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 (índice en proceso)

1. Contexto de la aplicación

1.1. Análisis del problema

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, el olvido o la falta de motivación. A esto se le añade la escasez de herramientas eficaces que respondan a esa necesidad. Buscar esa investigación y añadirla a la bibliografía

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.

1.2. 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/mantenga un hábito, será recompensado mediante logros y avances visibles en un personaje personalizable. 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.

2. 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 en la nube. 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.

2.1. 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:

2.1.1. Requisitos funcionales

- Crear, modificar y eliminar hábitos y metas.
- Asignar notificaciones para recordar hábitos y metas.
- Desbloquear logros en base al cumplimiento de hábitos y metas.

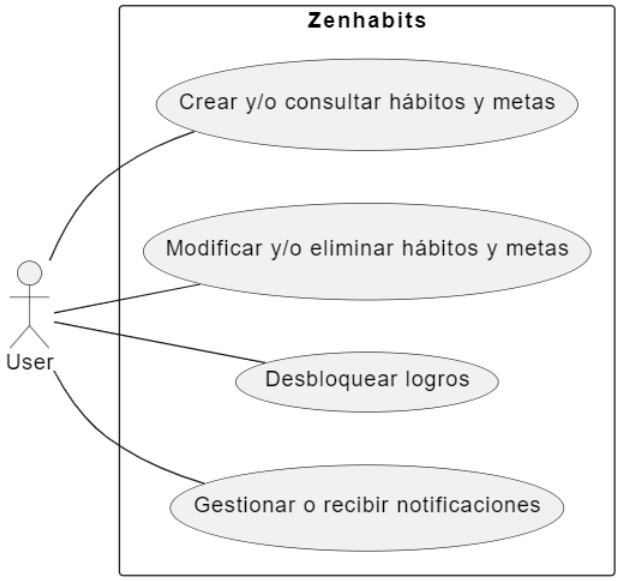
2.1.2. Requisitos no funcionales

- Usabilidad
  - 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.

2.2. Casos de uso

Los casos de uso, derivados de los requisitos funcionales, muestran la manera en que los "actores" interactúan con el sistema, estableciendo las funcionalidades principales desde el punto de vista de quien utiliza la aplicación.



En el caso de ZENHABITS, el actor principal es el usuario, el cual es la persona que interactúa con dicha aplicación. Este puede iniciar sesión; añadir, editar o eliminar hábitos o metas; desbloquear logros; personalizar a su personaje/avatar y gestionar o recibir notificaciones.

3. Diseño

3.1. 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 e iOS) como en escritorio (Windows, macOS y Linux) y Web (navegadores modernos) en fases futuras.

La solución sigue un modelo cliente-servidor. Se estructura internamente bajo los principios de Clean Architecture, permitiendo separación clara de toda su estructura. Además, se utilizarán prácticas de diseño adaptativo para garantizar interfaces "responsive" en distintos dispositivos y tamaños de pantalla. Explicar clean architecture y MVVM (dar créditos en bibliografía)

3.1.1. Despliegue

- Cliente Multiplataforma (Flutter + Dart)

  La aplicación se desarrolla con Flutter en Dart, lo que posibilita la compatibilidad multiplataforma (Android, iOS, escritorio (Windows, macOS y Linux) y 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 utilizando JWT para autenticación.

- API Backend (Rust + Axum) docketizada

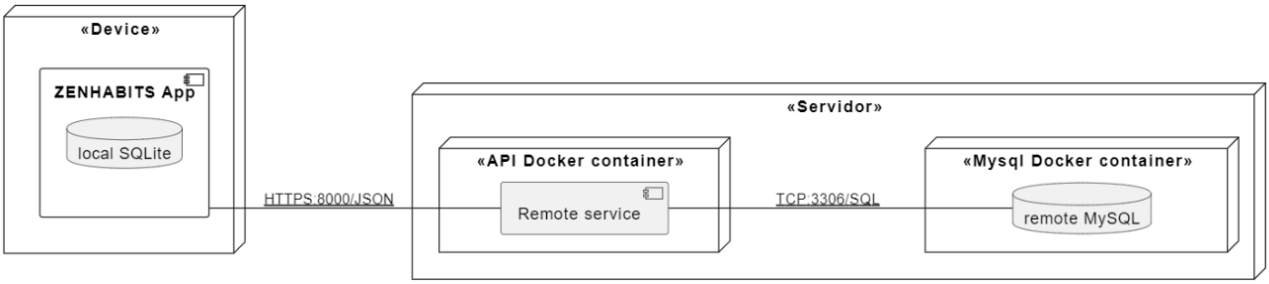
  La API está implementada en Rust utilizando el "framework" Axum para crear endpoints seguros y eficientes. Además, se utiliza SQLx para interactuar de manera asíncrona y segura con la base de datos MySQL alojada en la nube y la autenticación de usuarios se realiza mediante JSON Web Tokens (JWT).

- Base de Datos (MySQL Docketizada)

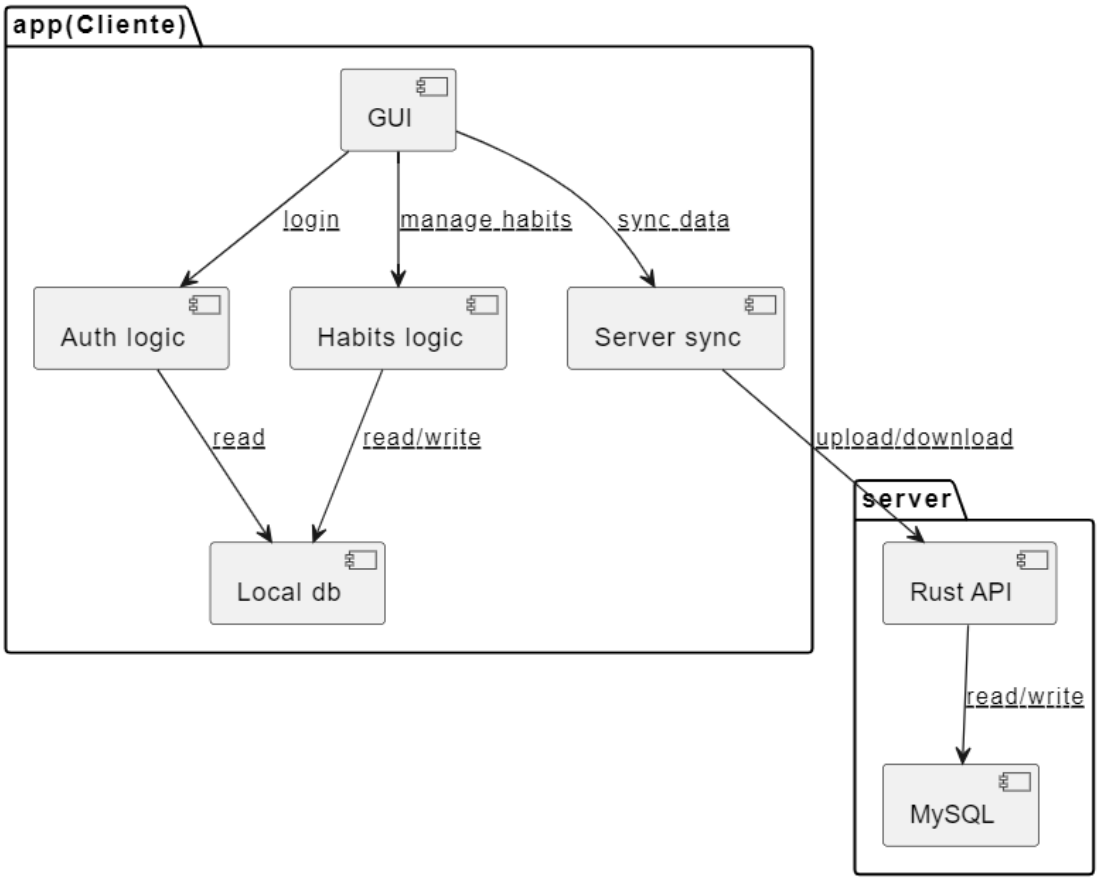
  La base de datos MySQL se ejecuta dentro de un contenedor Docker, lo cual garantiza un entorno aislado, reproducible y escalable. Esta base de datos almacena de forma persistente y segura la información relativa a usuarios, hábitos, metas, logros y configuraciones, asegurando su integridad y consistencia.

- Servicios remotos y Virtualización

  A futuro, se plantea ampliar el uso de Docker para incluir, además de la base de datos MySQL y la API en Rust, otros componentes del sistema, consiguiendo una arquitectura más modular y escalable. Además, también se plantea la integración de notificaciones y el servidor en la nube a futuro.



3.1.2. Componentes



***Sigo sin entender bien este diagrama, probablemente esté mal, ayuda.**

Quiero poner en anexos otros diagramas y tal, pero no sé cómo poner aquí que está en x anexo y poner los anexos, no entiendo

3.2. 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.

3.2.1. 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.

3.2.2. 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 en JWT, 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.

3.2.3. Pruebas Técnicas

- Pruebas de API (usando Postman):
    - Validación de Endpoints: Comprobar que cada endpoint responde según lo esperado: operaciones CRUD (crear, leer, actualizar, 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)
- 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.

3.2.4. 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.

4. Base de datos

4.1. 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:

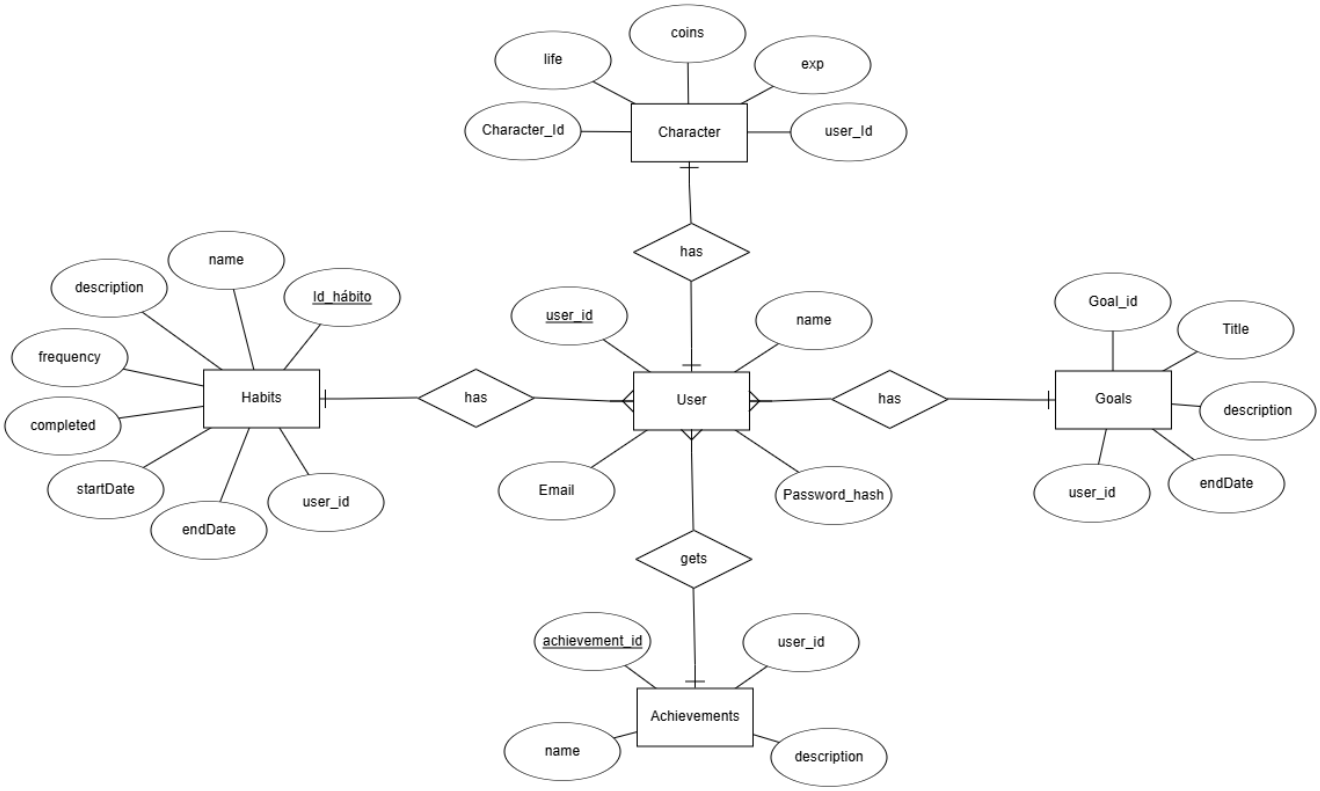
- Almacenamiento local (SQLite): Permite el funcionamiento offline de la aplicación desde cualquier dispositivo.
- Almacenamiento en la nube (MySQL): Centraliza los datos para su sincronización y disponibilidad desde múltiples dispositivos a través de la API en Rust.

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 en la nube) permite ofrecer una experiencia de usuario fluida y continua, incluso cuando no hay conectividad de red.

4.2. 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)*

5. Implementación

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

5.1. Entorno de desarrollo

- Entornos de desarrollo/herramientas:
  - Visual Studio Code:
    - Versión: 1.100.2 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 (para pruebas multiplataforma).

5.2. 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 con MySQL
Base de datos nube	MySQL	Gestión relacional de datos
Seguridad	JWT	Autenticación basada en tokens
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.
- **JWT**: Mecanismo de autenticación usando tokens cifrados.
- **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.

6. 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.

6.1. GUI

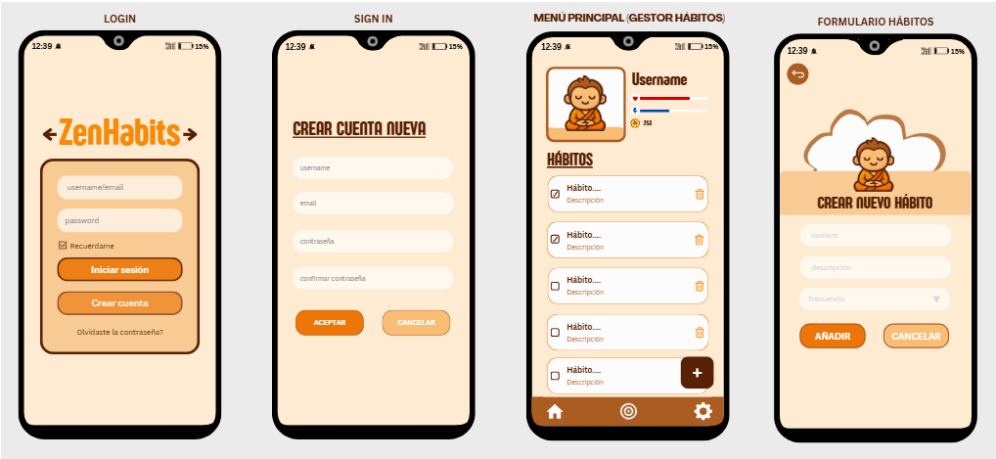
Su Interfaz Gráfica de Usuario presenta una estética limpia y minimalista, que facilita la navegación del usuario. Los componentes visuales están organizados de forma jerárquica para priorizar las acciones más comunes, como consultar hábitos, crear nuevas tareas y gestionar metas personales.

6.1.1. 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.
- Pantalla de configuración: Perfil o notificaciones entre otras cosas.



6.1.2. UX

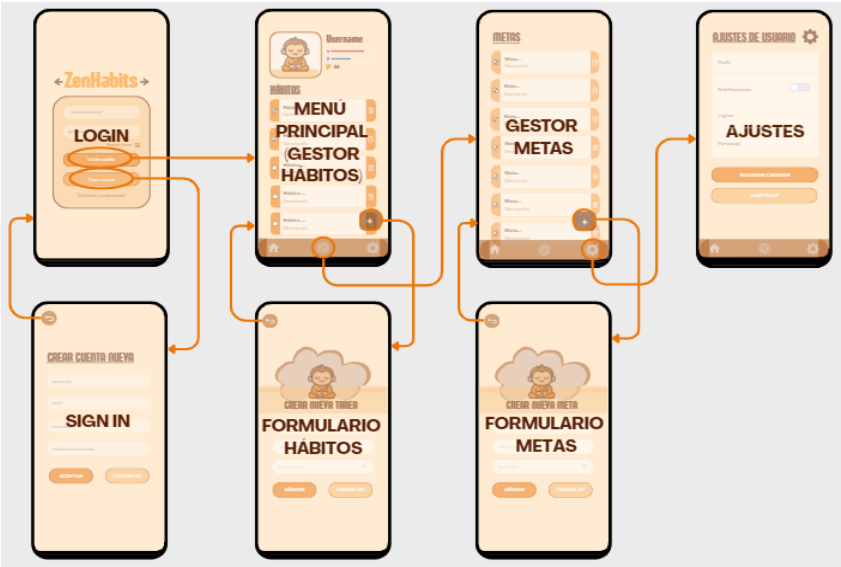
La experiencia de usuario en *ZENHABITS* se centra en la simplicidad, la claridad y la motivación del usuario. Está diseñada para:

- 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.

6.2. Diagrama de navegación

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.



6.3. Características visuales

6.3.1. 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	Blanco	#FFFFFF	Aplicado en inputs, tarjetas de hábitos y metas, para claridad

Uso específico	Color	Hexadecimal	Descripción
etiquetas			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.

6.3.2. 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.

6.3.3. 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.

6.4. 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).

7. Implementación

7.1. Puesta en producción

Qué pongo aquí, como o que debe hacer el usuario para descargar mi aplicación y usarla?

7.2. 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?¿

7.3. 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 integrarán servicios como Prometheus y Grafana para la monitorización del rendimiento y el uso de recursos.

- *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.

8. Elementos destacables del desarrollo

8.1. 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. 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.
- 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.

8.2. Problemas

8.2.1. 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 para mi 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.

- **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.
  - Apagón nacional en España: Un corte de suministro eléctrico a nivel nacional coincidió con una etapa clave del desarrollo, afectando el ritmo de trabajo y obligando a reorganizar tareas sobre la marcha.

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.

8.2.2. Errores

- Emulador Android: Tuve varios problemas a la hora de encontrar un emulador que fuera capaz de iniciar mi aplicación siendo compatible con la librería Floor y que fuera lo suficientemente potente. Al final utilicé el Pixel 6 API 35 | x86_64.
- Errores de tipado:
  Tanto en Rust como SQL como en Flutter con Dart y Floor.
  - Rust y MySQL: INT no es compatible con u32, por lo que tuve que cambiar los tipos de los id de mis structs de la API a i32, ya que INT si acepta negativos.
  - Fluter Dart + Floor: Mi entidad 'Habits' requiere dos fechas, las cuales quería que fueran DateTime porque es cómodo de uar, pero Floor no admite ese tipo, así qué implementé un convertidor, para pasarlo a milisegundos y que la db generada en Floor guarde un INT grande.

- Docker:
  Tuve varios errores no muy graves pero que hicieron que tardase más en completar mis tareas:
  - Errores de nombres que no coincidían por no estar atenta, como el nombre de la base de datos en minúsculas cuando estaba en mayúsculas ya que es una constante.
  - Error de puerto ocupado porque se quedó corriendo algo en segundo plano y no me di cuenta. El puerto que se quedó así fue el de la base de datos, el 3306.
  - Error al copiar carpetas por rutas en el Dockerfile: me cuesta bastante las rutas relativas, pero lo acabé sacando.

9. Conclusiones

No se que preguntarme aún para hacer esto

Debilidades:

Amenazas:

Fortalezas:

Oportunidades:

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/

Anexos

No entiendo, debería poner, ANEXO 1, ANEXO 2 ¿?

SI ES POSIBLE MIRA MI CÓDIGO POR ENCIMA O LO QUE SUELAS MIRAR, POR SI TE CHIRRÍA ALGO MUCHO, AÚN ESTÁ EN PROCESO, DEBO PROBAR SI CONECTA BIEN LA API CUANDO EL SERVIDOR ESTÉ INICIADO, TERMINAR DE AÑADIR LOGS y REFACTORIZAR EL CÓDIGO SEPARANDO COSILLAS Y CREANDO CONSTANTES PORQUE ESTÁ HECHO UN DESASTRE (estas dos últimas cosas están en local, por eso no las vas a ver en el repo)

REPOSITORIO PÚBLICO EN GITHUB: https://github.com/albealvant/zenhabits-project