Logo_Cam_Rojo

CONSEJERÍA DE EDUCACIÓN

Comunidad de Madrid

**IES ENRIQUE TIERNO GALVAN**

**Parla**

**CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA**

**Curso 2024/2025**

**Proyecto DAM**

***TITULO:*** Zenhabits

***Alumno*:** Alba Almoril Benito

***Tutor*:** Julián Parra Perales

Junio de 2025

**CONTENIDO**

[**1.** **Contexto de la aplicación** 1](#_Toc200005425)

[1.1. Solución y objetivo 1](#_Toc200005427)

[**2. Análisis 2**](#_Toc200005428)

[2.1. Análisis de requisitos 2](#_Toc200005429)

[2.2. Casos de uso 3](#_Toc200005432)

[**3. Diseño 4**](#_Toc200005433)

[3.1. Modelo arquitectónico 4](#_Toc200005434)

[3.2. Plan de pruebas 6](#_Toc200005435)

[**4. Base de datos 8**](#_Toc200005436)

[4.1. Contexto de la BD 8](#_Toc200005437)

[4.2. Entidades y MER 9](#_Toc200005438)

[**5. Interfaz 10**](#_Toc200005439)

[5.1. GUI 10](#_Toc200005440)

[5.2. Diagrama de navegación 12](#_Toc200005443)

[5.3. Características visuales 13](#_Toc200005444)

[5.4. Usabilidad 14](#_Toc200005448)

[**6. Implementación 14**](#_Toc200005449)

[6.1. Entorno de desarrollo 15](#_Toc200005450)

[6.2. Tecnologías 15](#_Toc200005451)

[6.3. Funcionamiento 17](#_Toc200005452)

[6.4. Puesta en producción 18](#_Toc200005453)

[6.5. Líneas futuras 18](#_Toc200005454)

[**7. Elementos destacables del desarrollo 19**](#_Toc200005455)

[7.1. Innovaciones 19](#_Toc200005456)

[7.2. Problemas 21](#_Toc200005457)

[**8. Conclusiones 23**](#_Toc200005460)

[**Bibliografía 25**](#_Toc200005461)

[**Anexo 1 26**](#_Toc200005462)

[**Anexo 2 27**](#_Toc200005463)

[**Anexo 3 28**](#_Toc200005464)

# **Contexto de la aplicación**

La sociedad actual lleva un ritmo de vida rápido, el cual muchas veces dificulta el hecho de mantener costumbres saludables y ser productivos.

Según investigaciones recientes (Duhigg, 2012), la creación y mantenimiento de hábitos demandan constancia, repetición y motivación duradera. Sin embargo, la mayoría de las personas desisten en pocas semanas debido a la falta de disciplina, la falta de motivación o incluso por olvidar esas responsabilidades. A esto se le añade la escasez de herramientas eficaces que respondan a esa necesidad.

Hay numerosas aplicaciones en el mercado enfocadas en la administración de tareas (como Todoist o Trello) o hábitos (como Habitica o HabitNow), sin embargo, muchas de estas tienen restricciones: su utilización se basa únicamente en la conexión a internet, poseen interfaces poco intuitivas o no producen el compromiso adecuado.

## **Solución y objetivo**

En respuesta a la necesidad de la sociedad actual de mantener hábitos saludables y de alcanzar metas personales en un mundo demasiado rápido y ajetreado; propongo el desarrollo de *ZENHABITS*, una aplicación multiplataforma diseñada para mejorar la productividad personal gracias a una combinación de gestión y “gamificación” inspirada en Habitica.

*ZENHABITS* no solo ofrece herramientas de organización; además, busca motivar al usuario a través de recompensas como logros, lo cual fomentará la constancia y el compromiso a largo plazo. También permitirá gestionar hábitos y metas de forma sencilla e intuitiva. Cada vez que el usuario complete, queriendo decir que mantiene un hábito, será recompensado mediante logros y avances visibles en un personaje, como por ejemplo un aumento de monedas que se podrán usar para personalizar a dicho. También, este personaje gastará energía, subirá de nivel o perderá vida según el grado de cumplimiento de las tareas, funcionando como una especie de reflejo del progreso personal.

El objetivo de este proyecto es que, ZENHABITS, consiga mantener la motivación y constancia de los usuarios ofreciéndoles una experiencia agradable, intuitiva y divertida. Además, como objetivo personal, aspiro a que el desarrollo de esta aplicación me proporcione nuevas habilidades, herramientas y lenguajes para mi crecimiento personal.

# **Análisis**

El desarrollo técnico y general de *ZENHABITS*, se basa en que es una aplicación que ofrece una interfaz atractiva y multiplataforma, junto con un sistema de almacenamiento híbrido: combina el almacenamiento de datos local y sincronización remota. Para asegurar esta cumpla con las necesidades de los usuarios y mantenga una buena calidad, es imprescindible realizar un análisis detallado de los requisitos que guiarán todo el proceso de diseño e implementación.

## **Análisis de requisitos**

Buscamos identificar las funcionalidades esenciales del sistema y las condiciones bajo las que se debe operar. Para identificarlas, haremos el análisis de requisitos, hay dos tipos:

### **Requisitos funcionales**

* Crear, modificar, consultar y eliminar hábitos y metas.
* Gestionar notificaciones para recordar hábitos y metas.
* Desbloquear logros en base al cumplimiento de hábitos y metas.

### **Requisitos no funcionales**

* UI:
* La interfaz será intuitiva, clara y sencilla, permitiendo que cualquier usuario pueda manejarla sin necesidad de formación previa.
* La experiencia de usuario se verá reforzada por un diseño visual minimalista, con iconografía reconocible y navegación fluida.
* Se contempla una guía o ayuda básica accesible para apoyar al usuario.
* Portabilidad:
* La aplicación debe estar disponible para distintos tipos de dispositivos, incluyendo móviles y escritorio, y adaptarse correctamente a diferentes tamaños de pantalla mediante diseño “responsive”.
* Fiabilidad:
* El sistema debe permitir su uso sin conexión a internet, garantizando que los datos esenciales sigan disponibles localmente.
* Se deben evitar pérdidas de información durante la sesión del usuario.
* Seguridad:
* El sistema garantizará la protección de datos personales y credenciales, asegurando que solo el usuario legítimo pueda acceder a su información.
* El acceso estará protegido mediante mecanismos seguros de autenticación.
* Eficiencia:
* El sistema debe responder de forma rápida y fluida a las interacciones del usuario, incluso en dispositivos con recursos limitados.

## **Casos de uso**

En el caso de ZENHABITS, el actor principal es el usuario, el cual es la persona que interactúa con dicha aplicación. Este puede crear, consultar, modificar o eliminar hábitos o metas; desbloquear logros y gestionar o recibir notificaciones.

Para desbloquear logros se debe completar un hábito o una meta varias veces, como conseguir objetivos y los logros son una de las recompensas.

Diagrama

El contenido generado por IA puede ser incorrecto.

# **Diseño**

## **Modelo arquitectónico**

*ZENHABITS* ha sido diseñada para ofrecer una solución *multiplataforma* eficiente, escalable y segura. La propuesta busca ser funcional tanto en dispositivos móviles (Android y iOS) como en escritorio (Windows, macOS y Linux) y Web en fases futuras.

La solución propuesta sigue un modelo cliente-servidor, donde el backend y la aplicación móvil interactúan a través de una API REST. Internamente, tanto el backend como el frontend se estructuran siguiendo los principios de Clean Architecture, lo que permite una separación clara de responsabilidades y una mayor mantenibilidad del código. Esta organización facilita futuras ampliaciones del sistema, ya que las distintas capas: “data” (datos), “domain” (lógica de negocio) y “presentation” (presentación) se encuentran desacopladas.

En la capa “presentation” de la aplicación, se ha implementado el patrón MVVM (Model-View-ViewModel). Este patrón facilita la gestión del estado y la interacción con la lógica de negocio, además de mejorar la testabilidad del código.

Por otra parte, el servicio remoto, tanto la API REST como su base de datos estarán dockerizados para conseguir un entorno aislado, modular y escalable.

Además, se aplicaron prácticas de diseño adaptativo (“responsive design”) para garantizar que la interfaz de usuario se adapte correctamente a distintos dispositivos, tamaños de pantalla y orientaciones como móviles, tablets u ordenadores, esto nos asegura una experiencia de usuario coherente y agradable independientemente de sus dimensiones o resoluciones; algo muy importante teniendo en cuenta que es una *aplicación multiplataforma*.

De forma general, estas decisiones arquitectónicas contribuyen a un sistema robusto, escalable y preparado para evolucionar en el tiempo con mínimos costes de refactorización.

* Cliente Multiplataforma (Flutter + Dart)

La aplicación se desarrolla con Flutter utilizando el lenguaje Dart, lo que permite la compatibilidad multiplataforma (Android, iOS, escritorio (Windows, macOS y Linux) y en el futuro Web. También gestiona la interfaz de usuario, la lógica de presentación, el almacenamiento local a través de SQLite y la sincronización de datos por medio del protocolo HTTP seguro para autenticación.

* API Backend (Rust + Axum)

La API está implementada en Rust utilizando el "framework" Axum para crear endpoints seguros y eficientes.

* Base de Datos (MySQL)

Esta base de datos almacena de forma persistente y segura la información relativa a usuarios, hábitos, metas, logros y configuraciones, asegurando su integridad y consistencia de manera asíncrona y segura utilizando SQLx.

* Servicios remotos y Virtualización

Se utiliza Docker para virtualizar tanto la base de datos MySQL como la API en Rust, encapsulando cada uno en contenedores diferentes. Esto permite una arquitectura más modular, escalable y portable; en la cual cada componente del sistema se ejecuta en su propio entorno aislado, evitando conflictos de dependencias y facilitando la administración.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **Plan de pruebas**

El plan de pruebas define la estrategia que se seguirá para validar el correcto funcionamiento de ZenHabits antes de su puesta en producción. Su objetivo es asegurar que la aplicación cumpla con los requisitos funcionales, no funcionales y proporcione una buena experiencia de usuario.

* + 1. **Pruebas Funcionales**
* Gestión de contenidos: Verificar que el usuario puede crear, editar y eliminar hábitos y metas, y que dichos cambios se reflejan tanto en la interfaz como en la persistencia de datos.
* Notificaciones y logros: Asegurar que al asignar recordatorios estos se generan correctamente y, al completar acciones definidas, se desbloquean los logros asociados.
* Personalización del personaje: Confirmar que la modificación del avatar (nivel o atributos) se actualiza de forma coherente y persistente.
  + 1. **Pruebas No Funcionales**
* *Compatibilidad multiplataforma*: Realizar pruebas en diferentes dispositivos y resoluciones (Android, iOS, escritorio y Web) para garantizar que la experiencia del usuario es consistente y los contenidos se adaptan correctamente.
* *Funcionamiento offline*: Comprobar la disponibilidad de datos mediante el almacenamiento local, asegurando que la aplicación funciona sin conexión y sincroniza correctamente cuando se restablece la conectividad.
* *Seguridad*: Evaluar el mecanismo de autenticación y autorización basado, verificando que sólo usuarios autorizados pueden acceder a la información.
* *Rendimiento*: Ejecutar pruebas de carga en la API y del sistema general para asegurar que la aplicación responde de forma fluida, incluso en condiciones de uso intensivo o en dispositivos con recursos limitados.
  + 1. **Pruebas Técnicas**
* **Pruebas de API (usando Postman):**
  + *Validación de Endpoints*: Comprobar que cada endpoint responde según lo esperado: operaciones: (actualizar e insertar, leer, eliminar) sobre usuarios, hábitos, metas, y demás datos.
  + *Manejo de Errores y Seguridad*: Verificar que las solicitudes no autorizadas se rechazan y que el manejo de errores es correcto (por ejemplo, mensajes de error claros cuando se envían datos incorrectos).
  + *Pruebas de Carga y Rendimiento*: Simular múltiples peticiones concurrentes para evaluar el tiempo de respuesta, la estabilidad y la robustez de la API (esto se puede hacer con un script en Python para ahorrar tiempo).
* **Pruebas de Base de Datos:**
  + *Integridad de Datos*: Comprobar que las operaciones realizadas desde la API (insertar, actualizar o eliminar) se reflejan correctamente en la base de datos y se mantienen las relaciones y restricciones definidas.
  + *Persistencia y Sincronización*: Verificar que la base de datos (MySQL en contenedor Docker) maneja correctamente la persistencia de la información y permite su recuperación sin errores.
  + *Pruebas de Recuperación*: Simular escenarios de fallo (por ejemplo, la pérdida de conexión) y comprobar que, al restablecerse, la sincronización se reanuda sin pérdida de datos.
  + *Escalabilidad*: Ejecutar pruebas que simulen un aumento en el volumen de transacciones, asegurando que la base de datos mantiene la integridad y rendimiento esperado.
    1. **Pruebas de Interfaz y Experiencia de Usuario (UI/UX)**
* *Usabilidad*: Realizar pruebas con usuarios reales, donde se observe la facilidad para navegar por la aplicación, localizar funciones y realizar acciones de manera intuitiva.
* Verificar que la interfaz se ajusta adecuadamente a diferentes tamaños y orientaciones de pantalla.
* Asegurarse de que cada interacción (como pulsar botones o realizar gestos) produzcan respuestas visuales o sonoras que confirmen dicha acción.

Una vez completadas todas estas pruebas de manera satisfactoria y se corrijan las incidencias detectadas, el sistema podrá considerarse preparado para su despliegue en producción. La documentación de cada fase de pruebas, junto con informes de rendimiento y de usabilidad, servirá para dar el visto bueno final al sistema.

# **Base de datos**

## **Contexto de la BD**

La base de datos es una parte muy importante de una aplicación, ya que permite la persistencia de la información relacionada con hábitos, metas, configuraciones y logros de los usuarios.  
En el caso de ZENHABITS se utilizarán dos entornos de almacenamiento:

* Almacenamiento local (SQLite): Permite el funcionamiento offline de la aplicación desde cualquier dispositivo.
* Almacenamiento remoto (MySQL): Centraliza los datos para su sincronización y disponibilidad desde múltiples dispositivos a través de la API en Rust.

Como se menciona en apartados anteriores, la gestión de la base de datos remota se realizará mediante SQLx en Rust, proporcionando operaciones seguras, asincrónicas y eficientes. Tanto la API RESTful desarrollada en Rust como el servidor de base de datos MySQL estarán dockerizados, lo cual facilita su despliegue, mantenimiento y escalabilidad en distintos entornos.

Además, la arquitectura híbrida de almacenamiento (SQLite local en el dispositivo y MySQL remoto) permite ofrecer una experiencia de usuario fluida y continua, incluso cuando no hay conectividad de red.

## Diagrama El contenido generado por IA puede ser incorrecto. **Entidades y MER (modelo Entidad Relación)**

* **Usuario:** Representa a cada usuario registrado en la aplicación.
* Atributos: Id\_usuario, Nombre, Email, Password\_hash.
* **Personaje:** Representa el avatar o personaje que evoluciona según los progresos del usuario.
* Atributos: Id\_personaje, Vida, Monedas, Experiencia, Id\_usuario (clave foránea).
* **Hábitos:** Representa los hábitos que el usuario desea mantener.
* Atributos: Id\_hábito, Nombre, Descripción, Frecuencia, Id\_usuario (clave foránea).
* **Metas:** Representa metas u objetivos mayores que el usuario desea alcanzar.
* Atributos: Id\_meta, Título, Descripción, Fecha\_límite, Id\_usuario (clave foránea).
* **Logros:** Representa los logros desbloqueados por los usuarios al cumplir objetivos o alcanzar hitos dentro de la aplicación.
* Atributos: Id\_logro, Nombre, Descripción, Id\_usuario (clave foránea).

Usuario → puede tener múltiples → Hábitos, Metas, Logros. *(1:N)*

Usuario → posee → Personaje *(1:1)*

Usuario → obtiene → Logros al cumplir tareas, hábitos o metas. *(1:N)*

# **Interfaz**

La interfaz de *ZENHABITS* se ha diseñado pensando en la simplicidad, accesibilidad y eficiencia de uso. El objetivo es ofrecer una experiencia coherente en todas las plataformas (Android, iOS, Web y Escritorio), aunque las primeras referencias visuales se basen en el diseño móvil.

## **GUI**

Su Interfaz Gráfica de Usuario presenta una estética limpia y minimalista, que facilita la navegación del usuario. Los componentes visuales están organizados de forma jerárquica para priorizar las acciones más comunes, como consultar hábitos, crear nuevas tareas y gestionar metas personales.

### **UI**

La Interfaz de Usuario (UI) está compuesta por diferentes pantallas o vistas:

* Pantalla de Login/Registro: Introducción de credenciales para acceso seguro.
  + Pantalla para iniciar sesión y pantalla para crear cuenta.
* Pantalla Principal Gestión de hábitos: Muestra hábitos activos y acceso rápido a crear nuevos hábitos.
  + Pantalla con el formulario de creación de hábitos.
* Pantalla de Gestión de metas: Se podrán visualizar y navegar al formulario para añadir nuevas metas.
  + Pantalla con el formulario de creación de metas.
* Menú de navegación inferior: Accesos rápidos a hábitos, tareas, metas y perfil de usuario.
* Interfaz de usuario gráfica, Aplicación

  El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación

  El contenido generado por IA puede ser incorrecto.Pantalla de configuración: Perfil o notificaciones entre otras cosas.

### **UX**

La experiencia de usuario en *ZENHABITS* se centra en la simplicidad, la claridad y la motivación del usuario. Está diseñada para:

* Formularios: cortos y guiados, con validaciones claras para evitar errores.
* Navegación intuitiva: Todos los accesos principales están siempre visibles gracias a una barra de menú inferior y a botones flotantes.
* Retroalimentación inmediata: Cada acción muestra una respuesta (visual) que confirma su éxito o advierte en caso de error.

## **Diagrama de navegación**

Diagrama

El contenido generado por IA puede ser incorrecto.Este diagrama representa la estructura y flujo entre las pantallas principales de la aplicación en su versión móvil. Aunque en escritorio/Web se adaptará con menús laterales u otras disposiciones, la lógica de navegación se mantiene.

## **Características visuales**

### **Paleta de colores**

| **Uso específico** | **Color** | **Hexadecimal** | **Descripción** |
| --- | --- | --- | --- |
| **Fondo general** | Crema | #FFEAD2 | Color base de toda la interfaz, fondo de todas las pantallas. |
| **Botones principales y logotipo** | Naranja | #F78A07 | Usado en botones de acción principal como “Iniciar sesión” y el logo. |
| **Botones secundarios** | Naranja claro | #F9BD76 | Utilizado para acciones secundarias como “Cancelar” o “Volver”. |
| **Cuadros de texto y etiquetas** | Blanco | #FFFFFF | Aplicado en inputs, tarjetas de hábitos y metas, para claridad visual. |
| **Texto de títulos y botón de creación** | Marrón oscuro | #582105 | Color fuerte que resalta los títulos, botones de “Crear” o “Añadir”. |
| **Menú de navegación inferior y botón back** | Marrón medio | #AA5C21 | Base para la barra inferior, íconos del menú y el botón de retroceso. |

### **Tipografía**

La aplicación *ZENHABITS* utiliza una tipografía sans-serif, simple y amigable para reforzar su carácter accesible y relajado:

* Títulos: tamaño grande (entre 20 y 24 píxeles) y con estilo negrita, en color marrón oscuro (#582105), para destacarse claramente del resto del contenido.
* Texto de botones: tamaño medio (16 píxeles), también en negrita, con colores de alto contraste dependiendo del fondo: blanco sobre botones naranjas y marrón sobre botones claros.
* Textos secundarios o de apoyo: (como descripciones, etiquetas o formularios) tienen un tamaño más pequeño (14 píxeles), sin resaltado en negrita y también en color marrón oscuro.

Gracias a estas características de la tipografía se garantiza una lectura clara y coherente en toda la aplicación.

### **Estilo general**

* **Estética**: Simple, cálida y amigable
* **Forma predominante**: Bordes redondeados en todos los elementos.
* Íconos: Estilo flat, amigables, coherentes con el estilo zen.
* **Espaciado**: Uso generoso de espacio para evitar sobrecarga visual.

## **Usabilidad**

Esta aplicación ha sido diseñada con un enfoque centrado en el usuario, priorizando la simplicidad, la claridad visual y la facilidad de uso. Su estructura intuitiva permite que cualquier persona, independientemente de su experiencia tecnológica, pueda crear, gestionar y realizar seguimiento de hábitos y metas personales de manera efectiva.

La navegación se basa en un menú inferior fijo con íconos reconocibles que permiten acceder rápidamente a las secciones principales de la app (hábitos, estadísticas, perfil, etc.). Los botones son grandes, contrastados y están etiquetados de forma clara, lo que mejora la accesibilidad visual. Además, se utilizan colores cálidos y una jerarquía visual bien definida, con títulos destacados y espacios generosos, lo que evita la sobrecarga cognitiva.

Los formularios para crear o editar hábitos han sido simplificados al máximo, minimizando el número de pasos y campos requeridos. Los mensajes visuales de confirmación, el uso de íconos familiares y el diseño limpio refuerzan la experiencia de usuario y reducen el margen de error.

Para facilitar aún más el uso correcto de la aplicación, se ha desarrollado un manual de usuario completo, que explica detalladamente cada funcionalidad. Este manual se encuentra disponible como documento externo en el repositorio del proyecto en GitHub (ver Anexos).

# **Implementación**

Para el desarrollo de ZENHABITS se han utilizado tecnologías actuales de desarrollo multiplataforma y backend, con almacenamiento local y remoto (dockerizado).

## **Entorno de desarrollo**

* Entornos de desarrollo/herramientas:
* Visual Studio Code:
  + Versión: 1.100.2
  + Extensiones: Flutter: Dart code, Dart: Dart code, Rust Analycer, Git Extension Pack, Marckdown All in One, Marckdown preview, PlantUML
* Android Studio:
  + Versión:
  + Emulador: Pixel 6 API 35 |x86\_64
* Visual Studio:
  + Versión:
* Docker Desktop:
  + Versión: 28.0.4
* Git:
  + Versión: 2.34.1
* Sistemas operativos: Android, iOS, Windows y Linux.

## **Tecnologías**

| **Elemento** | **Tecnología** | **Función principal** |
| --- | --- | --- |
| **App multiplataforma** | ***Flutter (Dart)*** | UI adaptativa para Android, iOS, Web y Escritorio |
| **Base de datos local** | ***SQLite*** | Persistencia local de datos |
| **API Backend** | ***Rust + Axum*** | Creación de endpoints seguros y eficientes |
| **ORM y acceso a datos** | ***SQLx*** | Conexión segura y asíncrona |
| **BD remota** | ***MySQL*** | Gestión relacional de datos |
| **Contenedores** | ***Docker*** | Aislamiento y despliegue de servicios |
| **Control de versiones** | ***Git*** | Gestión de versiones, ramas y seguimiento de código |
| **Plataforma de desarrollo** | ***GitHub*** | Almacenamiento y gestión del proyecto |
| **Pruebas de API** | ***Postman*** | Testeo de peticiones y respuestas HTTP |

* **Flutter (Dart)**: Framework para crear una app única que funcione en Android, iOS, Web y escritorio en Dart, que es el lenguaje de programación para Flutter.
* **SQLite**: Motor de base de datos ligero para guardar datos localmente sin conexión.
* **Rust + Axum**: Backend seguro y rápido, ideal para crear APIs REST
* **Cargo**: Herramienta de gestión para proyectos en Rust.
* **SQLx**: Librería Rust para trabajar con bases de datos SQL de forma segura.
* **MySQL**: Sistema relacional de datos ejecutado en contenedor para facilitar despliegue y mantenimiento.
* **Docker**: contenedores para MySQL y despliegue futuro de la API.
* **Git**: Control de versiones para el desarrollo del código.
* **GitHub**: Plataforma para almacenar el código en un repositorio y gestionar el proyecto.
* **Postman**: pruebas manuales de la API REST.

## **Funcionamiento**

**Ejecución capturas y capturas de pruebas y esas cosas:** No se como debería ponerlas, directamente las capturas? Como lo explico, no entiendo este apartado. Añado mensajes de errores gestionados para el usuario?¿

## **Puesta en producción**

Una vez finalizado el desarrollo y completadas las fases de pruebas, se continua con la puesta en producción o implantación, tanto del backend como de la aplicación móvil, asegurando su funcionamiento en un entorno real y accesible para usuarios externos.

*La parte del backend*, desarrollada enRust, fue contenedorizado mediante Docker, lo que facilitó su despliegue en un entorno limpio y replicable. Junto a él, *la base de datos MySQL* también se ejecuta dentro de un contenedor separado, lo que permite mantener el aislamiento y la portabilidad del sistema completo. Esta arquitectura fue diseñada para ejecutarse tanto en entornos locales como en servidores remotos compatibles con Docker.

*Aplicación móvil*

La aplicación fue desarrollada con Flutter y está diseñada para ejecutarse en dispositivos móviles Android e iOS. Para su distribución, existen dos vías posibles:

* Distribución directa (modo pruebas o beta):  
  El archivo .apk (Android Package) puede ser generado y compartido manualmente con los usuarios para su instalación directa en dispositivos Android. Este archivo puede enviarse por correo, alojarse en un repositorio privado o compartirse mediante plataformas como Firebase App Distribution o Google Drive. El usuario debe habilitar la opción de "instalación desde orígenes desconocidos" en su dispositivo para poder instalarla.
* Publicación en tienda (Play Store o App Store):  
  En una versión futura, la aplicación puede ser publicada oficialmente en Google Play o App Store, siguiendo los procesos de revisión y firma de cada plataforma. Esto permitiría una instalación automática y actualizaciones gestionadas desde la propia tienda.

*Acceso y uso por parte del usuario*

Para utilizar la aplicación, el usuario debe instalarla en su dispositivo móvil (mediante APK o desde la tienda). Una vez instalada, la app se conecta automáticamente con el backend desplegado (configurado en la app mediante la base URL del servidor) y puede comenzar a utilizar todas sus funcionalidades.

El sistema está diseñado para funcionar tanto online como offline, gracias al uso de SQLite como base de datos local. Esto garantiza que el usuario pueda continuar utilizando la aplicación incluso sin conexión a internet, sincronizándose nuevamente cuando se recupere la conectividad.

## **Líneas futuras**

Con esta base sólida, el proyecto está preparado para una evolución progresiva. Entre las siguientes fases previstas, destacan:

* *Escalabilidad y Orquestación*
* Kubernetes será incorporado para la orquestación de contenedores, permitiendo escalado automático, balanceo de carga y alta disponibilidad.
* Se plantea la integración de servicios como Prometheus y Grafana para la monitorización del rendimiento y el uso de recursos.
* El servidor, podrá alojarse en un proveedor cloud o en una máquina con Docker instalado. La ejecución del backend se realiza mediante docker-compose, lo que levanta tanto la API como la base de datos de forma coordinada. El sistema expone sus endpoints REST, que son consumidos por la aplicación.
* *Sistema de Notificaciones Personalizadas*
* Desarrollo de un sistema de notificaciones basado en eventos, adaptable a las preferencias del usuario.
* Las notificaciones incluirán tanto eventos importantes del sistema como interacciones personalizadas (recordatorios, logros obtenidos, etc.).
* *Personalización del Personaje*
* Se habilitará la personalización visual del personaje, incluyendo selección de apariencia, ropa, y mascotas u objetos acompañantes.
* Este módulo permitirá reforzar la conexión del usuario con la aplicación mediante elementos de gamificación.
* *Optimización y Modularización*
* Separación por módulos funcionales (microservicios) para mejorar mantenibilidad, pruebas y despliegues independientes.

# **Elementos destacables del desarrollo**

## **Innovaciones**

*Backend con Rust y MySQL dockerizado*

El backend fue desarrollado utilizando Rust, una decisión estratégica basada en la necesidad de una buena seguridad en memoria y concurrencia eficiente. Aunque Rust no era una tecnología previamente dominada, su elección se debe a una visión a largo plazo y a creciente adopción en entornos de producción críticos.

Comparado con alternativas como:

* Python (Django/Flask): Python ofrece rapidez de desarrollo inicial, pero implica un mayor uso de recursos en ejecución y menor control de bajo nivel. Rust, en cambio, permite un uso más eficiente del hardware y evita errores comunes gracias a su sistema de tipos y control de propiedad.
* Node.js: Aunque Node.js es conocido por su rendimiento en aplicaciones I/O intensivas, depende de un modelo de event-loop y un recolector de basura que, en ciertos casos, introduce latencias y complejidades adicionales. Rust, con sus herramientas como tokio, ofrece concurrencia segura y predecible sin sacrificar rendimiento.

El lenguaje Rust fue complementado con Docker, permitiendo encapsular tanto la API como la base de datos MySQL en contenedores aislados. Yo utilicé Docker Compose para orquestar tanto el contenedor del backend, como el de la base de datos. Esto me permitió levantar toda la infraestructura del proyecto con un solo comando (docker-compose up). Esta elección ofreció múltiples beneficios:

* Entorno de desarrollo reproducible, evitando inconsistencias entre equipos.
* Despliegue simplificado y adaptable a cualquier entorno, local o en la nube, en mi caso actualmente, local.
* Aislamiento de dependencias y reducción del riesgo de errores por conflictos de configuración.

En cuanto a la base de datos, se optó por MySQL frente a PostgreSQL por varias razones:

* MySQL presentó mejor rendimiento en operaciones simples y de lectura intensiva, que eran prioritarias en este proyecto.
* Su compatibilidad con las bibliotecas Rust disponibles fue más directa y estable en el momento del desarrollo.
* PostgreSQL, si bien más avanzado para casos complejos, implicaba una curva de configuración innecesaria para los requerimientos inmediatos.

*Aplicación móvil con Flutter, Dart y SQLite (Floor)*

Para la interfaz móvil se eligió Flutter, en combinación con el lenguaje Dart y la base de datos local SQLite gestionada con la librería Floor.

Flutter fue seleccionado frente a otras opciones multiplataforma como:

* React Native: Aunque ampliamente utilizado, React Native depende de un puente entre JavaScript y código nativo, lo que puede generar problemas de rendimiento y complejidad en animaciones y personalizaciones UI. Flutter, en cambio, utiliza su propio motor de renderizado, garantizando una experiencia visual fluida y consistente.
* Kotlin Multiplatform: Aunque permite compartir lógica de negocio, aún requiere desarrollar interfaces separadas para Android e iOS. Flutter ofrece una solución completamente unificada que acelera el desarrollo y facilita el mantenimiento.

Dart, como lenguaje, ofreció:

* Sintaxis clara y tipado estático, facilitando el mantenimiento y la detección de errores en tiempo de compilación.
* Excelente integración con Flutter, reduciendo la fricción entre lenguaje y framework.

La persistencia local se implementó con SQLite, gestionado mediante la librería Floor, una solución que combina simplicidad y potencia:

* Floor sigue el patrón DAO y permite trabajar con consultas SQL reales, manteniendo control total sobre la base de datos.
* Se eligió sobre Hive, qué aunque es muy rápida, no está basada en SQL y tiene limitaciones para relaciones complejas.
* También se prefirió sobre Moor (ahora Drift), que ofrece una capa de abstracción poderosa pero más compleja de configurar, lo que no se adecuaba al plazo del proyecto.

Floor, además, presenta similitudes con Room de Android, lo cual facilitó su adopción y redujo el tiempo de aprendizaje ya que ya lo conocía.

## **Problemas**

### **Dificultades**

* **Curva de Aprendizaje:**

Para realizar este proyecto, decidí innovar en cuanto a mis conocimientos en todas las tecnologías; esto generó una curva de aprendizaje dura e intensa las primeras semanas. Para superar esta dificultad, dediqué tiempo a revisar la documentación oficial, páginas de información y a realizar pruebas que me ayudaron a comprender mejor el funcionamiento de estas herramientas y a adoptar buenas prácticas de desarrollo. Las tecnologías son las siguientes:

* + **Flutter**: Esta fue la primera nueva tecnología a la que me enfrenté, junto con Dart, el lenguaje recomendado. Al ser un framework relativamente nuevo para mi, tuve que familiarizarme con su estructura basada en widgets y el manejo de estado dentro de las aplicaciones.
  + **Rust**: Representó el mayor reto, ya que su enfoque en la seguridad y manejo de la memoria es complejo si no estás acostumbrado a lenguajes de más bajo nivel, como es mi caso. Para entender su funcionamiento, fue necesario dedicar mucho tiempo a leer documentación oficial, analizando ejemplos y escribiendo código de prueba, pudiendo así aprovechar ciertas ventajas de Rust como la prevención de errores en tiempo de compilación y el alto rendimiento.
  + **Docker**: Lo conocía con anterioridad, pero no lo había utilizado nunca. Su uso resultó ser relativamente sencillo una vez comprendidos los conceptos fundamentales; aunque al principio puede parecer complejo debido a la abstracción que introduce respecto al sistema operativo y las redes internas. Sin embargo, cuando se asientan bien las bases, su utilización se vuelve fluida y sencilla.
* **Contexto Técnico y Organizacional:**
  + Tiempo limitado: El desarrollo completo debía llevarse a cabo en unos tres meses, lo que obligó a priorizar funcionalidades críticas y dejar de lado otras menos importantes.
  + Presión externa y mala organización inicial: La falta de una planificación clara en las fases tempranas generó una carga de trabajo desigual, especialmente en el tramo final. A pesar de ello, se logró entregar una versión funcional y escalable.

Estos desafíos pusieron a prueba la adaptabilidad del equipo y reafirmaron la elección de tecnologías con buena documentación, alta comunidad y herramientas modernas de depuración y testing.

### **Errores**

* Emulador Android: Me surgieron problemas y errores al intentar ejecutar la aplicación en un emulador Android compatible. Debido a la integración de la librería Floor para la gestión de la base de datos local con SQLite, varios emuladores fallaban al iniciar o se comportaban de forma inestable. Tras múltiples pruebas, se conseguí que funcionase correctamente el emulador “Pixel 6” con API 35 (x86\_64).
* Errores de tipado:

Los errores de tipado fueron una constante durante la implementación, sobre todo al trabajar con lenguajes fuertemente tipados como Rust y Dart, así como al definir las relaciones con MySQL y con Floor con SQLite:

* *Rust y MySQL*: Uno de los problemas más destacados fue la incompatibilidad entre los tipos u32 (entero sin signo en Rust) y INT de MySQL, que sí permite valores negativos. Esta diferencia provocaba errores de deserialización al interactuar con la base de datos. La solución fue ajustar los modelos de datos en Rust, modificando los identificadores (id) a tipo i32, alineando así los tipos entre backend y base de datos y evitando conversiones manuales o pérdida de datos.
* *Fluter + Dart + Floor:* La librería Floor no admite directamente tipos DateTime, lo cual dificultó el uso de dos campos de fecha dentro de la entidad Habits. Para solventarlo, necesité implementar un TypeConverter personalizado que transformase las fechas en milisegundos desde la época Unix (tipo int), lo que permitió almacenar las fechas de forma válida en SQLite y convertirlas de nuevo a DateTime al leerlas.
* Docker:

Aunque Docker facilitó la configuración y despliegue de servicios, también se presentaron varios inconvenientes menores que afectaron al flujo de trabajo y consumieron tiempo de depuración:

* *Error por desigualdad de nombres*: en cuanto al nombre de la base de datos, utilizar un nombre en mayúsculas en una parte de la ruta y minúsculas en otra, provocaba fallos de conexión.
* *Puerto en uso*: En más de una ocasión, el puerto 3306, utilizado por MySQL, se encontraba ocupado debido a que un contenedor anterior seguía corriendo en segundo plano. Esto impedía reiniciar correctamente los servicios. La solución pasó por revisar los contenedores activos (docker ps) y detener manualmente los procesos residuales.
* *Errores en rutas relativas en Dockerfile*: Al construir las imágenes Docker, se produjeron fallos al copiar carpetas debido a rutas relativas incorrectas. Tras experimentar y consultar documentación, logré entender las rutas relativas bien y establecer correctamente las rutas en COPY.

# **Conclusiones**

El desarrollo de esta aplicación supuso todo un reto, pero al final se logró alcanzar un resultado funcional, eficiente y escalable.

El proceso general ha tenido una curva de aprendizaje elevada, cuyo proceso a sido entretenido pero duro. Tuve que trabajar con tecnologías desconocidas en un principio, como Rust, Flutter y Docker; provocando una inversión de tiempo considerable en formación y adaptación. Además, el desarrollo del proyecto tuve que completarlo en menos de tres meses, con presión constante por cumplir plazos y una organización interna que, en sus primeras fases, no estaba suficientemente definida; lo cual llevó a dejar de lado ciertas características importantes de la aplicación y dejarlas para un futuro.

También tuve en cuenta la cantidad de aplicaciones similares en funcionalidad, las cuales pueden afectar en la adopción de mi producto si no se diferencia lo suficiente, buscando soluciones creativas y llamativas para el usuario además de funcionales para sus necesidades.

A pesar de ello, el resultado evidencia importantes oportunidades de crecimiento futuro. La arquitectura actual no solo permite una fácil integración de funcionalidades adicionales, como notificaciones personalizadas para el usuario, personalización visual del personaje (apariencia, accesorios, mascotas), o incluso elementos de gamificación, sino que también se encuentra bien posicionada para migrar hacia un entorno orquestado con Kubernetes y microservicios. El hecho de haber trabajado con tecnologías emergentes se convierte así en una ventaja estratégica, ya que su adopción está en alza, lo que facilitará su mantenimiento, escalabilidad y adaptabilidad a futuro.

En resumen, este proyecto ha evidenciado tanto la viabilidad técnica como el potencial de evolución de la solución propuesta, demostrando que incluso con recursos limitados y bajo presión es posible construir productos robustos y modernos, siempre que se adopte una visión tecnológica bien fundamentada y se mantenga una actitud resiliente ante los imprevistos.

# **Bibliografía**

Dart Language Team. (13 de Abril de s.f.). Recuperado el 2025, de Dart Programming Language: https://dart.dev/guides

Docker, Inc. (s.f.). Recuperado el 28 de Mayo de 2025, de Docker Documentation: https://docs.docker.com/

Duhigg, C. (2012). *The power of habit: Why we do what we do in life and business.* Nueva York: Random House.

Google. (3 de Mayo de s.f.). Recuperado el 2025, de Flutter documentation.: https://docs.flutter.dev/

Rust Project Developers. (25 de Mayo de s.f.). Recuperado el 2025, de The Rust Programming Language: https://doc.rust-lang.org/book/

# **Anexo 1**

# **Anexo 2**

# **Anexo 3**

**REPOSITORIO PÚBLICO EN GITHUB:** [**https://github.com/albealvant/zenhabits-project**](https://github.com/albealvant/zenhabits-project)