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

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 (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.

## 1.1. 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 hábitos, lo cual quiere decir que lo mantiene, o consiga alcanzar sus metas, será recompensado mediante logros y avances visibles en un personaje, como por ejemplo un aumento de monedas que se podrán usar para personalizarle. 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 profesional.

# 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 remota. Para asegurar que 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, consultar y eliminar hábitos y metas.
- Gestionar notificaciones para recordar hábitos y metas.
- Desbloquear logros en base al cumplimiento de hábitos y metas.

### 2.1.2. Requisitos no funcionales

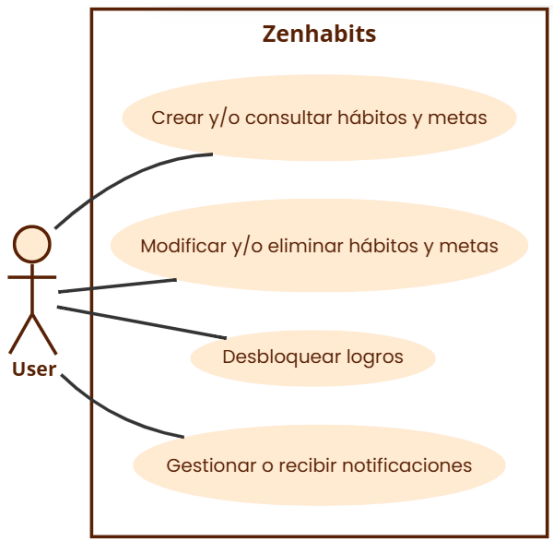
- UI:
  - La interfaz será intuitiva, clara y sencilla, permitiendo que cualquier usuario pueda manejarla sin necesidad de formación previa.
  - La experiencia de usuario se verá reforzada por un diseño visual minimalista, con iconografía reconocible y navegación fluida.
  - Se contempla una guía o ayuda 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 un diseño adaptativo (“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

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, los cuales son una de las recompensas que incentivan y motivan al usuario.



# 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 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, la aplicación se estructura 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 facilidad de probar el código.

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

Además, se aplicarán 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*.

- **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 remota por medio del protocolo HTTP seguro para autenticación.

- **Base de Datos Local (SQLite)**

  La base de datos local almacena los datos de forma segura además de persistente toda la información de los usuarios, hábitos y metas utilizando Floor, una librería ORM (Object Relational Mapping) para bases de datos SQLite. Está inspirada en Room, el ORM de Android.

- **API Backend (Rust + Axum)**

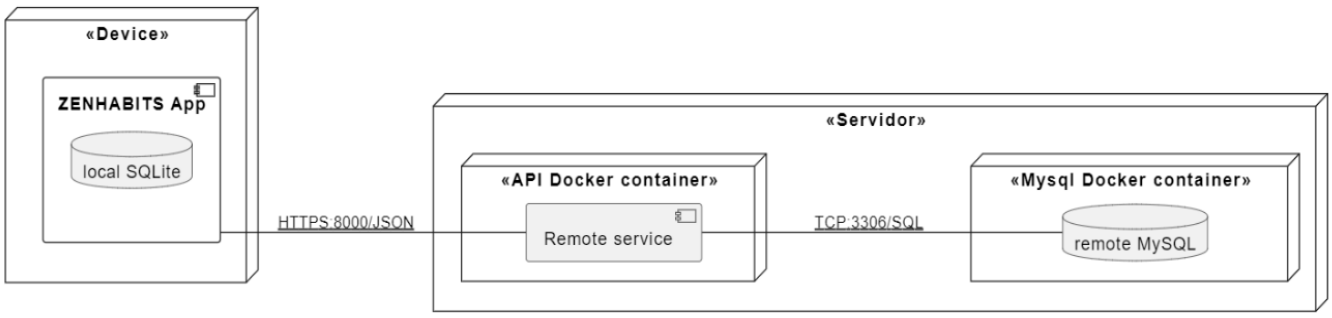
  La API está implementada en Rust utilizando el "framework" Axum para crear endpoints seguros y eficientes.

- **Base de Datos remota (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.



## 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, 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: (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.

### 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

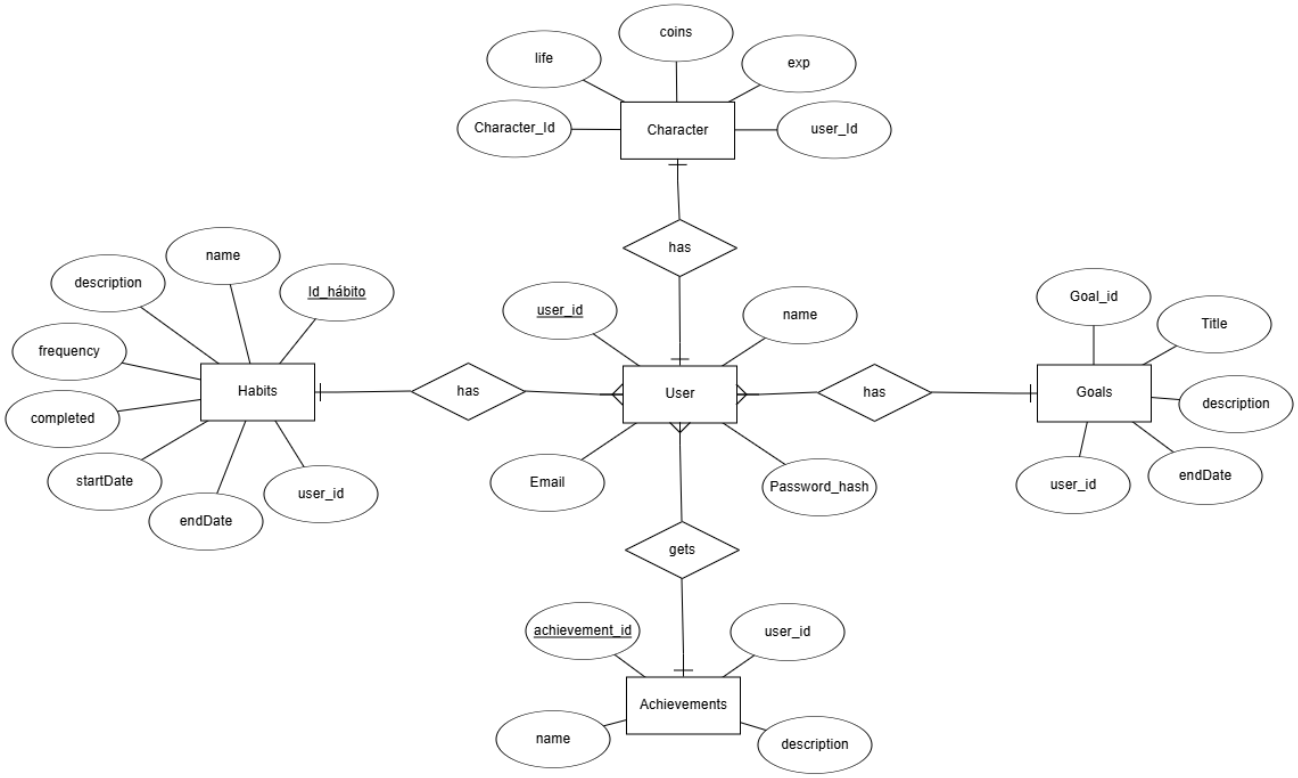
La base de datos es una parte muy importante de una aplicación, ya que permite la persistencia de la información relacionada con hábitos, metas, configuraciones y logros de los usuarios. En el caso de ZENHABITS se utilizarán dos entornos de almacenamiento:

- *Almacenamiento local (SQLite):* Permite el funcionamiento offline de la aplicación desde cualquier dispositivo.
- *Almacenamiento 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 REST 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 y MySQL remoto) permite ofrecer una experiencia de usuario fluida y continua, incluso cuando no hay conectividad de red.

## 4.2. Entidades y MER (modelo Entidad Relación)




Consultar Anexo II para ver el Esquema Relacional

- **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. 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.

## 5.1. GUI

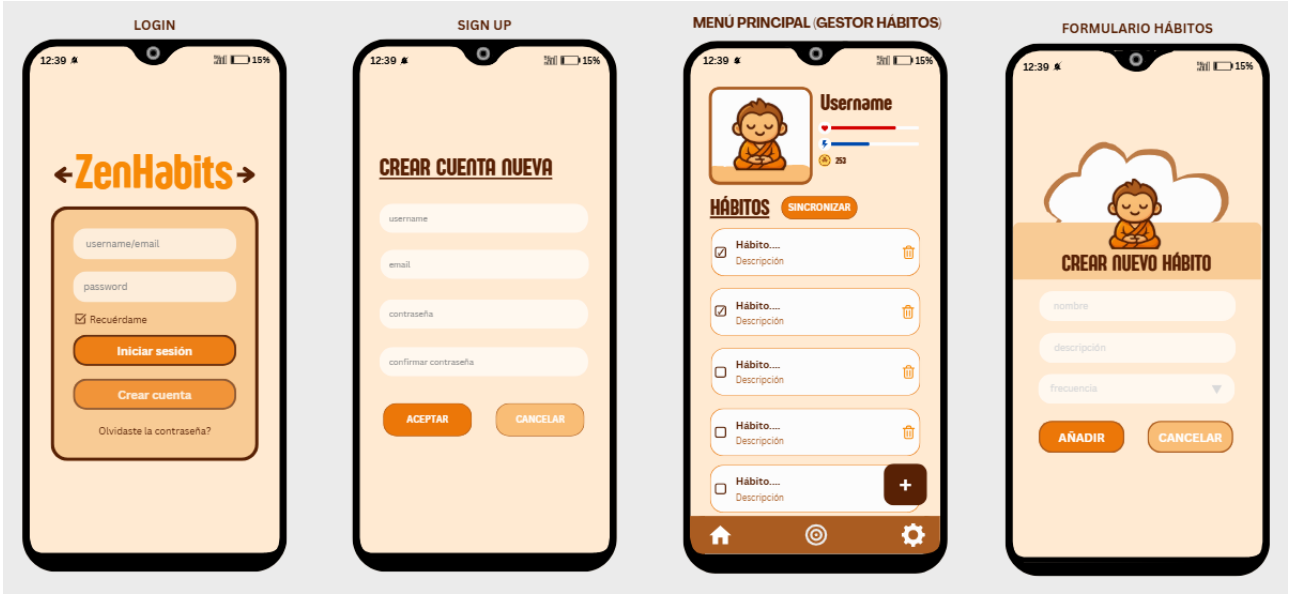
Su Interfaz Gráfica de Usuario presenta una estética limpia y minimalista, que facilita la navegación del usuario. Los componentes visuales están organizados de forma jerárquica para priorizar las acciones más comunes, como consultar hábitos, crear nuevas tareas y gestionar metas personales.

### 5.1.1. UI

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

- *Pantalla de Registro:* Introducción de datos necesarios.
    - Pantalla para crear una nueva cuenta de usuario.
- *Pantalla de Login:* Introducción de credenciales para acceso seguro.
    - Pantalla para iniciar sesión introduciendo nombre y contraseña.
- *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*: Se podrá configurar el perfil de usuario (datos como nombre, email o contraseña), gestionar las notificaciones, visualizar los logros y modificar al personaje).



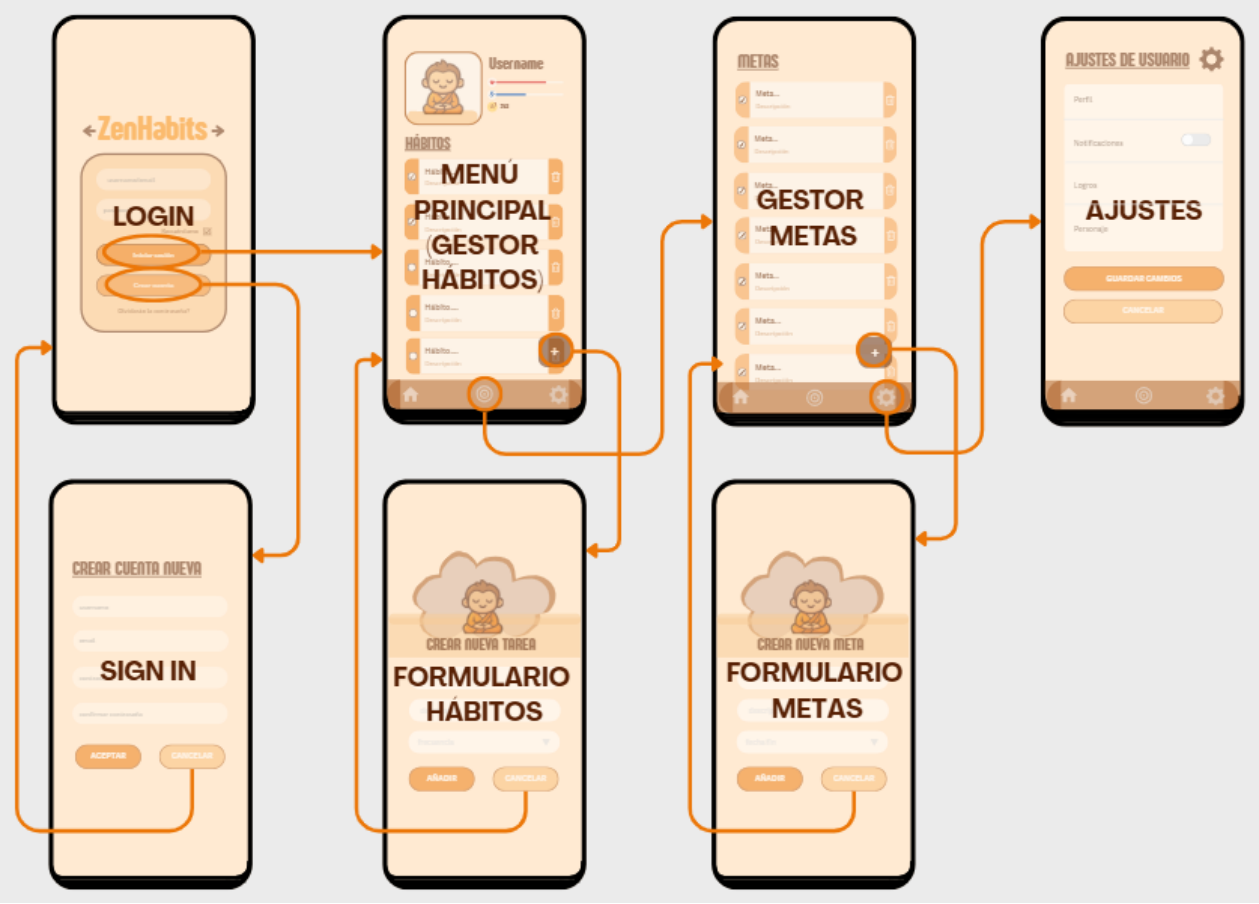

### 5.1.2. UX

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.

## 5.2. 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 también existe y estará adaptado otras disposiciones, la lógica de navegación se mantiene.



## 5.3. Características visuales

### 5.3.1. 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 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. |

### 5.3.2. 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.

### 5.3.3. 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.

## 5.4. 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.

## 5.5. Manual de usuario

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 *(Consultar Anexo I)*

# 6. Implementación

Para el desarrollo de ZENHABITS se han utilizado tecnologías actuales de desarrollo multiplataforma y 'backend', con almacenamiento local y remoto (dockerizado). Para poder realizar esto de forma correcta se necesita un entorno correcto y preparado para su desarrollo.

## 6.1. Entorno de desarrollo

Tanto para el desarrollo de la aplicación multiplataforma como el de la API, se ha utilizado Visual Studio Code.

### 6.1.1. 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:* Para emular teléfonos android
  - Versión: Android Studio Koala Feature Drop | 2024.1.2
  - Emulador: Pixel 6 API 35 |x86_64
- *Microsoft Visual Studio Community 2022 (64 bits):*
  - Versión: 17.11.5
- *Oracle VM VirtualBox:* Para virtualizar una máquina Linux
  - Versión: 7.0.2 r154219 (Qt5.15.2)
  - Máquina virtual: Debian (64-bits)
- *Docker Desktop:*
  - Versión: 28.0.4
- *Git:*
  - Versión: 2.34.1

### 6.1.2. Sistemas operativos

Los sistemas operativos disponibles, al ser multiplataforma, son: Android, iOS, Windows y Linux.

## 6.2. 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 |
| **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.
  - *Versión*: 3.29.3 (Flutter) y 3.7.2 (Dart).
- **SQLite**: Motor de base de datos ligero para guardar datos localmente sin conexión.
- **Rust + Axum**: Backend seguro y rápido, ideal para crear APIs REST
  - *Versión*: 1.86.0 (Rust).
- **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.

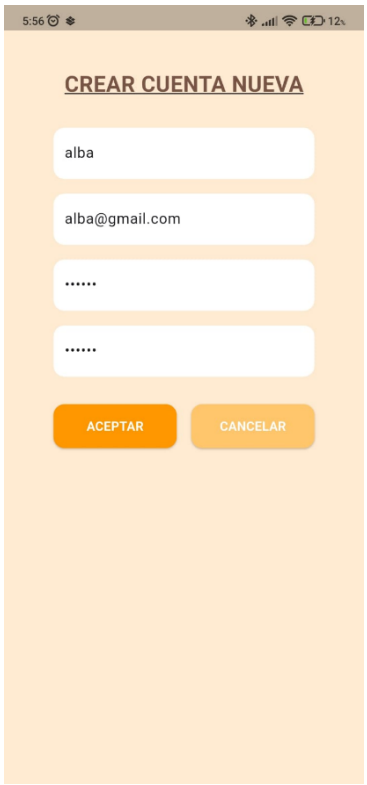
## 6.3. Funcionamiento

*ZENHABITS* es una aplicación *multiplataforma* diseñada para ayudarte a crear y mantener hábitos saludables con enfoque en el minimalismo digital y la atención plena, además de tus metas u objetivos. Todo esto se puede hacer de la siguiente manera.

- **Crear cuenta e iniciar sesión:**

  Para poder utilizar *ZENHABITS* es necesario tener una cuenta personal, para poder gestionar tus propios hábitos y metas, para ello se rellena un formulario con los campos nombre de usuario, correo y contraseña**.**

  Después podrás entrar a tu cuenta introduciendo tu nombre de usuario y contraseña.





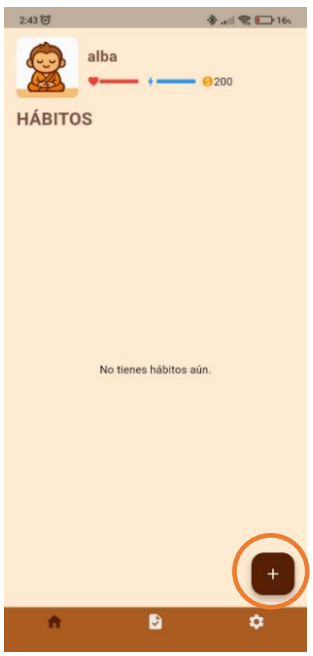
*(Capturas de ejecución móvil)*


*(Captura ejecución escritorio: Windows y linux)*

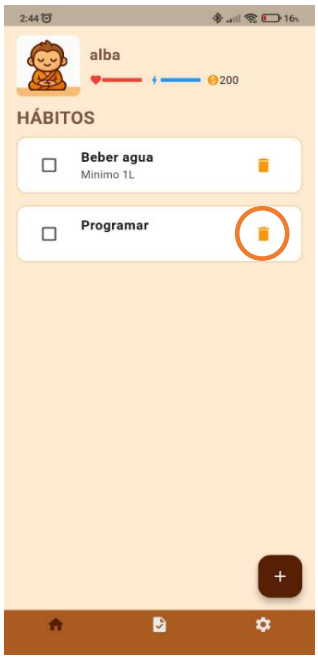
- **Menú principal y creación de hábitos:**

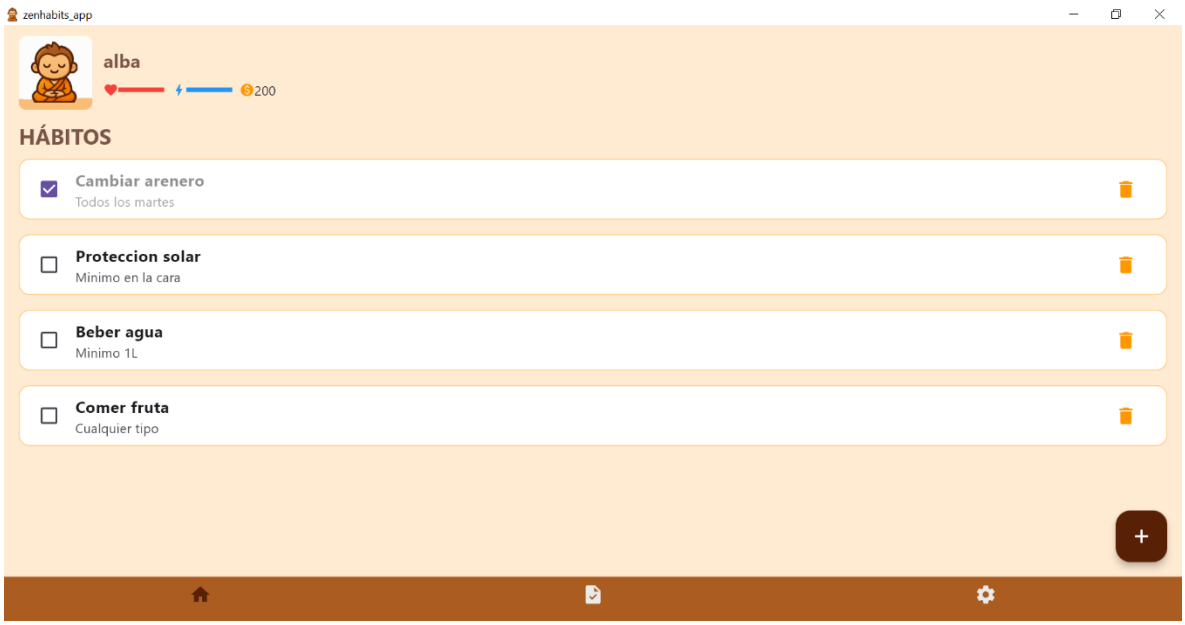
Tras iniciar sesión se mostrará el menú principal donde se observa el personaje personalizable junto a sus características y debajo nuestros hábitos (ninguno ya que es nueva la cuenta).

Se procede a crear un par de hábitos pulsando el botón flotante y aparecerán en nuestra pantalla principal. En la creación de hábitos la descripción puede quedar vacía si el usuario lo desea. *(Capturas ejecución móvil)*



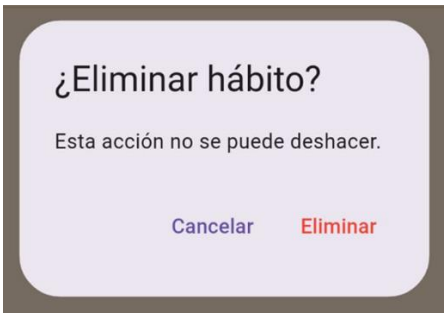


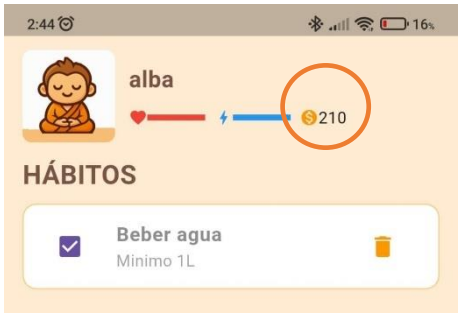





*(Captura ejecución escritorio: Windows y Linux)*

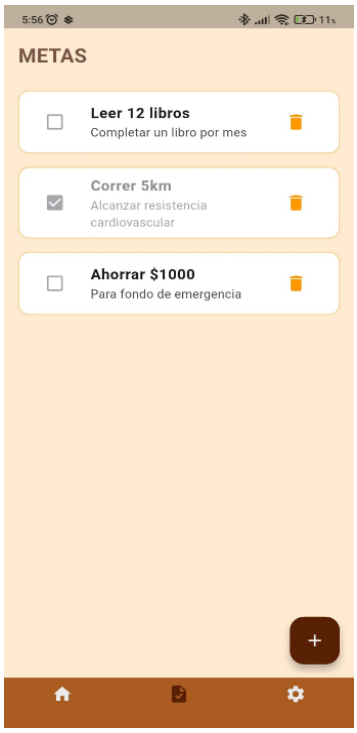
Además, podremos borrarlos pulsando un icono de la papelera en la propia etiqueta del hábito a la derecha, lo cual nos muestra una ventana emergente para que confirmemos la acción) o marcarlos como completados con el cuadro de verificación en la etiqueta a la izquierda, provocando que las monedas de nuestro personaje aumenten como recompensa (cada hábito completado serán 10 monedas).



- **Gestor de metas:**

Esta pantalla funciona exactamente igual que la pantalla principal con los hábitos, lo único que no aparece el personaje y sus detalles. Aquí podrás ir al formulario de creación de metas, eliminar metas y marcar como completadas estas; lo cual se verá reflejado en las monedas del personaje de la pantalla principal.




*Para una explicación extensa paso a paso, consultar manual de usuario en GitHub (Anexo II)*

### 6.3.1. Pruebas

Para validar el sistema de ZenHabits, se realizaron distintas pruebas para garantizar la calidad y escalabilidad de la aplicación.

#### 6.3.1.1. Pruebas de la API

Se utilizó Postman como herramienta principal para validar los "endpoints" expuestos por la API de ZENHABITS. Estas pruebas incluyeron operaciones de eliminar, actualizar e insertar y obtener datos sobre los usuarios y sus recursos principales como los hábitos. *(Capturas en el anexo III)*

- *Validación de 'endpoints':*
  - Comprobación de que la respuesta de todos los endpoints eran los códigos de estado HTTP esperados (200: OK, 201: Created, 400: Bad Request, 401: Unauthorized, 404: Not found, etc.).
  - Se comprobaron los métodos GET y POST en las rutas confirmando que se actualizaban los datos.
- *Manejo de errores y seguridad:*
  - Se enviaron peticiones con datos malformados o incompletos para validar la gestión de errores, comprobando que la API devolvía mensajes descriptivos y útiles.

#### 6.3.1.2. Pruebas de la Base de datos

- *Integridad de datos:*

- Se confirmó que las operaciones realizadas desde la API afectaban correctamente a la base de datos MySQL desplegada en el contenedor Docker. Además de verificar que las relaciones entre usuarios, hábitos y metas respetaron las claves foráneas y otras restricciones.

- *Pruebas de recuperación y escalabilidad:*

- Se simularon fallos de conexión con la base de datos y se comprobó que, al restaurar el servicio, la API reanudaba correctamente la sincronización de datos.

## 6.4. 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.

Para facilitar el despliegue del backend desarrollado en Rust, así como la base de datos MySQL, se utilizó Docker como herramienta principal de contenerización. Esto permitió garantizar un entorno homogéneo, controlado y fácilmente replicable, tanto para entornos de prueba como de producción. El sistema se compone de dos contenedores principales:

- *Backend (Rust):* Compilado y empaquetado en un contenedor ligero, el backend expone una API HTTP accesible desde clientes móviles y de escritorio.

- *Base de datos MySQL:* Alojada en un contenedor independiente, con volúmenes persistentes para conservar los datos en reinicios o actualizaciones.

Ambos contenedores se gestionan mediante *Docker Compose*, lo que facilita su orquestación y el mantenimiento de variables de entorno, volúmenes y redes privadas entre servicios.

1. Compilación de contenedores: `docker-compose build`
2. Ejecución en segundo plano: `docker-compose up -d`

3. Persistencia de datos**:** Volúmenes Docker fueron definidos para la base de datos, permitiendo que los datos sobrevivan a reinicios del contenedor.
4. Variables de entorno y secretos**:** Se configuraron mediante un archivo .env externo, aislando credenciales y configuraciones sensibles del código fuente.
5. Exposición del servicio**:** En la etapa actual, el 'backend' se encuentra expuesto únicamente en el *entorno local*, escuchando en un puerto específico. Esto permite la interacción directa con las aplicaciones cliente (móvil o escritorio) que se ejecuten en el mismo equipo o en una red local configurada adecuadamente.

   La configuración del puerto se define en el archivo docker-compose.yml, asegurando que el contenedor esté mapeado al puerto deseado en la máquina host:

```
ports:
  - "3000:3000"
```

   Aunque actualmente no se encuentra accesible desde redes externas, esta configuración permite validar completamente el funcionamiento del sistema de forma aislada y segura durante el proceso de pruebas y desarrollo.

   En una etapa futura, podrá desplegarse en un servidor accesible públicamente, aplicando medidas como firewall o autenticación para gestionar el tráfico externo y garantizar la seguridad del servicio.

## ***Aplicación móvil***

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

- Distribución directa:

  El archivo .apk (Android Package) y/o ejecutable 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 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.

## ***Aplicación escritorio***

Además de la versión móvil, se desarrolló una versión de escritorio utilizando Flutter, compatible tanto con sistemas operativos Windows como Linux. Esta variante está pensada para su uso en entornos donde se requiera una interfaz más amplia o donde no se disponga de un dispositivo móvil.

- Distribución directa:

Se generaron ejecutables específicos para cada plataforma mediante comandos de Flutter. Estos archivos pueden ser distribuidos manualmente a través de medios digitales como correo electrónico, enlaces de descarga (Google Drive, Dropbox, etc.) o incluso dispositivos físicos (USB).

- En *Windows*, el ejecutable (.exe) se encuentra dentro del directorio:

  `build/windows/runner/Release/`

- En *Linux*, el binario compilado se encuentra en:

  `build/linux/x64/release/bundle/`

Los usuarios pueden ejecutar la aplicación directamente sin necesidad de realizar instalaciones complejas, siempre que se cumplan los requisitos de dependencias del sistema.

Esta versión de escritorio permite ampliar el espectro de usuarios potenciales, proporcionando acceso desde estaciones de trabajo y equipos tradicionales, manteniendo una experiencia de usuario coherente con la aplicación móvil.

***Acceso y uso por parte del usuario***

Para utilizar la aplicación, el usuario debe instalarla en su dispositivo móvil (mediante el ejecutable, la APK o desde la tienda). Una vez instalada, la aplicación se conecta automáticamente con el 'backend' desplegado (configurado en la aplicación 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.

## 6.5. Líneas futuras

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

- **Escalabilidad y Orquestación**
  - *Ejecución del servidor:* Realizada mediante docker-compose en local, 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. A futuro podrá alojarse en un proveedor cloud para facilitar su escalabilidad, disponibilidad continua y mantenimiento remoto.
  - *Kubernetes:* Plataforma de orquestación de contenedores que será incorporada permitiendo escalado automático, balanceo de carga y alta disponibilidad.
- **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, permitiendo 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.

- **Validaciones y seguridad**

Se implementarán nuevas capas de validación utilizando JWT (JSON Web Tokens), permitiendo reforzar la seguridad en las operaciones sensibles del sistema, garantizando que únicamente los usuarios autenticados y autorizados puedan acceder a determinados recursos. Además, esta medida facilitará una mejor gestión de sesiones, proporcionando mayor control sobre el tiempo de expiración y la renovación de credenciales.

- **Soporte multilenguaje**

Con el fin de mejorar la accesibilidad y expandir el alcance de la aplicación, la interfaz podrá mostrarse en distintos idiomas según las preferencias del usuario, sin modificar la funcionalidad principal de la plataforma y ampliando el público.

- **Personalización visual general**

*Adaptabilidad la comodidad*: Se añadirán opciones como la posibilidad de cambiar el tamaño de la tipografía, así como la selección entre diferentes temas visuales (claro u oscuro). Estas mejoras están orientadas a ofrecer una experiencia más inclusiva.

# 7. Elementos destacables del desarrollo

## 7.1. Innovaciones

*Backend con Rust y MySQL dockerizado*

El backend fue desarrollado utilizando Rust, una decisión 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 u otras 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 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.

## 7.2. Problemas

### 7.2.1. 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 mí, 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.

### 7.2.2. 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, 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) e 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.

# 8. Conclusiones

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

El proceso general ha tenido una curva de aprendizaje elevada, cuyo proceso ha 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, incluso hubo un apagón nacional en una de las fases críticas del desarrollo; 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.

En cuanto a temas destacados en mi proyecto debido a que me resulta sencillo hacerlo, podría destacar mi habilidad con el diseño y poder hacer una interfaz agradable e intuitiva.

Considero que el resultado representa una valiosa oportunidad de crecimiento personal y profesional, ya que este proyecto me ha ayudado a afianzar conocimientos clave para mi carrera como programadora e incluso aprender otros muchos. Haber trabajado con tecnologías modernas, cuya adopción está en expansión, me ha permitido adquirir experiencia práctica que será muy valiosa para enfrentar desafíos en entornos profesionales reales. Además, me motiva especialmente ver el potencial que tiene para seguir evolucionando, y confío poder dedicar más tiempo al desarrollo de *ZENHABITS* en el futuro.

En resumen, este proyecto no solo ha demostrado el potencial evolutivo y viabilidad de la solución propuesta, sino que también me ha permitido desarrollar habilidades esenciales para mi crecimiento como desarrolladora. Ha sido una experiencia dura, corta e intensa pero entretenida, en la que confirmé que es posible construir soluciones

sólidas y actuales, incluso con recursos limitados, siempre que se trabaje con una base tecnológica bien pensada y una actitud perseverante frente a los obstáculos.

Para finalizar y cambiando de tema, quiero añadir que, a pesar de las dificultades y del ritmo más lento de lo previsto, este grado ha sido una experiencia que realmente he disfrutado. Por diversas circunstancias, he tenido que completarlo en tres años en lugar de dos, pero como se suele decir, *más vale tarde que nunca*. A lo largo del camino he aprendido mucho, no solo en lo técnico, sino también sobre mí misma, y me siento orgullosa de haber llegado hasta aquí además de haber conocido a gente que actualmente está en una parte importante e imprescindible de mi vida.

# 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.

Flutter Explained. (16 de 6 de 2021). *Floor Database in Flutter - Best Practices*. Obtenido de YouTube: https://www.youtube.com/watch?v=W-AwQpWM4f0

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

PlantUML. (8 de 6 de 2024 ). Obtenido de PlantUML: https://plantuml.com/es/

QuashBugs. (10 de 10 de 2023). *Exploring Flutter's Floor Library for Efficient Data Persistence*. Obtenido de QuashBugs: https://quashbugs.com/blog/exploring-flutters-floor-library-for-efficient-data-persistence

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

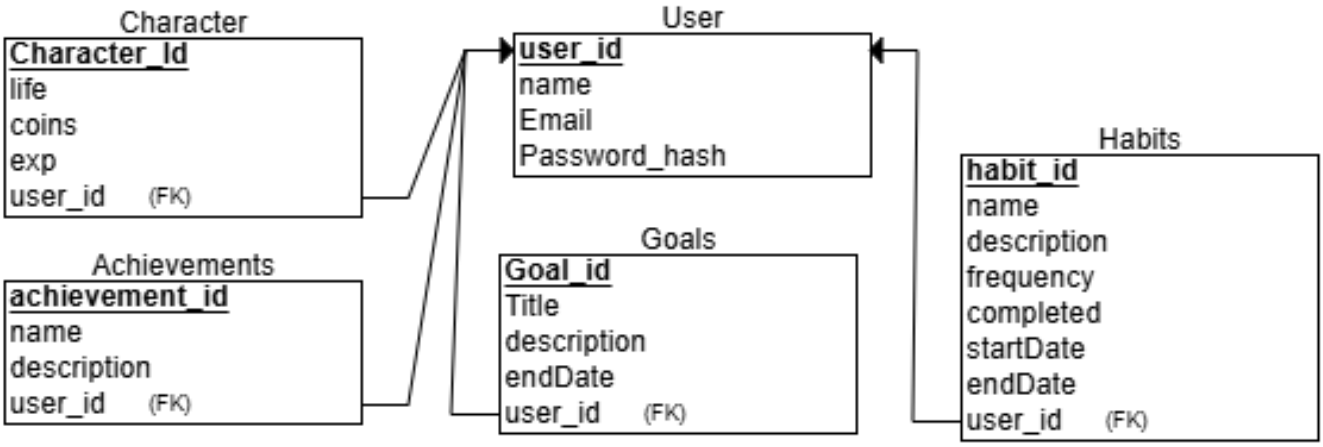
# Anexo I

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

En este repositorio se encuentra el código fuente de la aplicación junto con los ejecutables, memoria y el **manual de usuario**. Este último se encuentra en la carpeta docs junto al archivo de la memoria, se llama *ZenHabits_User Manual.p*

## Anexo II

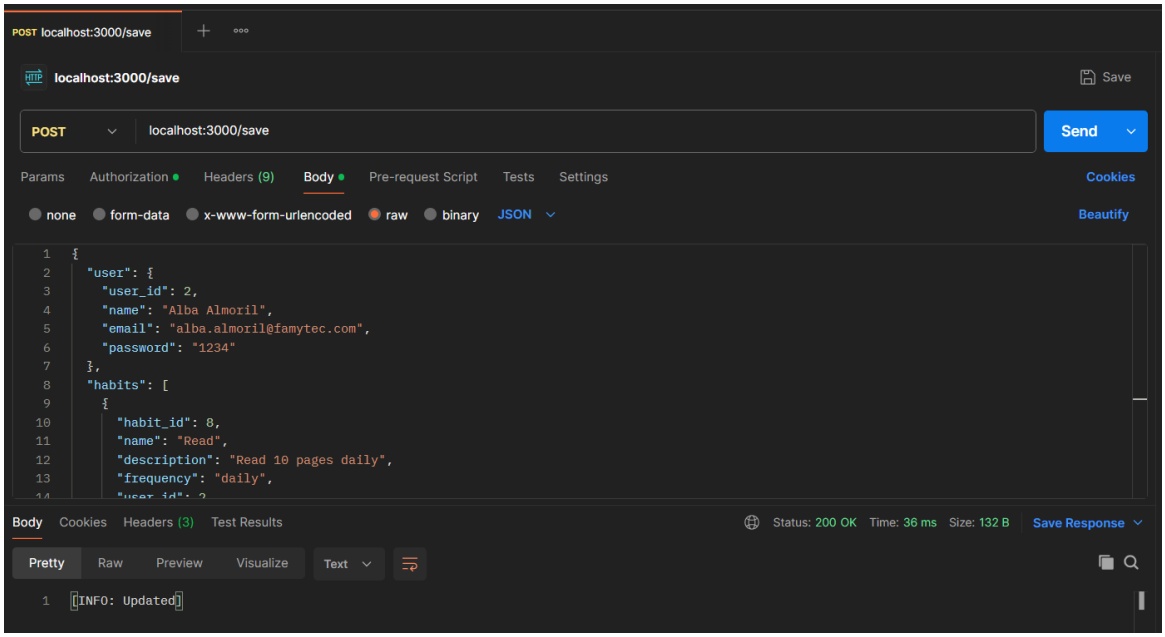
- *Esquema relacional de la base de datos:*



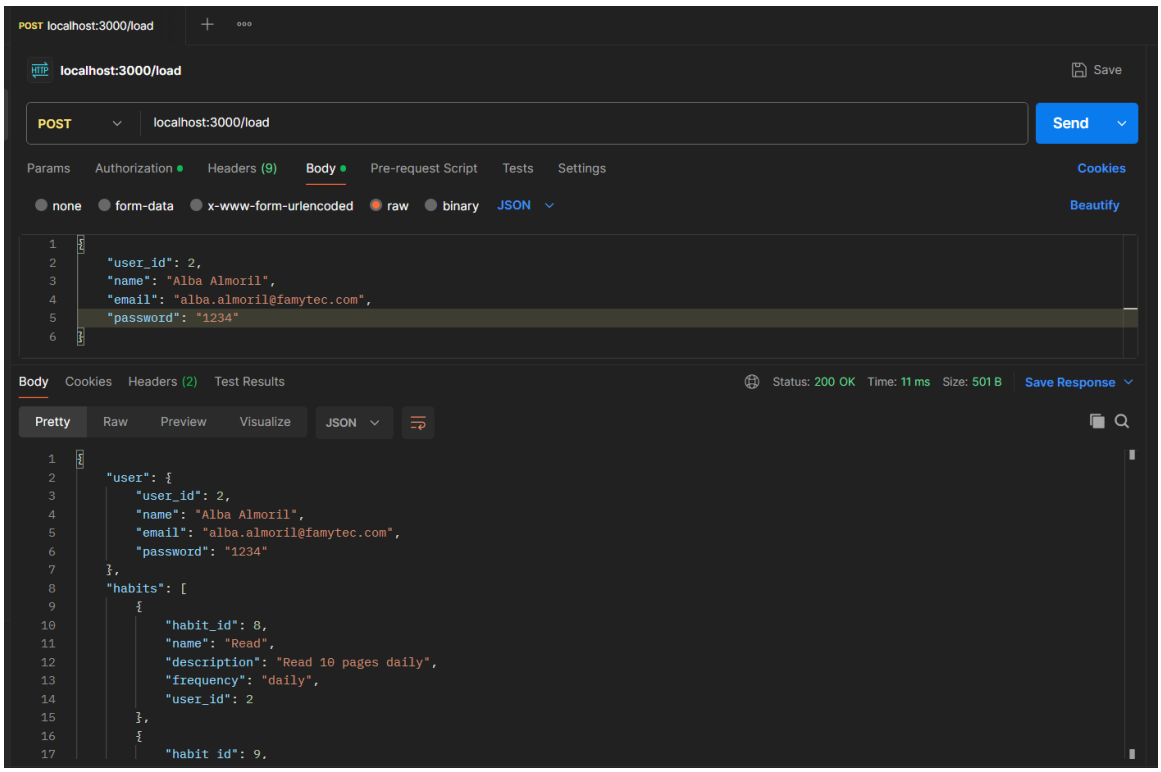
# Anexo III

Capturas de las pruebas realizadas en Postman para los endpoints de la API.

- **Endpoint save:**

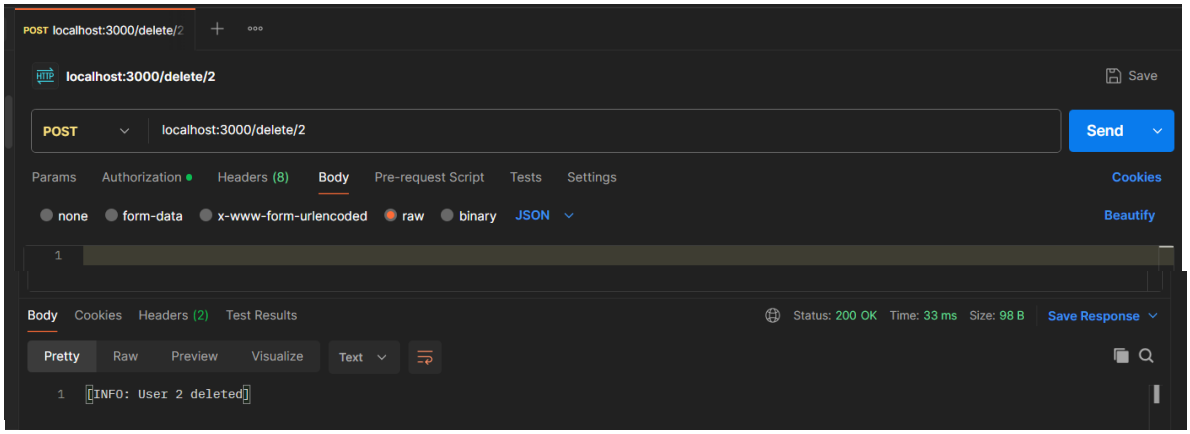



- **Endpoint load:**



- **Endpoint delete:**

Eliminamos al usuario 2 anteriormente creado, junto sus hábitos.




Intentamos eliminar un usuario 3, pero este no existe, por lo que no lo encuentra.