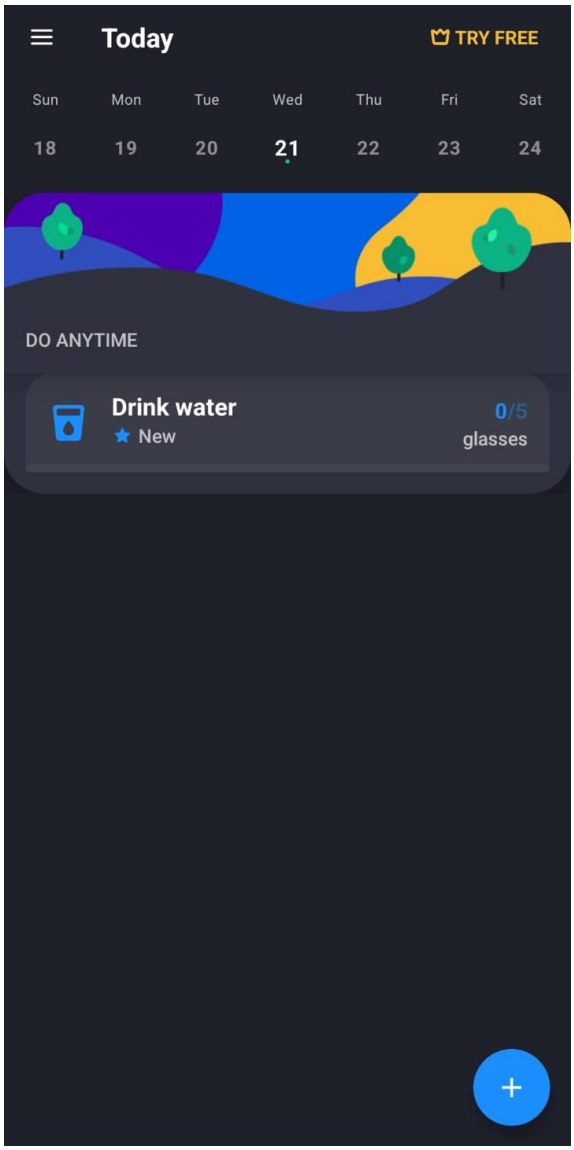

Figura 3: Interfaz Productive

4.1.4 HabitNow

HabitNow es una aplicación creada por el desarrollador argentino *Agustín Ezequiel García Rampini.* Es algo más simple que el resto de aplicaciones que hemos explicado

previamente, y es probablemente la que más se parece a la nuestra a nivel de funcionalidad. Permite programar nuestras rutinas, ofreciéndonos notificaciones y una vista de calendario.

Su monetización es un solo pago de 5.50 euros, que desbloquea la opción de crear todas las rutinas que se quieran, además de no vender datos personales.

En la *figura 3* se puede ver la interfaz de HabitNow

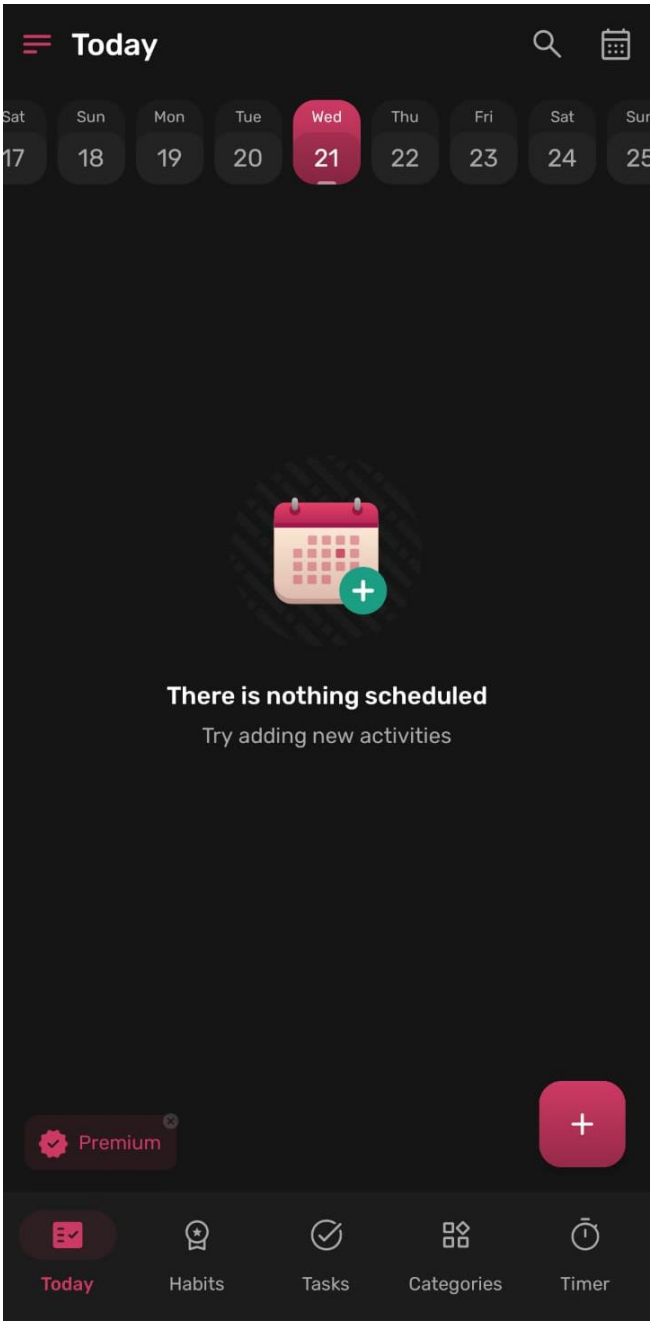

Figura 4: Interfaz HabitNow

5 OBJETIVOS

En este apartado se expondrán los objetivos que debe cumplir la aplicación para llevar a cabo el desarrollo del Trabajo de Fin de Grado. Se detallarán también los objetivos personales que se pretenden cumplir con el desarrollo del proyecto.

5.1 OBJETIVOS DEL SISTEMA

El objetivo principal es el diseño y construcción de una aplicación que facilite la obtención de rutinas saludables a través de la gamificación y otros sistemas que ayudan a mantener una interacción sostenida en el tiempo con el usuario como notificaciones.

El siguiente listado describe los objetivos principales que querremos implementar en el sistema:

- **Seguimiento de rutinas:** El sistema deberá ser capaz de realizar un seguimiento de las rutinas que cumple cada usuario.
- **Gamificación:** El sistema deberá ser capaz de representar el progreso del usuario a través de la gamificación.
- **Notificaciones:** El sistema deberá permitir presentar notificaciones al usuario en forma de recordatorio de las rutinas.
- **Estadísticas:** El sistema deberá ser capaz de presentar una serie de estadísticas personales relativas a cada usuario y sus rutinas.
- **Funcionalidad social:** El sistema deberá permitir la interacción social entre distintos usuarios que la usen.

5.2 OBJETIVOS PERSONALES

A continuación, describiré los objetivos principales que me llevaron a querer desarrollar este proyecto de forma personal.

Por un lado, la principal razón que me motivó a desarrollar este proyecto fue la crisis sanitaria causada durante la pandemia de la COVID-19. Esta situación extraordinaria tuvo consecuencias en la salud mental de toda la población humana, y se cebó especialmente con ciertos colectivos, como por ejemplo el de los jóvenes. A nivel personal noté el cambio en mí mismo y en muchos de mis amigos, y esto me llevó a buscar alguna idea que pudiese ayudarles.

Sabiendo que las rutinas saludables contribuyen al bienestar mental de forma directa, quise poner mi granito de arena e intentar ayudar al máximo número de personas posible a sobrellevar los efectos de la situación previamente explicada y volver a establecer unas rutinas diarias tras un período en el cual muchas personas perdimos la noción de tiempo o de rutina.

Por el otro lado, siempre he estado interesado en el desarrollo *fullstack*, y esto presentó una gran ocasión para desarrollar una aplicación entera por mi cuenta.

Finalmente, al ser un proyecto que abarca tantas cosas, permitía poner en prueba mi conocimiento de muchas de las asignaturas que he estudiado durante la carrera con un proyecto real.

6 CONCEPTOS TEÓRICOS

En este apartado se exponen los conceptos teóricos que necesitan ser comprendidos para conseguir un entendimiento del trabajo.

6.1 FRONTEND, BACKEND Y FULLSTACK

Frontend, backend y fullstack son términos que definen distintos tipos de desarrolladores del software.

El *desarrollador frontend* se encarga la programación de las interfaces gráficas que ve nuestro usuario, el *desarrollador backend* tiene como misión el desarrollo de la lógica interna del programa y el *desarrollador fullstack* realiza un desarrollo mixto de ambos ámbitos.

En el ámbito de este trabajo, al haber un solo programador que se ha encargado de toda la aplicación, se podría decir que durante su desarrollo se ha obtenido experiencia de *fullstack*.

6.2 GAMIFICACIÓN

La *gamificación*, o *ludificación* es, según la RAE: "Aplicar técnicas o dinámicas propias del juego a actividades o entornos no recreativos para potenciar la motivación y la participación, o facilitar el aprendizaje y la consecución de objetivos."

En otras palabras, consiste aplicar mecánicas o diseños basados en los juegos a entornos distintos con el objetivo de cambiar el comportamiento de las personas que están en dichos entornos.

En el contexto de este proyecto, se cogen varios elementos claves de los juegos (un sistema de progresión, existencia de un factor social, personalización visual) para fomentar el uso continuado y la motivación de participar en la creación de rutinas de la aplicación.

7 TÉCNICAS Y HERRAMIENTAS

En este apartado se explicarán las distintas técnicas metodológicas del diseño, además de las herramientas software utilizadas durante el desarrollo del proyecto.

7.1 TÉCNICAS Y HERRAMIENTAS USADAS EN EL BACKEND

7.1.1 SQL

SQL (Structured Query Language) es un lenguaje de programación diseñado para gestionar y manipular bases de datos relacionales. SQL permite definir la estructura de una base de datos, crear tablas, insertar, actualizar y eliminar datos, además de recuperar información a través de un sistema de consultas y filtros. SQL ofrece operaciones y funciones para realizar operaciones de agrupamiento y ordenamiento, y también para establecer relaciones entre diferentes tablas en una base de datos.

7.1.2 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto y gratuito. También conocido como Postgres, destaca por su alta escalabilidad, su rendimiento y su capacidad para manejar grandes volúmenes de datos. PostgreSQL es compatible con el estándar SQL y ofrece características avanzadas, como **la seguridad a nivel de fila (RLS), vistas, funciones y desencadenadores**. PostgreSQL es el lenguaje utilizado por **Supabase**, y a través del cual realizaremos todas nuestras operaciones que requieran crear, modificar o recuperar datos de nuestra base de datos.

7.1.3 Supabase

Supabase es una plataforma de código abierto que combina una base de PostgreSQL en la nube con una interfaz API que hace muy fácil su integración con aplicaciones móviles, ayudando a un desarrollo más rápido y eficiente. La plataforma se define a sí misma como una “alternativa de código abierto a Firebase”, e intenta dar las mismas funcionalidades que nos aportaría Firebase. Entre las funcionalidades a destacar por su uso en el proyecto tenemos:

- **Base de datos PostgreSQL:** Usada para almacenar y recuperar los datos persistentes usados por nuestra aplicación.
- **Sistema de seguridad a través de Row-Level Safety (RLS):** evita que los usuarios puedan insertar o modificar datos para los cuales no tienen permisos.
- **Sistema de autenticación y registro:** Supabase se encarga de la autenticación por nosotros, además de la gestión de correos de recuperación de contraseña y de verificación.
- **Triggers:** O desencadenadores, son funciones que se ejecutan cuando se cumplen ciertas condiciones en una operación de base de datos (por ejemplo, permiten ejecutar una función cada vez que se inserta un dato en cierta tabla). Se utilizaron ampliamente en el proyecto para validar los datos desde el lado de la base de datos.

Sin embargo, además de las que han sido usadas, Supabase nos ofrece muchas otras funcionalidades: **guardado de archivos, funciones *Edge*, base de datos a tiempo real…**

En la *figura 5* se puede observar la interfaz web de Supabase, que permite operar directamente con la base de datos, generar tablas, ver logs, modificar los permisos y la seguridad de la BBDD a través de una interfaz intuitiva.

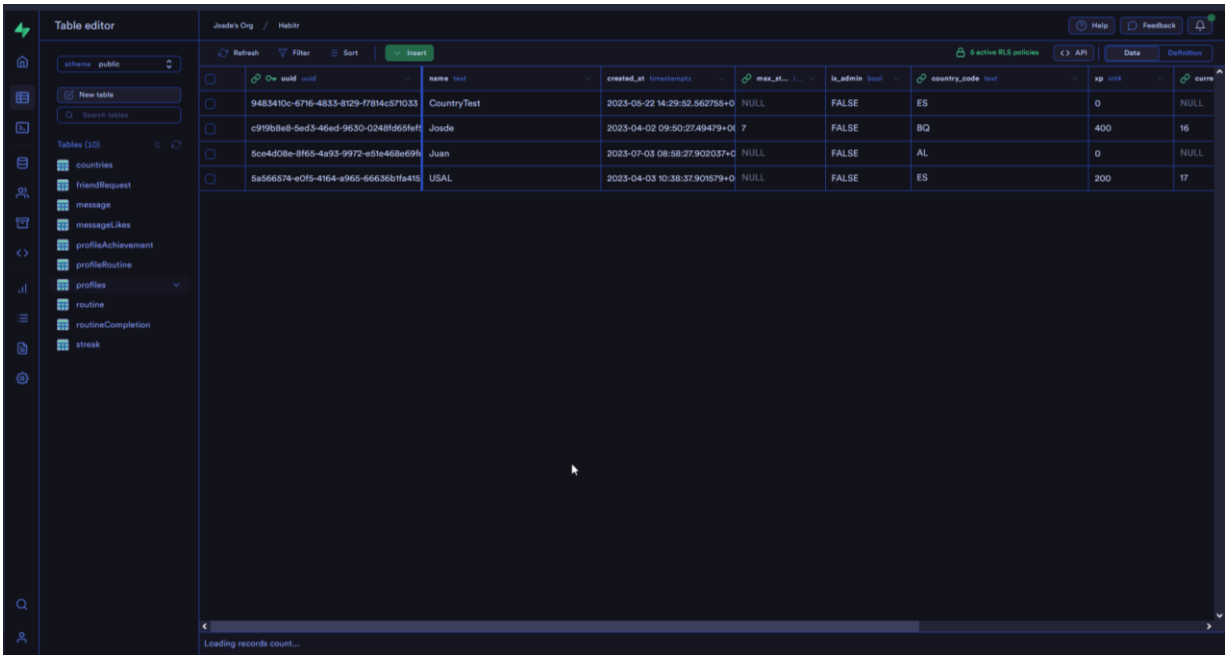

Figura 5: Interfaz de Supabase

7.1.4 Row-level Safety (RLS)

La RLS, o en español, **seguridad a nivel de fila**, es una funcionalidad avanzada de PostgreSQL que permite limitar el acceso a los datos que puede obtener un usuario en una BBDD usando *políticas de seguridad.*

RLS permite limitar el acceso estableciendo funciones booleanas de SQL para cada operación sobre una tabla, de manera que sólo se pueda usar esa operación sobre las filas que devuelvan verdadero en la función.

Fue utilizada ampliamente en el desarrollo de la aplicación, ya que al usar Supabase de forma directa, todos los usuarios parten de el mismo token de autenticación, los mismos permisos y las mismas tablas y, sin embargo, sería ridículo que un usuario pudiese hacer publicaciones en nombre de otro usuario.

Un ejemplo práctico sería, por ejemplo, una política que diga “un usuario sólo puede insertar mensajes cuyo atributo creator_id sea igual a su id de usuario”, que sería la que arreglase la situación planteada anteriormente.

7.2 TÉCNICAS Y HERRAMIENTAS USADAS EN LA APLICACIÓN

7.2.1 Flutter

Flutter es un *framework* de desarrollo de aplicaciones móviles de código abierto creado por Google. Flutter tiene como objetivo principal mantener un nivel alto de rendimiento a la vez que permite compilar tanto para Android como para iOS con un mismo código.

La característica principal de Flutter es su enfoque en la creación de interfaces de usuario mediante el uso de *widgets* personalizados. Los widgets son elementos de la interfaz de usuario, como botones, campos de texto o listas, que se pueden combinar y personalizar para construir interfaces complejas.

Además, Flutter utiliza el lenguaje de programación Dart, que también fue desarrollado por Google, como su lenguaje principal. Dart es un lenguaje moderno y eficiente que combina la facilidad de uso de lenguajes como JavaScript con el rendimiento y la seguridad de lenguajes de bajo nivel.

Finalmente, cabe destacar que tanto Dart como Flutter tienen una comunidad de desarrollo muy activa, que ha creado librerías muy útiles para el desarrollo del proyecto tales cómo la integración con el patrón BLoC o la integración con la API de Supabase.

7.2.2 Librería flutter_bloc

La librería flutter_bloc nos ayuda a implementar de forma rápida el patrón BLoC en nuestras aplicaciones. Esta librería trae una clase genérica de Bloc<TipoEvento, TipoEstado> que nos permite definir de forma fácil cuales son los estados que tendrá nuestro BLoC y cuáles son los eventos ante los cuales ha de responder y de qué forma.

Además de esto, incluye métodos para poder propagar los BLoCs a lo largo de nuestra aplicación, de manera que todas las pantallas que requieran acceder a la lógica de un componente puedan hacerlo a través del mismo BLoC, minimizando el impacto de rendimiento que puede llevar crear este tipo de clases.

7.2.3 C#

C# es un lenguaje de programación de propósito general y orientación a objetos desarrollado por Microsoft.

C# es conocido por su versatilidad y se utiliza ampliamente en el desarrollo de una variedad de aplicaciones, desde aplicaciones de escritorio hasta aplicaciones móviles y juegos. Proporciona un conjunto sólido de características y bibliotecas que facilitan la escritura de código.

En el contexto de este proyecto, C# fue usado para desarrollar el código del juego, puesto que este lenguaje integra de forma directa con el motor de videojuegos Unity.

7.2.4 Visual Studio Code

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft. Es una herramienta de desarrollo altamente popular y ampliamente utilizada, conocida por su diseño ligero, su amplia gama de extensiones y su gran capacidad de personalización.

Una de las características más destacadas de Visual Studio Code es su ecosistema de extensiones. Se pueden agregar extensiones desde el marketplace de VS Code para personalizar y ampliar las capacidades del editor según sus necesidades. Estas extensiones pueden agregar nuevas características, herramientas específicas para lenguajes de programación, integración con servicios en la nube, temas visuales y más.

En el contexto del proyecto, se utilizaron varias extensiones de Dart y Flutter que para refactorizar el código para hacerlo más legible o estándar dentro del mismo proyecto, además de extensiones que permiten crear directamente esqueletos de código para componentes que son parecidos (pantallas, BLoC…)

7.2.5 Android Studio

Android Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) creado por Google para el desarrollo de aplicaciones móviles para el sistema operativo Android.

Android Studio se basa en el popular IDE IntelliJ IDEA, conocido como uno de los IDEs de Java más completos, que ha sido adaptado para la plataforma Android. Proporciona una amplia gama de características y herramientas que facilitan el desarrollo de aplicaciones móviles de alta.

Especialmente, de todas las características que nos proporciona Android Studio, para el desarrollo del proyecto fue especialmente útil para crear y ejecutar emuladores de Android, con distintas versiones del sistema operativo, tamaños, y especificaciones de hardware, que permitió probar el proyecto en distintas características sin necesidad de recurrir a una variedad de dispositivos físicos, lo cual se hacía inalcanzable por su precio.

7.2.6 Unity

Unity es un motor de desarrollo de videojuegos multiplataforma ampliamente utilizado. Permite crear videojuegos en 2D y 3D, además de experiencias en realidad aumentada.

Unity ofrece un editor visual de escenas, objetos de juego prefabricados y assets que permite hacer todo lo relativo al desarrollo de videojuegos, excepto la programación, de forma intuitiva y sin entrenamiento previo.

Otra de las razones por las que Unity fue elegido fue por su capacidad de compilar una misma escena a distintos lenguajes y arquitecturas a través de su característica de compilación a C++ (IL2CPP), que permite integrarlo directamente con Flutter.

Finalmente, Unity tiene una comunidad muy activa de desarrolladores, lo cual fue de tremenda ayuda a la hora de desarrollar la parte del juego puesto que hay disponibles muchas guías, dudas resueltas y assets en sus foros y páginas oficiales.

En la *figura 6* se puede observar la interfaz del editor de Unity, enseñando la escena del juego desarrollado.

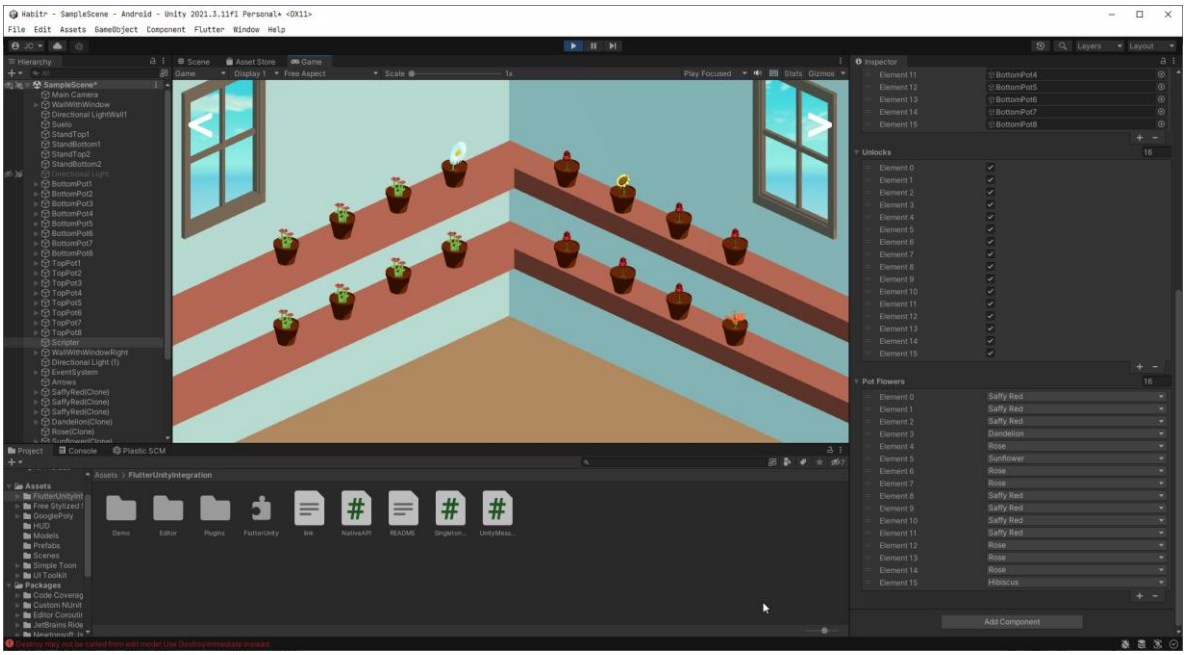

Figura 6: Interfaz de Unity

7.2.7 Google Poly

Google Poly fue un sitio web creado por Google que permitía descargar modelos 3D de forma gratuita para uso no comercial.

A pesar de que Google Poly cerró de forma definitiva en 2021, los usuarios de la comunidad de código abierto archivaron los contenidos de la página web para el uso de todos los modelos bajo la licencia CC-BY 4.0 (uso gratuito con crédito).

Los modelos que ofrecía Poly fueron utilizados durante el desarrollo del videojuego, y los créditos de los modelos utilizados se pueden encontrar en el fichero "*CREDITS.md"* de la carpeta que contiene los archivos fuente.

7.2.8 Blender

Blender es un software de modelado 3D de código abierto, que permite la creación, modificación y renderizado de modelos y escenas en 3D.

Durante el desarrollo se utilizó para ajustar algunos de los modelos que aparecen en el juego final.

7.3 HERRAMIENTAS CASE

7.3.1 Visual Paradigm

Visual Paradigm es una suite de software de modelado visual y diseño de software que proporciona herramientas para el análisis, diseño y visualización de sistemas y software.

Visual Paradigm se especializa en el desarrollo de diagramas de UML (Unified Modeling Language), que es el lenguaje por excelencia a la hora de realizar un proyecto de Ingeniería del Software.

Visual Paradigm fue usado ampliamente para crear los diagramas que se pueden ver en los distintos anexos de esta memoria, y fueron usados para facilitar el modelado del proyecto final.

7.3.2 Microsoft Project

Microsoft Project es una herramienta de gestión de proyectos desarrollada por Microsoft. Está diseñada para ayudar a los profesionales y equipos a planificar y estimar el tiempo de proyectos, además de para hacer un seguimiento del desarrollo de los mismos o planificar la gestión de recursos.

En este proyecto, se utilizó para desarrollar una planificación temporal basada en estimaciones, que se puede ver en el *Anexo I – Planificación temporal* que se encuentra adjunto a esta memoria.

7.4 HERRAMIENTAS DE CONTROL DE VERSIONES

7.4.1 Git

Git es un sistema de control de versiones de software y que fue desarrollado por Linus Torvalds en con el objetivo de proporcionar un sistema de seguimiento de cambios eficiente, confiable y flexible para proyectos de software.

A lo largo del desarrollo de este proyecto, Git ha sido una herramienta con un valor inexplicable. Es de una gran utilidad poder ver el historial de cada fichero, para poder entender cómo se han podido introducir regresiones o errores en ciertas funcionalidades, cómo para poder recuperar código viejo o tener una copia de seguridad en caso de pérdida.

7.4.2 GitHub

GitHub es una plataforma de alojamiento y control de versiones basada en Git que pertenece a Microsoft.

GitHub proporciona un entorno en línea para almacenar proyectos de software y mantener un control de sus versiones de forma cómoda, a través de múltiples plataformas y siempre accesible a través de internet.

Al igual que Git, fue de tremenda ayuda tener esta herramienta disponible, puesto que creaba la posibilidad de ser capaz de consultar el código de la aplicación desde cualquier lugar del mundo de forma casi instantánea, además de tener una copia de seguridad fiable de todos los archivos fuente y todos los cambios que recibieron a lo largo del desarrollo.

7.5 HERRAMIENTAS DE DISEÑO

7.5.1 Adobe XD

Adobe XD es una herramienta de diseño vectorial que también trae capacidades para el prototipado de interfaces de usuario. Se utilizó para crear un primer esbozo de la interfaz de la aplicación, que fue usado como base a la hora de implementar la interfaz en Flutter.

En la *figura 7* se puede observar la interfaz de Adobe XD, junto a un prototipo de algunas de las pantallas que se ven en la aplicación final, como son la del perfil, los amigos y las estadísticas.

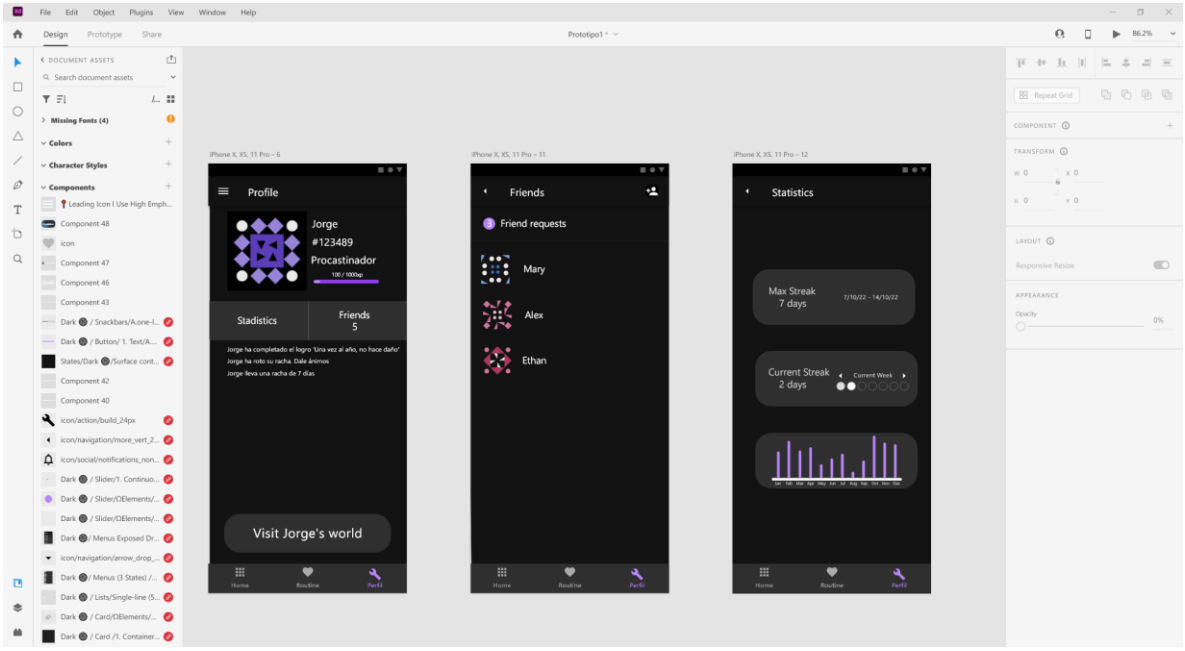

Figura 7: Interfaz de Adobe XD

7.5.2 Dbdiagram.io

DbDiagram es una herramienta que permite definir bases de datos a través de un lenguaje personalizado llamado DBML (Database Markup Language) para obtener diagramas de entidad relación.

Es de especial utilidad, puesto que los ficheros de DBML que se crean para obtener las gráficas posteriormente se pueden "traducir" de forma automática a ficheros de SQL, PostgreSQL o incluso generar documentaciones completas sobre la base de datos sólo con ese fichero DBML.

En el proyecto se utilizó para crear el diagrama entidad-relación que se puede ver en el *Anexo IV* y que representa de forma fiel la base de datos final que se utilizó.

En la *figura 8* se puede ver un ejemplo del lenguaje DBML y su diagrama resultante.

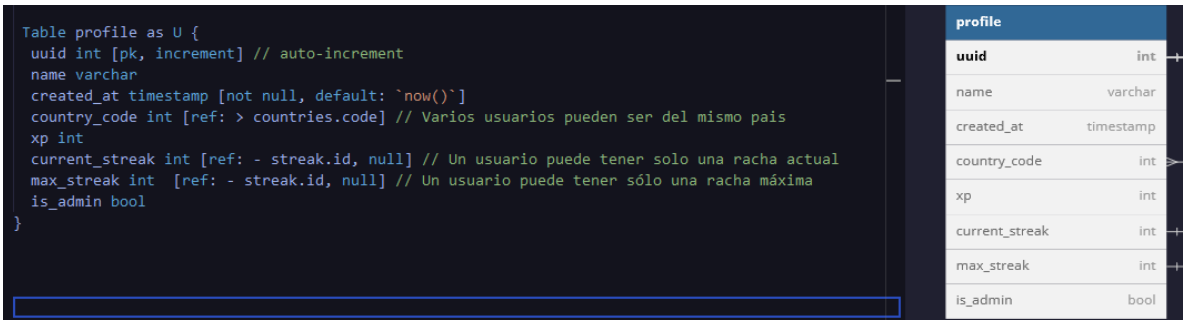

Figura 8: DBML y su diagrama resultante

8 ASPECTOS RELEVANTES DEL DESARROLLO

8.1 MARCO DEL TRABAJO

Este proyecto ha sido desarrollado bajo el marco de trabajo del Proceso Unificado. El Proceso Unificado está caracterizado por estar dirigido por casos de uso y ser iterativo e incremental.

Una definición más extensa de las características del Proceso Unificado es la siguiente:

- **Iterativo e incremental:** El proceso unificado está compuesta por varias fases, siendo la de Inicio, Elaboración, Construcción y Transición. Cada una de estas fases a su vez se divide en distintas iteraciones.
- **Dirigido por los casos de uso:** Los casos de uso se utilizan para definir los requisitos funcionales que ha de tener nuestra aplicación. Cada iteración debería de tomar un conjunto de casos de uso o escenarios y desarrollarlos a través de las distintas fases de una especificación de software: diseño, análisis, implementación y prueba.
- **Centrado en arquitectura:** En el proceso unificado existen distintos modelos que pueden cubrir todas las necesidades de nuestra aplicación. A lo largo del desarrollo, se irá refinando y concretando cual de estas arquitecturas se escogerá a partir de una propuesta inicial de arquitectura.

Como decíamos previamente, el proceso unificado está compuesto por distintas fases, que son las siguientes:

- **Inicio:** Se define un modelo de negocio y se realiza una elicitación de requisitos inicial.
- **Elaboración:** Se obtiene una visión refinada del proyecto que se ha de realizar, definiendo más en detalle los casos de uso y el modelo, y apareciendo nuevos requisitos y objetivos.
- **Construcción:** Se lleva a cabo el desarrollo del proyecto en código.
- **Transición:** Etapa final en la que el producto ha de estar preparado para su lanzamiento. Esta etapa se centra en el mantenimiento del software ya desarrollado.

En la *figura 9* podemos ver las fases e iteraciones del proceso unificado:

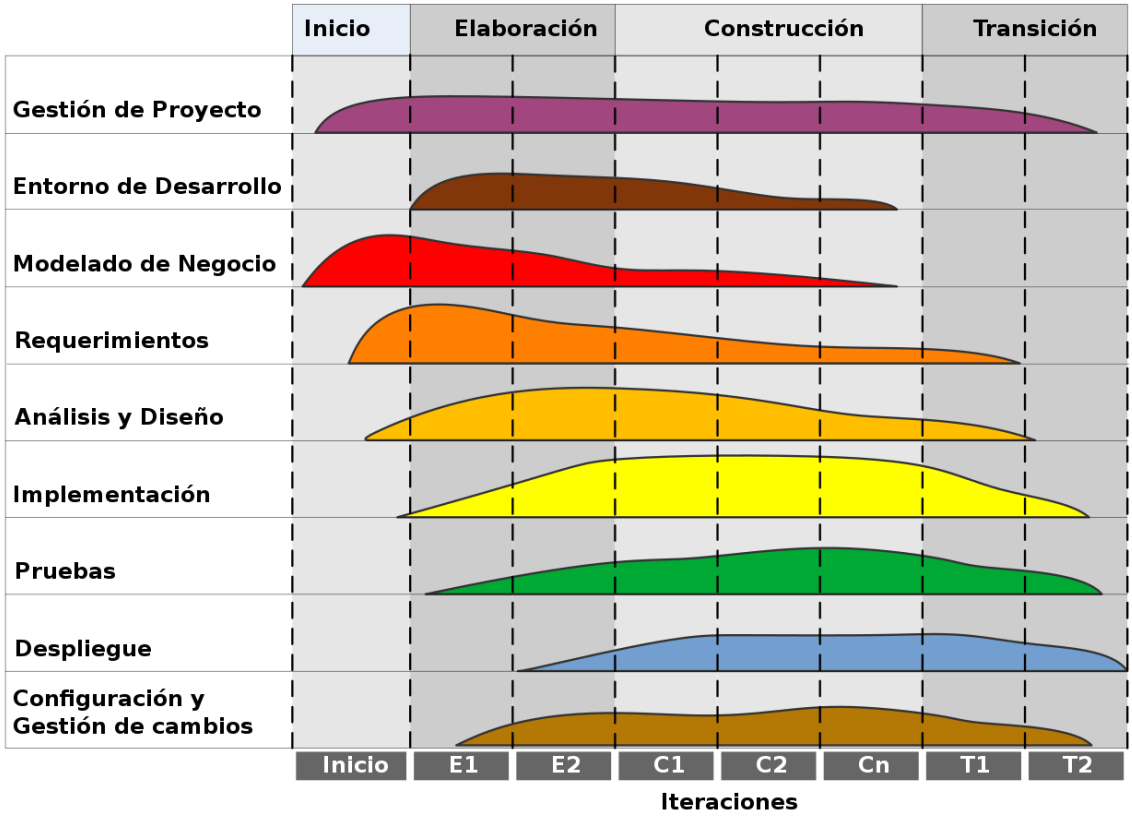

Figura 9: Visualización del Proceso Unificado

8.2 PLANIFICACIÓN TEMPORAL

Una de las primeras tareas que se realizó fue la planificación temporal, a través de la cual se pudo obtener una estimación del tiempo que llevará a cabo realizar el proyecto y de las tareas separadas en las que trabajaremos posteriormente.

Para obtener una vista detallada de la planificación temporal que se realizó, se puede ver en el documento suplementario *Anexo I – Planificación temporal*, que se incluye junto a esta memoria. En dicha planificación temporal se obtuvo el resultado de que se tardaría **116 días laborales** en desarrollar el proyecto entero.

Además de la planificación temporal, se realizó una estimación del esfuerzo usando el método de los **Use Case Points (UCP)** y el software EZEstimate, para comprobar que nuestra planificación fuera una estimación de tiempo realista, y se obtuvieron resultados muy parecidos, con un resultado de **140 días laborales, reafirmando que se tardarían unos 5-6 meses de tiempo real en desarrollar el proyecto.**

En la *figura 10* que se ven a continuación, se puede ver un ejemplo la separación en tareas, y el tiempo que ocuparía cada una de ellas dentro de nuestra planificación.

	Habitr	**116 days**	**Tue 10/01/23**	**Tue 20/06/23**	
	Especificación de requisitos	**12 days**	**Tue 10/01/23**	**Wed 25/01/23**	
	Fijar objetivos	3 days	Tue 10/01/23	Thu 12/01/23	
	Definición de requisitos funcionales	5 days	Fri 13/01/23	Thu 19/01/23	3
	Definición de requisitos no funcionales	3 days	Fri 13/01/23	Tue 17/01/23	3
	Realización del documento "Anexo II: especificación de requisitos"	5 days	Wed 18/01/23	Wed 25/01/23	5
	Hito: Fin de la especificación de requisitos	0 days	Wed 25/01/23	Wed 25/01/23	2
	Investigación	**10 days**	**Tue 10/01/23**	**Mon 23/01/23**	
	Investigación de la literatura académica	10 days	Tue 10/01/23	Mon 23/01/23	
	Comparación con aplicaciones parecidas	10 days	Tue 10/01/23	Mon 23/01/23	
	Hito: Fin investigación	0 days	Mon 23/01/23	Mon 23/01/23	8
	Análisis	**23 days**	**Thu 26/01/23**	**Mon 27/02/23**	**8;2**
	Modelo del dominio	5 days	Thu 26/01/23	Wed 01/02/23	
	Realización de casos de uso - Análisis	6 days	Thu 02/02/23	Thu 09/02/23	13
	Clases Análisis	3 days	Fri 10/02/23	Tue 14/02/23	14
	Arquitectura del modelo de análisis	2 days	Wed 15/02/23	Thu 16/02/23	15
	Realización del documento "Anexo III: Análisis de requisitos"	7 days	Fri 17/02/23	Mon 27/02/23	16
	Hito: Fin del análisis de requisitos	0 days	Mon 27/02/23	Mon 27/02/23	12

Figura 10: Ejemplo de planificación temporal del

8.3 ESPECIFICACIÓN DE REQUISITOS

Durante la fase de especificación de requisitos del proyecto, se ha esbozado un primer acercamiento de los objetivos que se quieren alcanzar con el proyecto, junto a los requisitos que tendrá que tener para alcanzar dichos objetivos.

Se ha realizado el catálogo de requisitos, que incluye la definición de los actores, requisitos funcionales, requisitos no funcionales, requisitos de información, y finalmente, de los casos de uso.

Para obtener más información sobre este proceso y la especificación resultante se puede consultar el documento suplementario *Anexo II – Especificación de requisitos del sistema*, que se incluye junto a esta memoria. Para la realización de este anexo, se utilizó la metodología explicada en *"Metodología para la Elicitación de Requisitos de Sistemas Software"* (Durán Toro & Bernárdez Jímenez, 2000).

A continuación, se presenta un ejemplo de cada etapa de la especificación de requisitos.

8.3.1 Participantes

Los participantes de este proyecto son:

- **Cruz García, Jorge**
- **Sales Mendes, André Filipe**
- **Villarubia González, Gabriel**

En la *tabla 1*, se puede ver un ejemplo de una especificación de participante:

Participante	**Cruz García, Jorge**
Organización	Universidad de Salamanca
Rol	Desarrollador, Programador
Es desarrollador	Sí
Es cliente	No
Es usuario	No
Comentarios	Ninguno

Tabla 1: Participante Cruz García, Jorge

8.3.2 Objetivos

Los objetivos funcionales que se han definido para la aplicación son:

- **Seguimiento de rutinas**
- **Gamificación**
- **Uso de notificaciones**
- **Servicio estadístico**
- **Funcionalidad social**
- **Alta, baja y visualización de rutinas**

En la *tabla 2* se presenta un ejemplo de una tabla de objetivos:

OBJ-001	**Seguimiento de rutinas**
Versión	1.0
Autores	Jorge Cruz García
Fuentes	• Sales Mendes, André Filipe • Villarubia González, Gabriel
Descripción	El sistema deberá ser capaz de realizar un seguimiento de las rutinas que cumple cada usuario.
Subobjetivos	Ninguno
Importancia	Vital
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta
Comentarios	Ninguno

Tabla 2: OBJ-001 Seguimiento de rutinas

8.3.3 Requisitos de información

Los requisitos de información definen la información que nuestro sistema ha de almacenar para poder realizar su funcionamiento correcto.

Los requisitos de información que se han identificado para el proyecto son los siguientes:

- **Información sobre usuarios**
- **Información sobre rutinas**
- **Información sobre las rutinas completadas**
- **Información sobre los mensajes sociales**
- **Información sobre las peticiones de amistad**

En la *tabla* 3 se presenta un ejemplo de una tabla de requisitos de información:

IRQ-001	**Información de usuarios**	
Versión	1.0	
Autores	Jorge Cruz García	
Fuentes	• Sales Mendes, André • Villarubia González, Gabriel	
Dependencias	Ninguno	
Descripción	El sistema deberá almacenar la información correspondiente a las cuentas de usuario creadas. En concreto:	
Datos específicos	• Identificador • País • Email • Nivel	
Tiempo de vida	Medio	Máximo
Ocurrencias simultaneas	Medio	Máximo
Importancia	Importante	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 3: IRQ-001 Información de usuarios

8.3.4 Requisitos funcionales

Los requisitos funcionales definen cómo se ha de comportar el sistema ante ciertos escenarios recurrentes. Es decir, definen la funcionalidad de la aplicación.

En este apartado se ha realizado una separación de las funcionalidades en paquetes, la definición de los distintos actores que tiene nuestro sistema y finalmente se han definido los casos de uso en detalle.

En la *figura 11* podemos ver la separación en paquetes:

Figura 11: Diagrama de paquetes

Los actores que definimos fueron los siguientes:

- **Usuario no logueado**
- **Usuario logueado**
- **Sistema**

En la *tabla 4* se muestra un ejemplo de una definición de actor:

ACT-001	**Usuario no logueado**
Versión	1.0
Autores	Cruz García, Jorge
Fuentes	• Sales Mendes, André Filipe • Villarubia González, Gabriel
Descripción	Este actor representa a un usuario que aún no se ha autenticado, para que pueda loguearse.
Comentarios	Ninguno

Tabla 4: ACT-001 Usuario no logueado

Tras definir los actores de nuestro sistema, se ha procedido a dividir todas las funcionalidades que ha de tener nuestra aplicación para cumplir sus objetivos en casos de uso, y se han realizado diagramas de casos de uso a la par que tablas que especifican el contenido de dichos casos de uso.

En la figura 12 se muestra un ejemplo de diagrama de caso de uso:

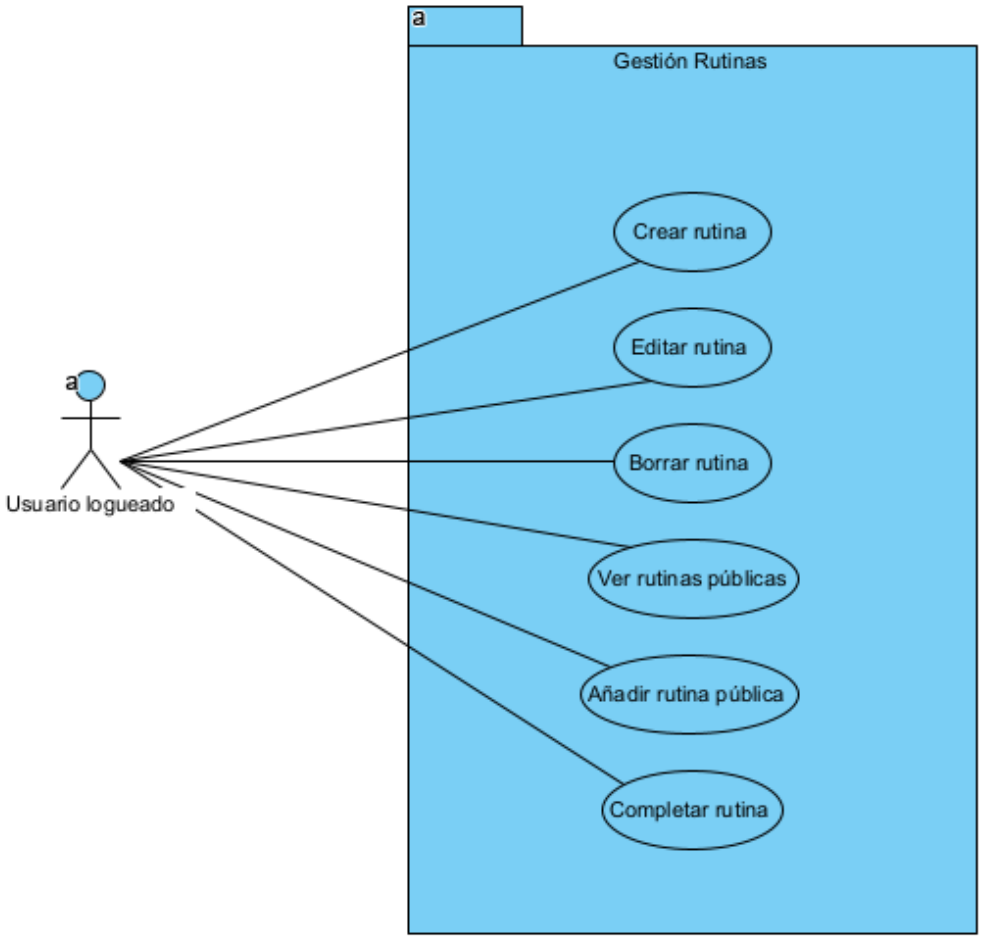

Figura 12: Diagrama de casos de uso "Gestión rutinas"

En la *tabla 6* se ve la especificación del caso de uso anterior:

UC-017	**Crear rutina**	
Versión	1.0	
Autores	Jorge Cruz García	
Fuentes	• Sales Mendes, André Filipe • Villarubia González, Gabriel	
Dependencias		
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario *solicite crear una rutina.*	
Precondición	• OBJ-006 Alta, baja y visualización de rutinas	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario logueado (ACT-002) *solicita crear una rutina.*
	2	La aplicación *le solicita los datos necesarios: nombre, tipo, horario para las notificaciones, nivel de privacidad y un icono decorativo.*
	3	El actor Usuario logueado (ACT-002) *proporciona los datos explicados en el paso anterior.*
	4	La aplicación *crea la rutina y la almacena. En caso de ser pública, la añade a la lista de rutinas públicas que se pueden descargar.*
Postcondición		
Excepciones	**Paso**	**Acción**
	3	No se proporcionan todos los datos o alguno de ellos es inválido. *Este caso de uso queda sin efecto.*
Rendimiento	**Paso**	**Tiempo máximo**
	-	-
Frecuencia esperada	20 veces por día(s)	
Importancia	Vital	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 5: UC-017 Crear rutina

8.3.5 Requisitos no funcionales

La definición de requisitos no funcionales conlleva encontrar las limitaciones de diseño que tendrá que tener nuestra aplicación para ser usada en general. En nuestro caso, se definen la manejabilidad simple, y la concurrencia.

En la *tabla 7* se expone un ejemplo de la definición de un requisito no funcional:

NFR-002	**Concurrencia**
Versión	1.0
Autores	Cruz García, Jorge
Fuentes	• Sales Mendes, André Filipe • Villarubia González, Gabriel
Dependencias	Ninguna
Descripción	El sistema *deberá permitir que pueda ser usado por varios usuarios a la vez*.
Importancia	Importante
Urgencia	Hay presión
Estado	Validado
Estabilidad	Alta
Comentarios	Ninguno

Tabla 6: NFR-002 Concurrencia

8.4 ANÁLISIS DE REQUISITOS

Durante la fase de análisis del sistema se ha realizado una profundización en el nivel de detalle de los requisitos que se habían determinado en la fase anterior, definiendo el modelo del dominio, la realización de los casos de uso, las clases de análisis y una primera vista de arquitectura al nivel de análisis.

Para obtener información más detallada del proceso realizado durante la fase de análisis se puede consultar el documento adjunto *Anexo III – Análisis de requisitos* que se incluye junto a esta memoria.

8.4.1 Modelo del dominio

El modelo del dominio detalla la información y gestión de información dentro del sistema de base de datos. Para detallar esto, se define un diagrama que explica el modelo entidad-relación de nuestras entidades del modelo de negocio.

En la *figura 13* se puede ver un esquema del modelo del dominio:

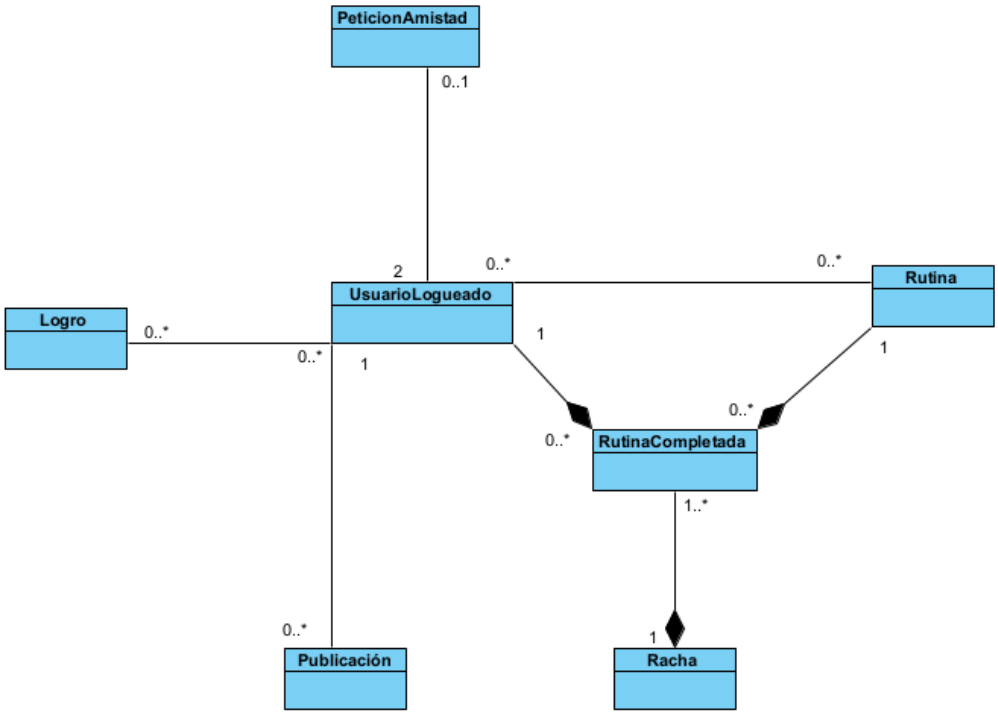

Figura 13:: Modelo del dominio

8.4.2 Realización de casos de uso – análisis

Durante la realización de casos de uso en análisis, se detalla de forma genérica los pasos que debería de tener caso de uso, y el flujo de su ejecución, a través de diagramas de secuencia.

En la *figura 14* se puede ver un ejemplo de diagrama de secuencia:

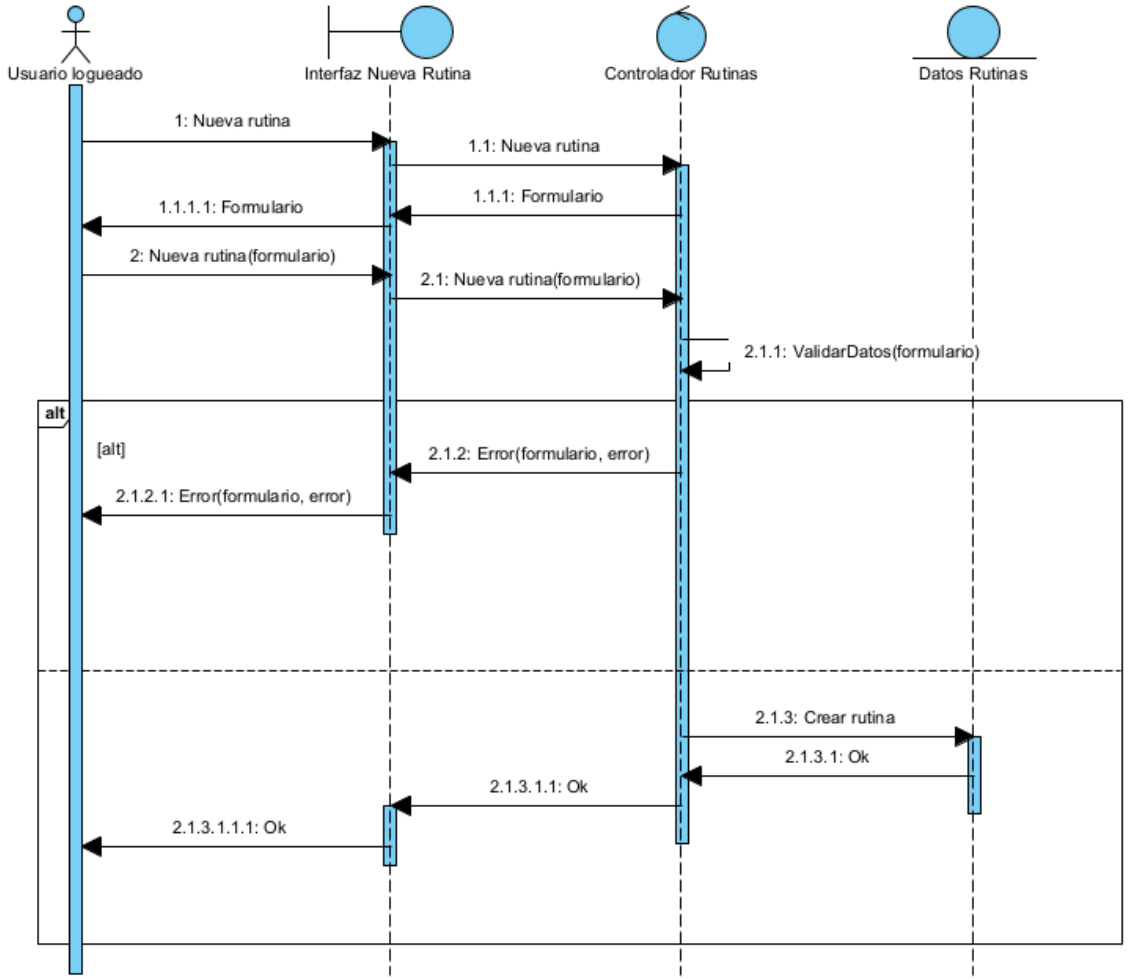

Figura 14: Diagrama de secuencia "crear rutina"

8.4.3 Clases de análisis

En el apartado de las clases de análisis, se han realizado diagramas de comunicación que detallan la forma en la que las clases que hemos establecido a nivel de análisis durante la elaboración de los diagramas de secuencia se comunican y se dividen en paquetes.

En la *figura 15* se puede ver un ejemplo de diagrama de comunicación entre clases de análisis:

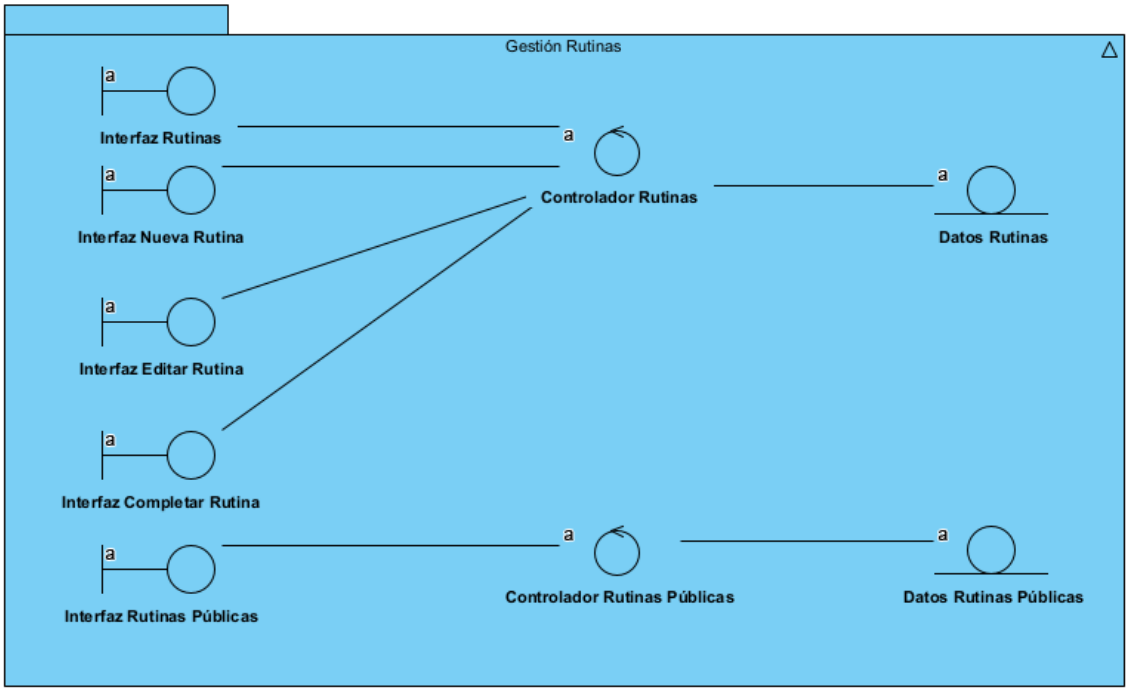

Figura 15: Diagrama de comunicación del paquete "Gestión Rutinas"

8.5 DISEÑO DEL SISTEMA

La fase de diseño se centra en el dominio de la solución, y representa de forma mucho más detallada y a bajo nivel el contenido de nuestro proyecto. En este apartado se obtiene ya una aproximación muy cercana a la implementación real del proyecto, incluyendo nombres de clases, métodos y atributos que podrían ser los mismos que se encontrarán a lo largo de la implementación final en código.

Para obtener información detallada sobre el proceso realizado durante la elaboración del diseño, se puede consultar el documento *Anexo IV – Diseño del sistema*, que se encuentra adjunto junto a esta memoria.

8.5.1 Patrones arquitectónicos

Para el desarrollo de la aplicación se ha seguido el patrón **BLoC**, que es muy parecido al patrón Modelo Vista Controlador. También se ha usado el patrón Singleton, para definir objetos globales de los cuales sólo necesitamos una instancia, pero cuya instancia ha de ser compartida por toda la aplicación.

8.5.1.1 Patrón BLoC

El patrón **BLoC** (*Business Logic Component)* es un patrón de código que separa de forma clara la capa de vista, de lógica y de datos. Se compone de los tres siguientes componentes:

- **Vista:** Las vistas definen cada una de las pantallas y *widgets* con los cuales representaremos nuestros datos. Para realizar funciones de lógica de negocio, la vista llamará a los BLoC.

- **BLoC:** Un BLoC encapsula las funcionalidades necesarias para todas nuestras vistas, realizando distintas operaciones sobre los datos que nos proveen los repositorios.
- **Repositorio o "Data":** Un repositorio es el encargado de proveer de datos a los BLoC. Encapsulan el acceso a los datos en una clase repositorio, de forma que, a la hora de desarrollar los BLoC y la lógica de negocio, no sea necesario conocer de donde proveen dichos datos.

El patrón BLoC aporta muchas ventajas:

- **Separación clara de responsabilidades.**
- **Bajo acoplamiento entre las capas.**
- **Desarrollo independiente de las capas.**

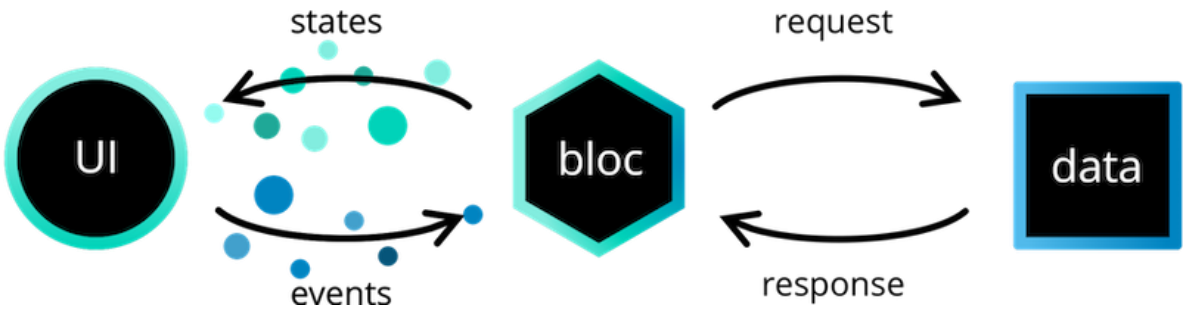

Figura 16: Flujo del patrón BLoC

En la *figura 17* se puede ver un ejemplo de cómo se estructura una aplicación que usa el patrón BLoC. Como podemos ver, los BLoC implementan todas las funcionalidades que requieren las vistas, mientras que los repositorios abstraen la obtención y modificación de datos de forma que los BLoC no necesiten entender si estos datos vienen de una API, de nuestra base de datos, de nuestro almacenamiento local, ni tengan que preocuparse de "parsearlos".

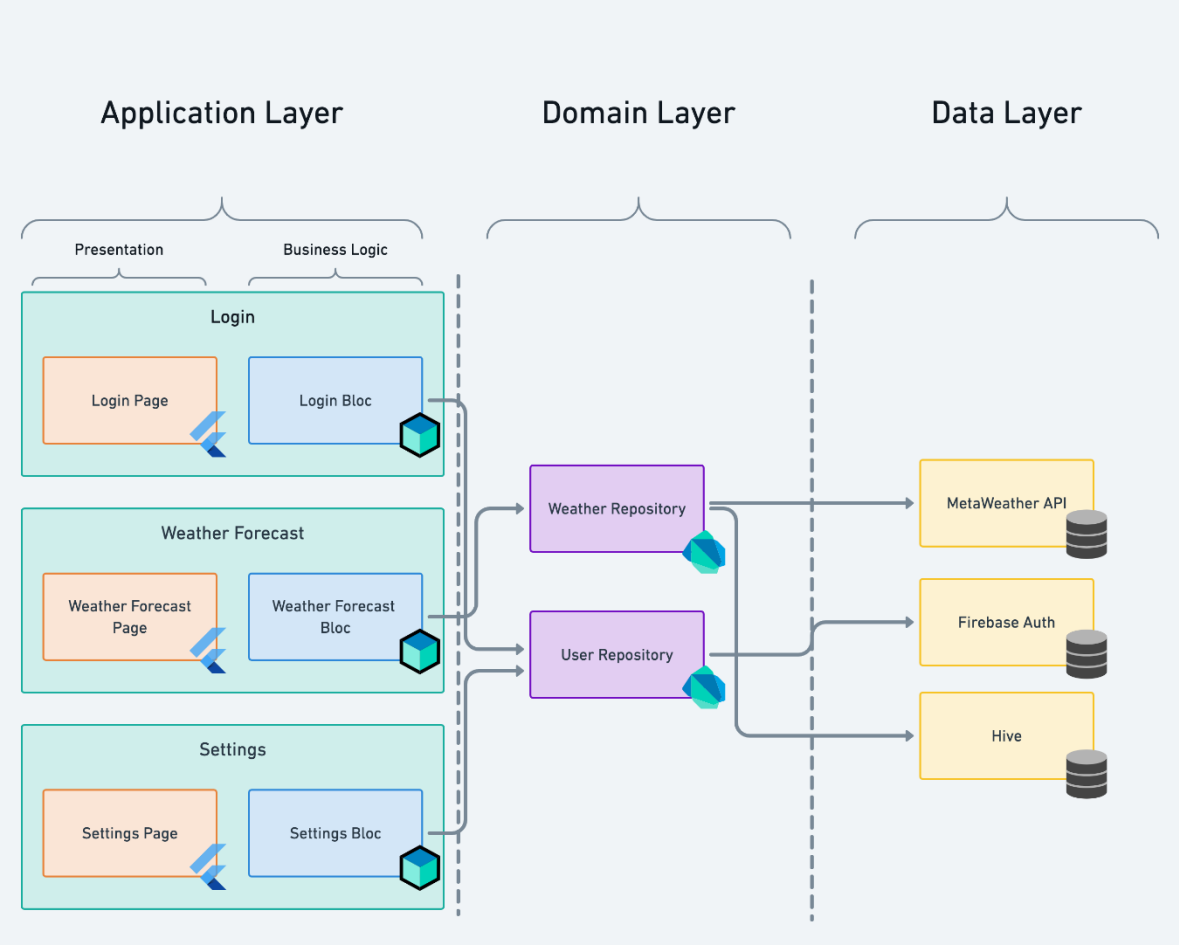

Figura 17: Vista de arquitectura de una aplicación que usa el patrón BLoC

El patrón BLoC no solo establece una arquitectura sólida para el desarrollo de la aplicación, si no que por su implementación a través de una biblioteca que permite gestionar el estado visual de la aplicación según el estado de los BLoC de forma automática, permite actualizar el estado de la aplicación de forma más simple, más rápida y más eficiente en memoria tal como se explica en *"Performance Analysis of BLoC and Provider State Management Library on Flutter"* (Prayoga, Munawar, Jumiyani, & Syalsabila, 2021)

8.5.1.2 Patrón Singleton

El patrón Singleton permite definir una clase con una sola instancia a lo largo del ciclo de vida de la ejecución de la aplicación.

Su propósito es garantizar que una clase cuya instanciación puede ser costosa y que es necesaria en varios lugares de la aplicación pueda ser accedida de forma simple y única.

La implementación del patrón es simple, y simplemente consiste en crear un constructor que en caso de no existir una instancia la clase, la cree, y en caso de que exista, se devuelva la instancia ya existente en vez de una nueva.

8.5.2 Clases de diseño

Durante la parte de las clases de diseño, se han separado las clases que tenemos en paquetes de **Vista, BLoC, Repositorio y Datos,** definiendo así una primera vista detallada del contenido de la implementación de la aplicación.

En la *figura 18* se puede ver un diagrama que explica la capa BLoC de la aplicación. Dentro del *Anexo IV* podemos consultar el resto de diagramas, que explican todas las capas de la aplicación.

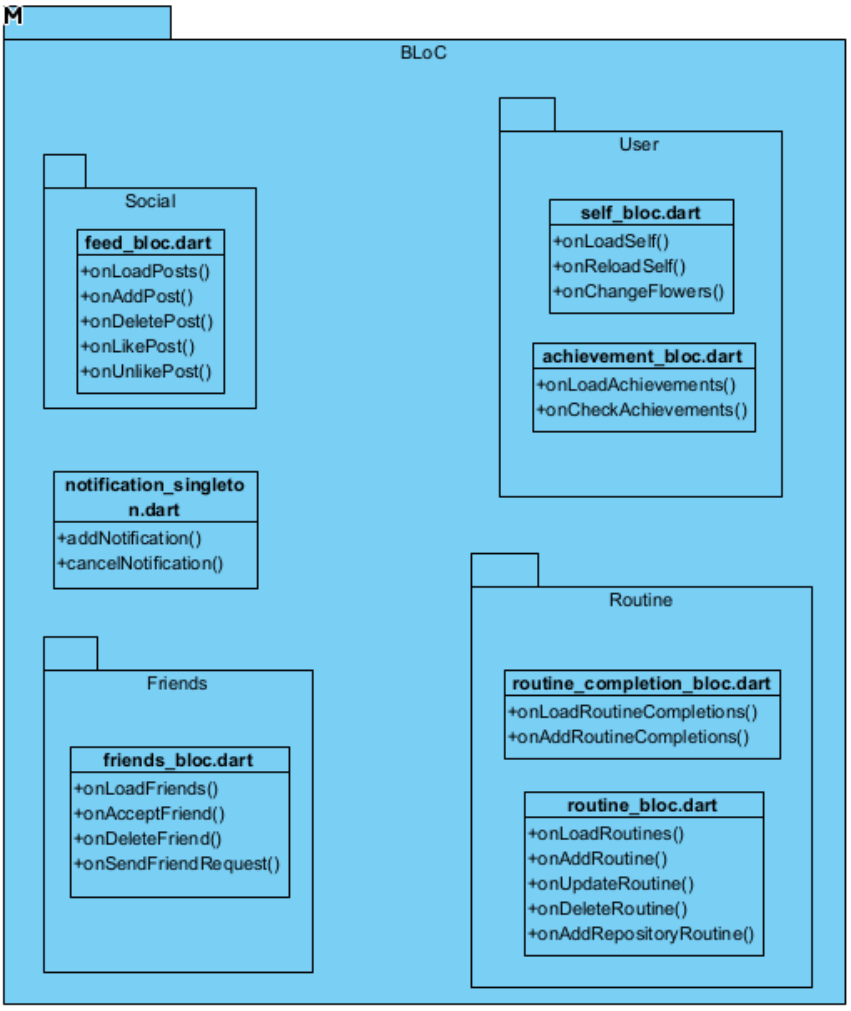

Figura 18: Capa BLoC

8.5.3 Vista arquitectónica del modelo del diseño

Durante este apartado, se ha definido la primera vista ordenación arquitectónica de la aplicación, aplicando el patrón BLoC.

En la *figura 19* se puede ver el diagrama de arquitectura al que se ha llegado:

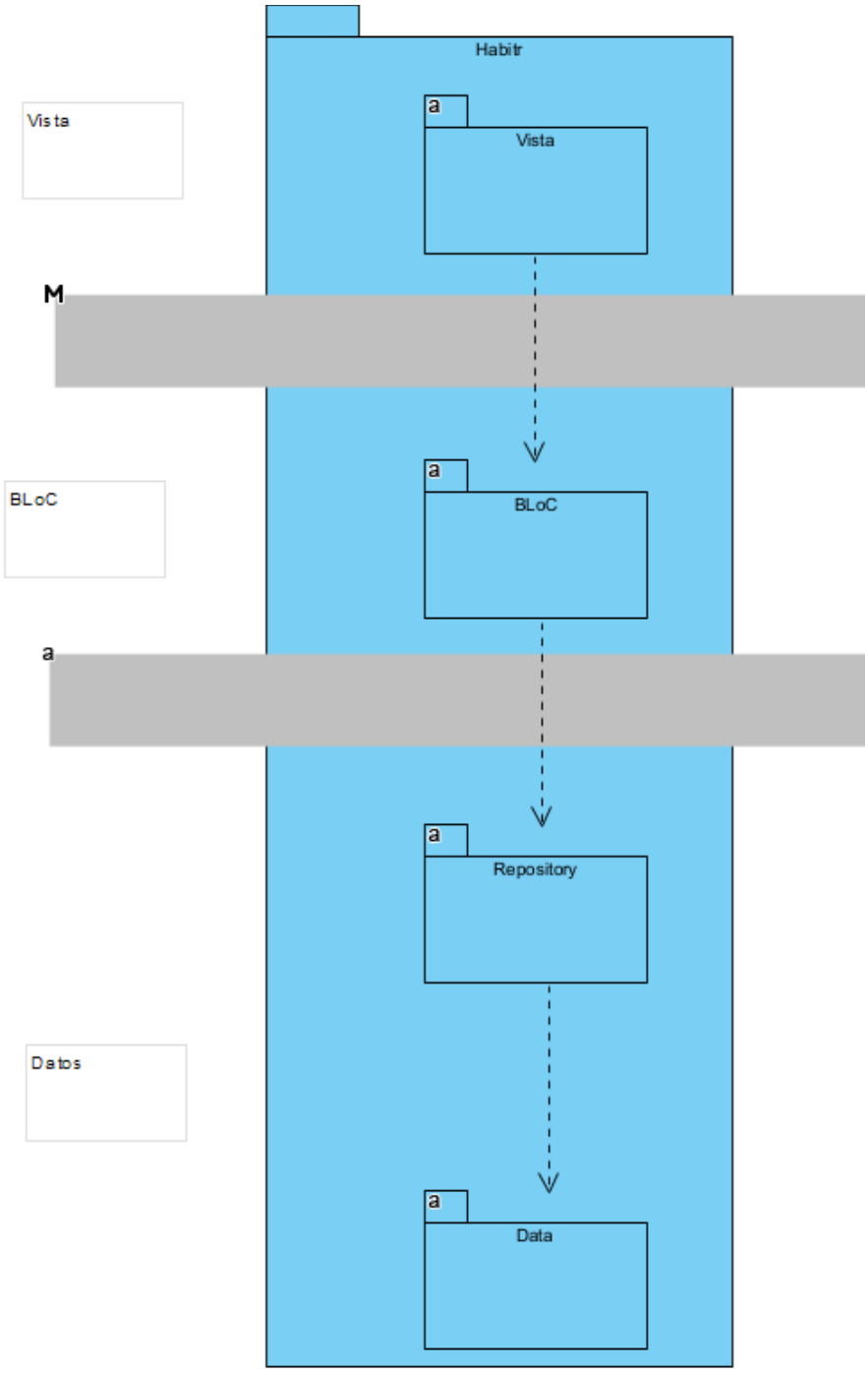

Figura 19: Vista arquitectónica de Habitr

8.5.4 Realización de casos de uso – diseño

En la realización de casos de uso de la etapa de diseño, se han detallado los diagramas de secuencia obtenidos en la fase de análisis, explicando los pasos de forma más detallada e introduciendo ya los nombres reales de las clases y módulos que podremos ver en la implementación.

En la *figura 20* se puede ver un ejemplo de diagrama de secuencia:

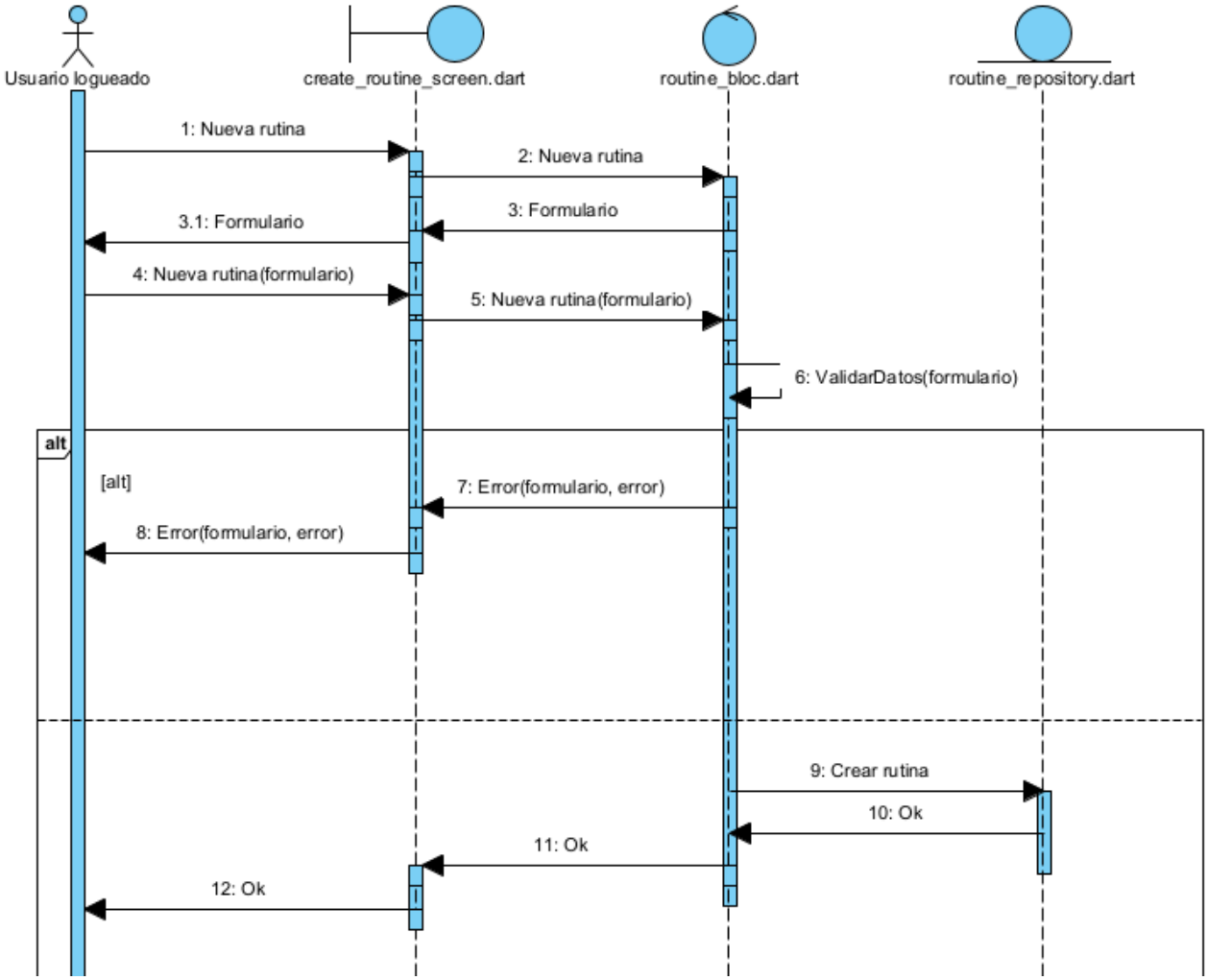

Figura 20: Diagrama de secuencia del caso de uso "Crear rutina" – diseño

8.5.5 Diseño de base de datos

Para que el sistema pueda almacenar la información previamente definida durante el *Anexo II* a través de los requisitos de información, se ha desarrollado el diseño de base de datos de SQL que se puede ver en la *figura 21*.

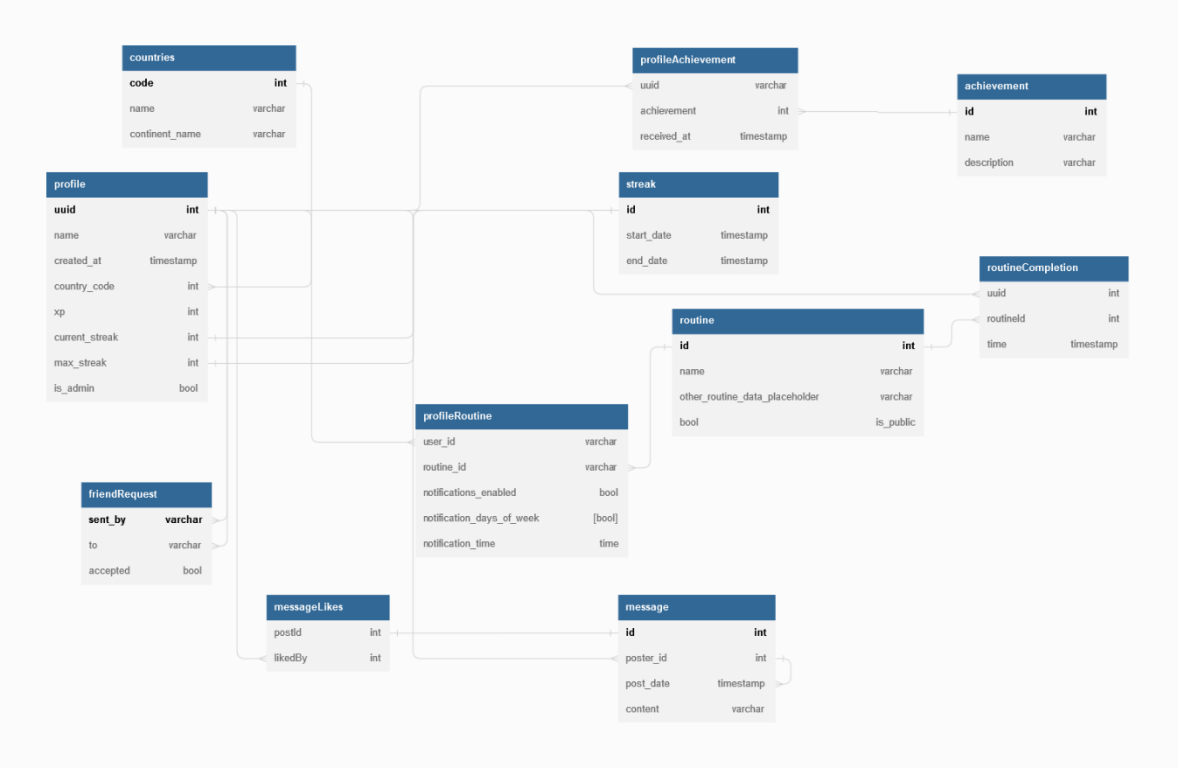

Figura 21: Modelo entidad-relación

8.5.6 Modelo de despliegue

El paso final que se ha realizado durante la fase de diseño ha sido un diagrama de despliegue (*figura 22)* que permite observar cómo se organizan y conectan los distintos nodos para el despliegue final.

Los tres nodos definidos en el diagrama son los siguientes:

- **Supabase:** Representa la base de datos distribuida que ha sido utilizada, y con la cual se conectarán todas las instancias de ejecución de la aplicación.
- **Teléfono Android:** Representa el entorno de ejecución de la aplicación en teléfonos Android.
- **Teléfono iOS:** Representa el entorno de ejecución de la aplicación en teléfonos iOS.

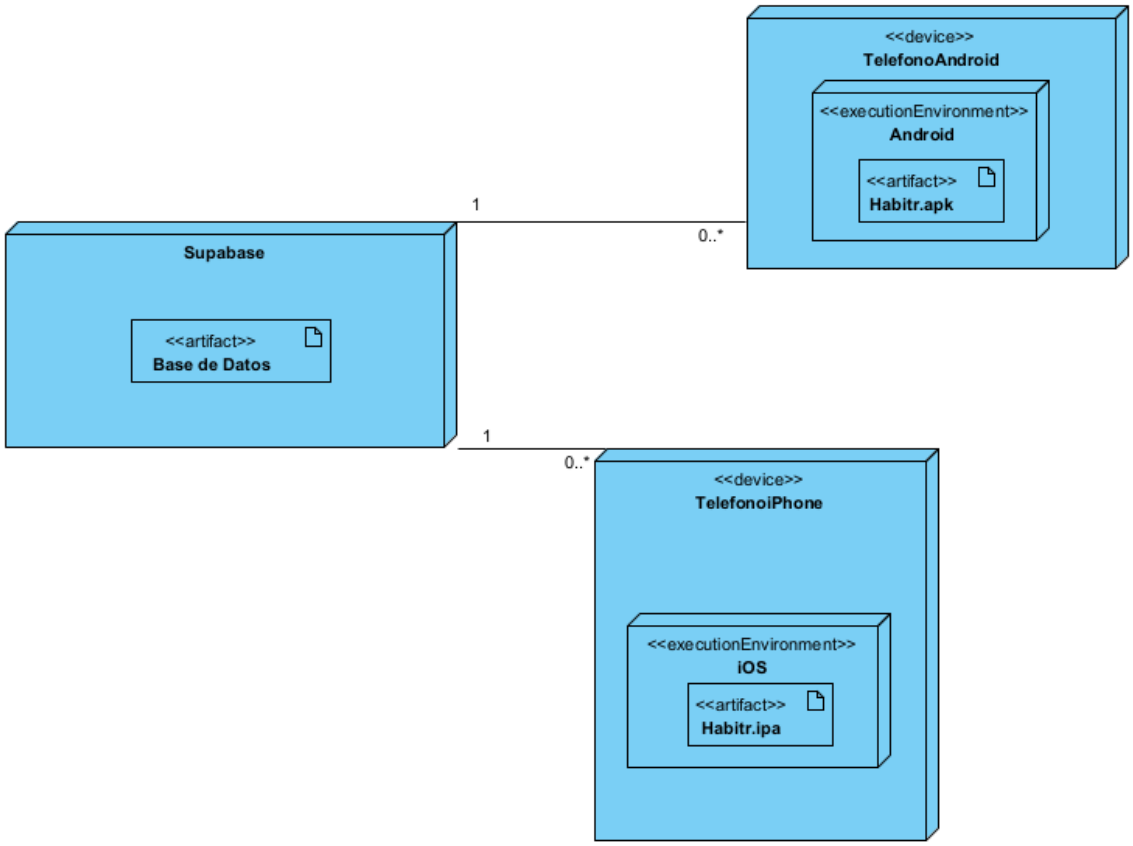

Figura 22: Diagrama de despliegue

8.6 IMPLEMENTACIÓN

Durante la fase de implementación se llevó a cabo toda la programación de la aplicación.

Es una de las fases más largas dado a la cantidad de problemas técnicos que suelen surgir durante un proceso así, pero haber realizado un proceso correcto de *Ingeniería del Software* permitió mitigar los efectos de dichos problemas.

A la hora de hacer el desarrollo, se han seguido los esquemas realizados previamente en la fase de diseño.

Se separó el desarrollo de la aplicación en tres capas para cada funcionalidad, y se fueron implementando lentamente cada una de ellas:

- **Vista:** Para empezar, se realizaba una implementación parcial del prototipo de la interfaz que se había creado previamente en Adobe XD, tal cómo se explicó en la sección de técnicas y herramientas y se probaba ese prototipo con datos falsos. Una

vez la funcionalidad estaba totalmente implementada, se acababa de retocar la vista para crear la versión final.

- **Repositorio:** Antes de empezar la codificación de cada funcionalidad, se concretaban los atributos que debía de incluir cada objeto relacionado a esa funcionalidad en nuestra BBDD, y se codificaba su clase y los métodos de repositorio que nos permitían recuperar objetos de la BBDD
- **BLoC:** Una vez se tenía la vista y los datos, se implementaba la funcionalidad necesaria para la aplicación y se usaba dicha implementación como "pegamento" entre la vista y los repositorios.

Tras haber desarrollado cada una de las tres capas, se iban refinando poco a poco hasta obtener la versión final.

8.7 PRUEBAS

Las pruebas son una parte muy importante en el desarrollo del software, puesto que son necesarios para realizar una garantía de la calidad y funcionamiento del producto desarrollado.

A lo largo del desarrollo, y tras el final de él, se han ido realizando pruebas unitarias manuales de cada uno de los componentes del sistema para comprobar su funcionamiento individual.

También se han realizado pruebas globales de la aplicación, comprobando si la integración entre distintos componentes es la adecuada y trabajan de forma correcta los unos con los otros.

Finalmente, se hicieron pruebas con amigos y familiares para detectar ciertos fallos de diseño o usabilidad.

8.8 FUNCIONALIDAD DEL SISTEMA

En este apartado se mostrará el funcionamiento de los distintos módulos de la aplicación. Para obtener una vista más detallada de cómo usar la aplicación se incluye el *Anexo V – Manual de usuario* donde se explica todo por pasos.

8.8.1 Autenticación

Nada más acceder a la aplicación, se nos enseñará la pantalla de login, la cual se puede ver en la *figura 23*. Si tenemos una cuenta, podemos rellenar el formulario para hacer acceso a la zona principal de la aplicación. Alternativamente podemos acceder a la pantalla de registro (*figura 24*) o al menu de recuperar contraseña (*figura 25)* presionando el texto inferior.

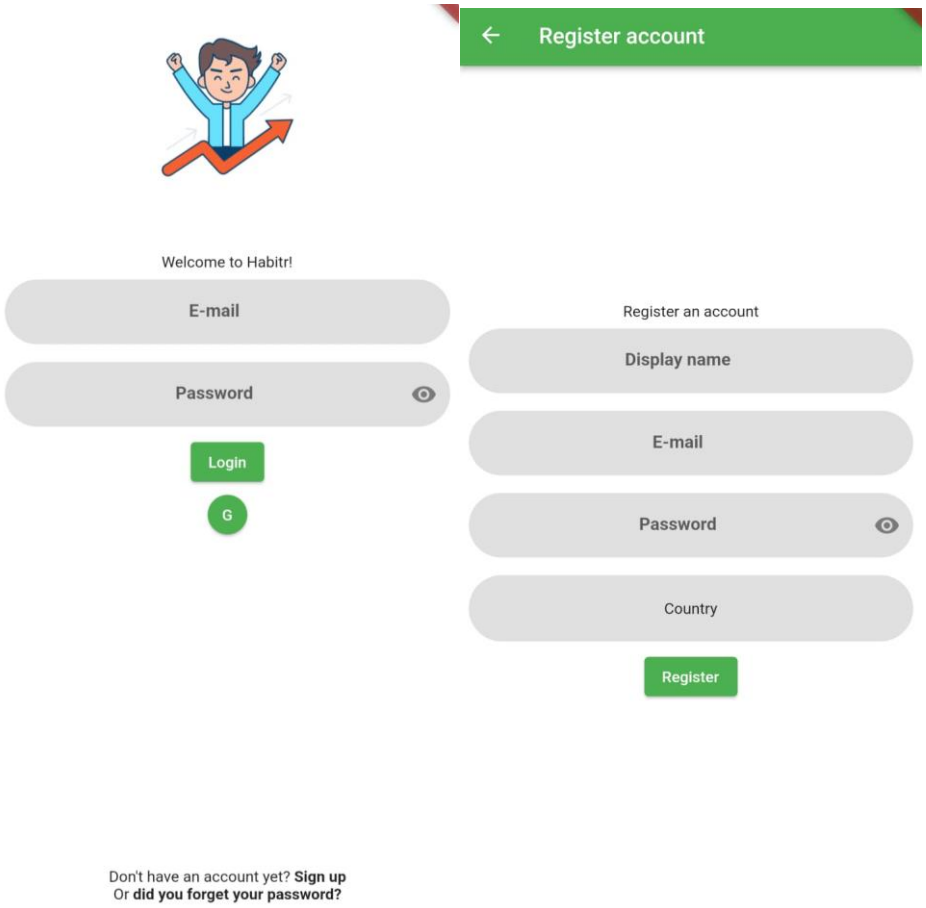

Figura 23: Vista Login

Figura 24: Pantalla Registro

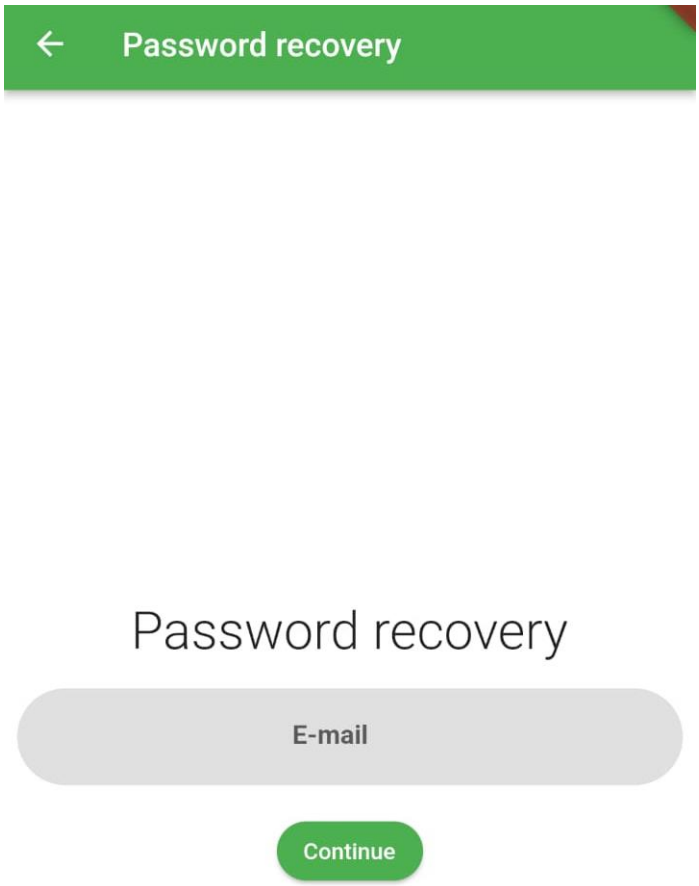

Figura 25: Pantalla Recuperar Contraseña

8.9 PANTALLA PRINCIPAL

Una vez se ha hecho login, se nos redirige a la pantalla principal de la aplicación. Podemos cambiar de vista usando la **barra de la parte inferior de la pantalla.** Se enseña esta vista en la *figura 26.*

Desde esta misma pantalla, podemos actuar sobre el juego (dándole click a las flores para cambiarlas), además de acceder al feed, el cual se ve en la *figura 27* deslizando hacia la derecha.

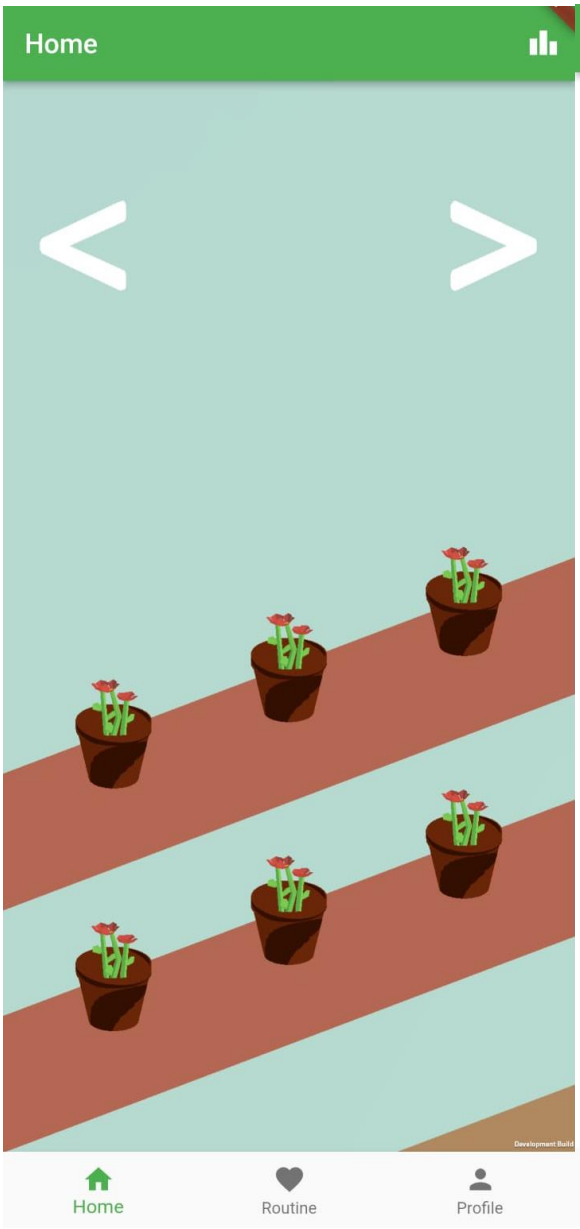

Figura 26: Pantalla Principal

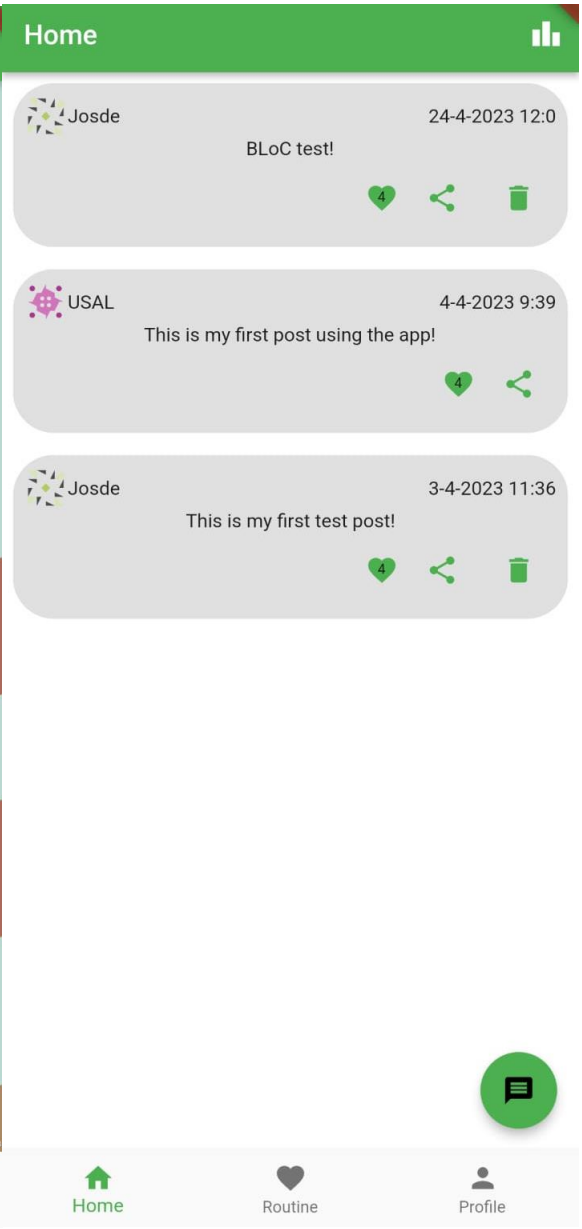

Figura 27: Pantalla Feed

8.10 GESTIÓN DE RUTINAS

Podemos acceder a la vista de gestión de rutinas clickeando el botón de "**Routine**" que podemos ver en la barra inferior de la aplicación en la *figura 26.*

Desde esta vista, la cual se enseña en la *figura 28* podemos crear, editar y borrar rutinas, además de acceder al repositorio de rutinas online.

Figura 28: Pantalla Lista de Rutinas

8.10.1 Añadir una rutina

Para añadir una rutina, podemos hacer click en el botón de “+” que podemos ver en la parte inferior de la lista de rutinas, en la *figura 28*, y rellenar el formulario que se nos presenta.

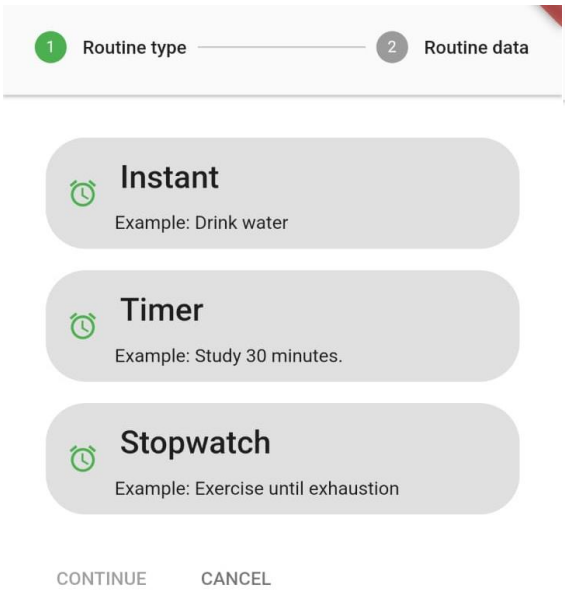

Figura 29: Formulario Nueva Rutina, parte 1

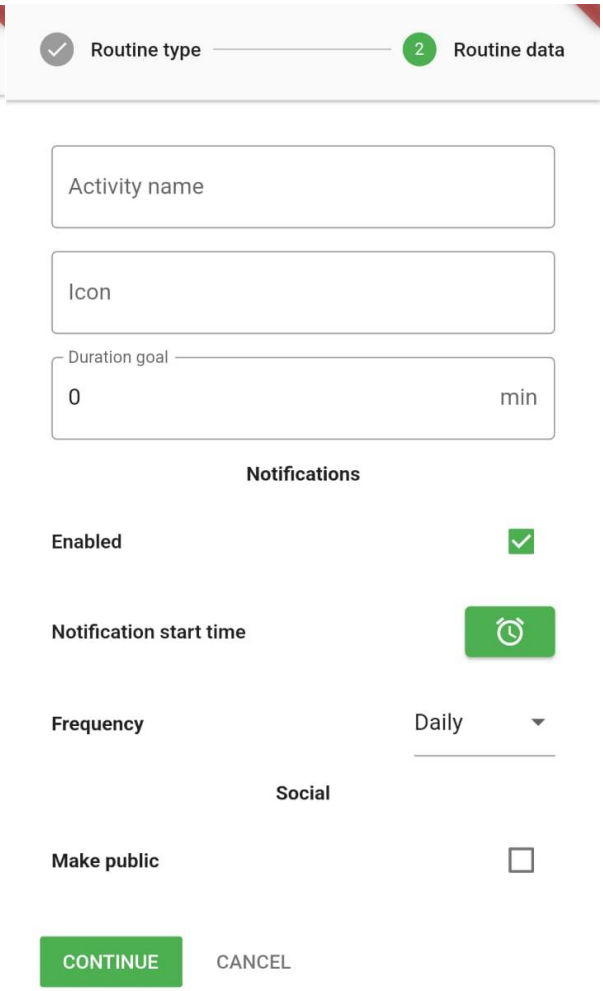

Figura 30: Formulario Nueva Rutina, parte 2

8.10.2 Consultar rutinas online

Para consultar el repositorio de rutinas online, se ha de hacer click en el botón que se encuentra al lado del botón "+" en la *figura 28*.

Figura 31: Pantalla Rutinas Online

8.10.3 Realizar una rutina

Para realizar una rutina, se ha de clickear en el nombre de la rutina desde la vista de lista que se ve en la *figura 28*.

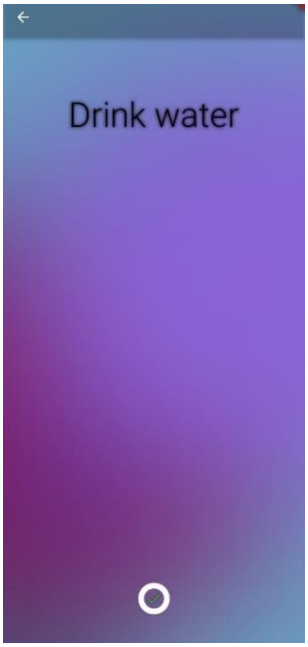

Figura 32: Pantalla Realizar Rutina

8.11 GESTIÓN DE USUARIO

8.11.1 Ver perfil y estadísticas

Para visualizar nuestro propio perfil, debemos darle click al botón “Profile” de la barra de la parte inferior de la pantalla, enseñada previamente en la *figura 26.* Para ver las estadísticas, podemos darle click al botón “Stadistics” que se ve en la *figura 33*

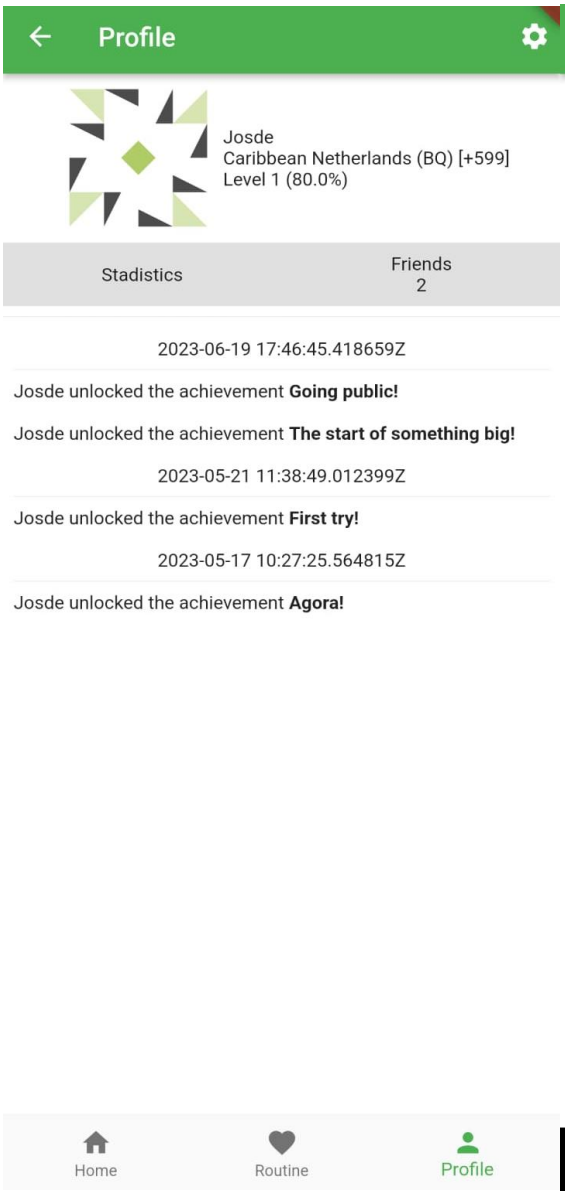

Figura 33: Pantalla Ver Perfil

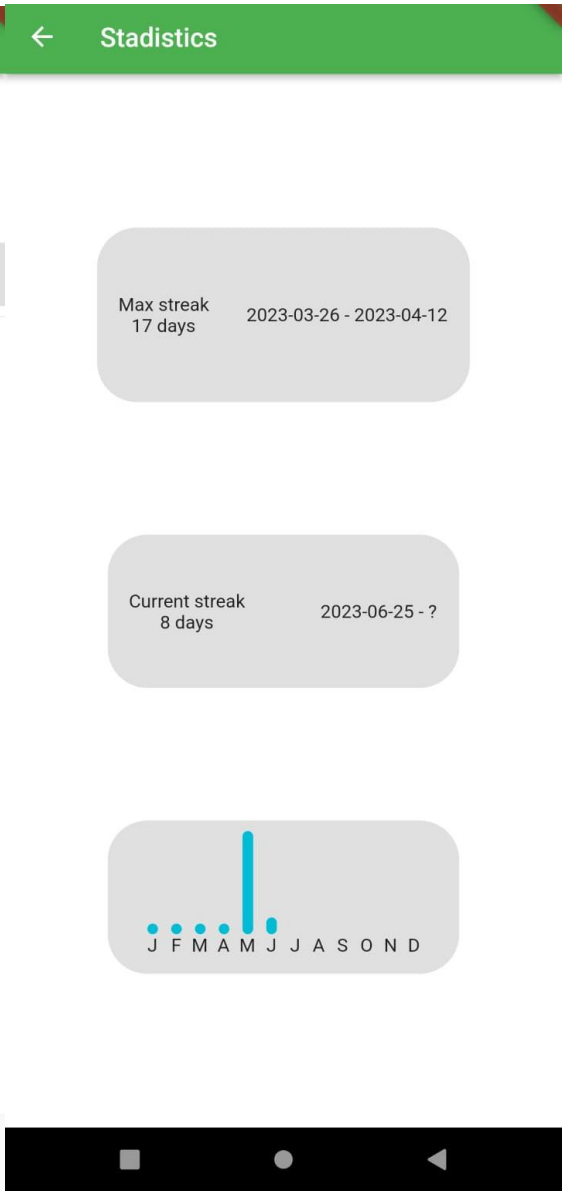

Figura 34: Pantalla Estadisticas

8.12 GESTIÓN SOCIAL

8.12.1 Ver y añadir amigos

Para ir a la parte de la gestión social, nos situamos en el perfil de nuestro usuario y le damos click al botón que pone “Friends”, el cual se puede apreciar en la *figura 33.* Esto nos traerá a la pantalla de amigos, la cual se puede ver en la *figura 35*. Presionar el botón de arriba una vez más nos llevará a la opción de añadir amigos, que se puede ver en la *figura 36*.

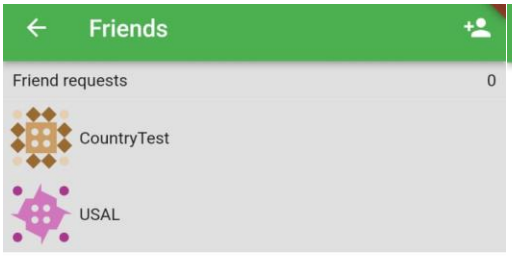

Figura 35: Pantalla Amigos

Figura 36: Pantalla Añadir Amigo // Compartir QR

9 LIMITACIONES DEL PROTOTIPO

Debido a las limitaciones temporales del proyecto, no se pudieron integrar algunas funcionalidades que se tenían ideadas y que serían de gran ayuda para cumplir de forma más efectiva los objetivos del proyecto.

Las dos limitaciones más grandes del prototipo actual son las siguientes:

- No existe un tutorial guiado de la aplicación, ni ningún tipo de explicación dentro de la misma. Se hizo la investigación para añadir un tutorial de este estilo, descubriendo un módulo de Flutter que lo facilitaba. Sin embargo, como se decía previamente, no hubo tiempo para incluirlo.
- Ampliación de la gamificación, incluyendo mecánicas que fomenten el acceso diario, una expansión en el nivel de personalización. Se tenía ideado un sistema por el cual las flores se irían marchitando, y habría que obtener agua completando las rutinas establecidas para poder mantenerlas vivas, pero no pudo ser implementado finalmente.

10 CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

Este apartado recoge las conclusiones que se han obtenido tras el desarrollo completo del proyecto, junto a las líneas de trabajo futuras con ideas para mejorar el proyecto que no pudieron ser realizadas por falta de medios o de tiempo.

10.1 CONCLUSIONES

Una vez terminado el proyecto, se puede decir que se han cumplido los objetivos establecidos al principio del proyecto:

- Se ha diseñado y llevado a cabo el desarrollo del proyecto hasta el punto de un primer prototipo funcional.
- Se han cumplido los objetivos funcionales que se establecieron al inicio del proyecto (Seguimiento de rutinas, notificaciones, estadísticas, funcionalidad social y gamificación)
- Se ha obtenido una vista sobre el esfuerzo que conlleva desarrollar un proyecto desde la fase de idea hasta el primer prototipo funcional, pasando primero por hacer todo el proceso de *ingeniería del software*.
- Se ha obtenido experiencia en el lenguaje Dart y en el framework Flutter, permitiendo desarrollar aplicaciones multiplataforma con tecnología puntera en el futuro.
- Se ha obtenido experiencia en el patrón BLoC, que a pesar de su reciente creación ha demostrado a lo largo del desarrollo ser un patrón que facilita enormemente el desarrollo de una aplicación, además de su mantenimiento y expansibilidad.

10.2 LÍNEAS DE TRABAJO FUTURO

A pesar de que se han completado muchos de los objetivos con los cuales se inició el desarrollo, hay varios que fueron incluidos en la propuesta del proyecto, pero que no pudieron ser completados por razones de tiempo y dificultad de desarrollo, y son los siguientes:

- Integración con servicios de Google Fit y Apple Health, para poder incluir objetivos basados en el seguimiento de estadísticas físicas como los pasos diarios.
- Análisis de las estadísticas recogidas de cada usuario, además de un servicio estadístico más completo.

Estas funcionalidades, junto a las explicadas previamente en el apartado de *limitaciones del prototipo*, quedarían pendientes para una futura versión mejorada de la aplicación.

11 REFERENCIAS

- Documentación Flutter:
  - https://docs.flutter.dev/
- Documentación BLoC:
  - https://bloclibrary.dev/
- Manual Unity:
  - https://docs.unity3d.com/Manual/index.html
- Documentación DBDiagram:
  - https://dbdiagram.io/docs/
- Documentación Supabase:
  - https://supabase.com/docs
- Documentación PostgreSQL:
  - https://www.postgresql.org/docs/current/
  - https://www.postgresql.org/docs/current/ddl-rowsecurity.html

12 BIBLIOGRAFÍA

Brennan, D., & WebMD, E. (2021). *Psychological Benefits of Routines*. Obtenido de WebMD: https://www.webmd.com/mental-health/psychological-benefits-of-routine

Durán Toro, A., & Bernárdez Jímenez, B. (2000). *Metodología para la Elicitación de Requisitos de Sistemas Software.* Sevilla: Universidad de Sevilla. Recuperado el 17 de 6 de 2023, de http://www.lsi.us.es/docs/informes/lsi-2000-10.pdf

OMS. (2022). *La pandemia de COVID-19 aumenta en un 25% la prevalencia de la ansiedad y la depresión en todo el mundo.* Ginebra: OMS. Obtenido de https://www.who.int/es/news/item/02-03-2022-covid-19-pandemic-triggers-25-increase-in-prevalence-of-anxiety-and-depression-worldwide

Prayoga, R. R., Munawar, G., Jumiyani, R., & Syalsabila, A. (2021). Performance Analysis of BLoC and Provider State Management Library on Flutter. *Journal Mantik*.

Pressman, R. S. (2010). *Ingeniería del Software: Un enfoque práctico.* McGraw-Hill.