Memoria

Desarrollo de Aplicaciones Multiplataforma

GrowTogether

Una app de gestión de hábitos

Jordi Patuel Pons

IES María Enríquez · Gandía · 2025-2026

Índice

1. Introducción

1.1 Presentación y motivación

La idea viene de la falta personal de un control de hábitos, después de haber leído el libro de "Hábitos Atómicos" de James Clear y de un desafío con amigos de ver quién iba más al gimnasio. Anteriormente había probado hojas de cálculo personales para el control de estos hábitos, apps que piden suscripciones a mi parecer demasiado caras o el grupo de WhatsApp que tenía con amigos para controlar los días que íbamos al gimnasio. A pesar de que existen apps de registro de hábitos, me gustaba la idea de que los amigos pudieran ver el progreso de los demás y pudieran competir entre ellos.

1.2 Factor diferenciador

Aunque existen aplicaciones de seguimiento de hábitos, la mayoría se centran en el usuario individual. GrowTogether añade una capa social: los usuarios pueden crear desafíos, invitar a amigos y competir de forma sana. Esta mecánica de competición amistosa convierte una tarea que suele ser solitaria en una experiencia compartida, fomentando la constancia a través del compromiso social.

1.3 Análisis de la situación de partida

El mercado actual ofrece varias alternativas para el seguimiento de hábitos:

- **Habitica**: gamificación tipo RPG, pero orientada al usuario individual y con una curva de aprendizaje alta.
- **Streaks** (iOS): diseño minimalista, limitada a 12 hábitos y sin componente social.
- **HabitBull / Loop Habit Tracker**: gratuitas con estadísticas, pero sin funcionalidad social ni desafíos.
- **Hojas de cálculo / notas**: solución manual sin automatización, recordatorios ni motivación social.

Ninguna de estas opciones combina un sistema de hábitos completo con desafíos competitivos entre amigos de forma gratuita.

1.4 Objetivos del proyecto

Objetivo principal: Desarrollar una aplicación móvil multiplataforma que permita a los usuarios registrar, hacer seguimiento y mejorar sus hábitos diarios, incorporando un sistema de desafíos sociales entre amigos.

Objetivos específicos:

1. Implementar un sistema de registro y seguimiento de hábitos con estadísticas y rachas.
2. Desarrollar un módulo de desafíos con ranking y puntuación entre participantes.
3. Crear un sistema de amistad con solicitudes, búsqueda y gestión de contactos.
4. Diseñar un panel de administración para la gestión de usuarios, contenidos y métricas.
5. Incorporar consejos diarios con temática de sostenibilidad medioambiental.
6. Desplegar la aplicación en infraestructura cloud para acceso global.

1.5 Relación con los contenidos de los módulos

1. **Acceso a datos**: API REST con Spring Boot para la conexión de la base de datos PostgreSQL con la aplicación.
2. **Desarrollo de interfaces**: Dos interfaces, una para el usuario (móvil) y otra para el administrador (web).
3. **Programación multimedia y dispositivos móviles**: App móvil para Android con Flutter.
4. **Digitalización**: Despliegue en servicios cloud en AWS para la digitalización de la aplicación.
5. **Sistemas de gestión empresarial**: Panel de administración con gestión de usuarios, recursos, contenidos y métricas de uso.
6. **Programación de servicios y procesos**: Gestión de tareas en segundo plano como recordatorios, notificaciones y sincronización.
7. **Sostenibilidad**: Consejos e ideas de hábitos de reciclaje, ahorro energético y movilidad sostenible.

2. Presentación de las tecnologías utilizadas

2.1 Justificación de la elección de las tecnologías

Para cumplir con los requisitos de multiplataforma, escalabilidad y gestión propia de datos, se ha seleccionado el siguiente stack tecnológico. Se han valorado alternativas en cada capa y se justifica la elección final:

- **Frontend (App & Web): Flutter (Dart)** Permite desarrollar la aplicación móvil (Android) y el panel de administración (Web) con un único código base. Se valoró React Native como alternativa, pero Flutter ofrece mejor rendimiento y un sistema de widgets más completo para interfaces personalizadas.
- **Backend (API): Spring Boot 3.3 (Java 17)** Framework robusto para construir APIs REST. Ofrece un ecosistema maduro con Spring Security (JWT), Spring Data JPA y validación integrada. Se valoró Node.js (Express), pero Spring Boot proporciona mayor estructura y tipado estático, más adecuado para una API con lógica de negocio compleja.
- **Base de Datos: PostgreSQL** Base de datos relacional robusta. Esencial para manejar la integridad de los datos sociales (Usuarios, Amigos, Desafíos, Participaciones) que una base NoSQL gestionaría con mayor complejidad.
- **Infraestructura (Cloud): AWS** Se utilizarán los servicios de Amazon Web Services para el despliegue: EC2 para el servidor de la API y RDS para la base de datos gestionada PostgreSQL. AWS ofrece un ecosistema completo, amplia documentación y disponibilidad de créditos educativos.
- **Docker** Permitirá empaquetar la API y sus dependencias en contenedores, asegurando que el entorno de desarrollo sea idéntico al de producción y facilitando el despliegue en cualquier proveedor Cloud.
- **Diseño y Prototipado: Figma** Herramienta estándar para el diseño de interfaces (UI) y experiencia de usuario (UX) antes de la implementación, permitiendo validar los flujos de navegación.
- **Control de Versiones: Git + GitHub** Para la gestión del código fuente, control de cambios y colaboración.

3. Análisis del proyecto

3.1. Requisitos funcionales y no funcionales

Requisitos funcionales

- **RF 1: Usuario**
  - **RF 1.1**: El usuario se podrá registrar con email y contraseña.
  - **RF 1.2**: El usuario podrá editar la información de su perfil.
  - **RF 1.3**: El usuario podrá buscar amigos, enviar solicitudes, aceptar o denegar estas solicitudes y eliminar amigos.
- **RF 2: Hábito**
  - **RF 2.1**: El usuario podrá crear un hábito definiendo nombre, descripción y frecuencia.
  - **RF 2.2**: El usuario podrá marcar un hábito como completado o desmarcarlo.
  - **RF 2.3**: El usuario podrá editar la información de un hábito existente.
  - **RF 2.4**: El usuario podrá ver el progreso y el historial de sus hábitos.
  - **RF 2.5**: El sistema enviará notificaciones con frecuencia personalizada, permitiendo acciones rápidas desde la notificación.
- **RF 3: Desafío**
  - **RF 3.1**: El usuario podrá crear un desafío estableciendo un objetivo y una fecha límite.
  - **RF 3.2**: El usuario podrá invitar a uno o más usuarios a un desafío.
  - **RF 3.3**: El sistema gestionará una puntuación basada en rachas (hábitos cumplidos consecutivamente).
  - **RF 3.4**: Los usuarios participantes podrán ver el progreso del desafío de sus amigos.
- **RF 4: Administrador**
  - **RF 4.1**: El administrador tendrá capacidad para eliminar usuarios, desafíos y hábitos.
  - **RF 4.2**: El administrador podrá añadir recursos globales como consejos e ideas de hábitos.
  - **RF 4.3**: El administrador tendrá acceso a visualizar métricas de uso de hábitos y desafíos.

Requisitos no funcionales

- **RNF 1: Rendimiento** — La API deberá responder a las peticiones en menos de 500ms en condiciones normales de uso.
- **RNF 2: Seguridad** — Las contraseñas se almacenarán cifradas con BCrypt. La autenticación se realizará mediante tokens JWT. Las comunicaciones se realizarán sobre HTTPS.
- **RNF 3: Disponibilidad** — La aplicación estará desplegada en AWS con una disponibilidad objetivo del 99 %.
- **RNF 4: Usabilidad** — La interfaz seguirá las guías de Material Design, con soporte para múltiples temas y multiidioma (castellano, inglés, valenciano).
- **RNF 5: Compatibilidad** — La app móvil será compatible con Android 5.0 (API 21) o superior.
- **RNF 6: Mantenibilidad** — El código seguirá una arquitectura por capas (Controller-Service-Repository) para facilitar el mantenimiento y la extensión.

Análisis de costes y viabilidad

Concepto	Coste
Desarrollo (mano de obra)	0 € (proyecto académico)
AWS (EC2 + RDS)	0 € (créditos educativos)
Herramientas (Flutter, Spring Boot, PostgreSQL)	0 € (open source)
Figma	0 € (plan gratuito)
GitHub	0 € (plan gratuito)
Total estimado	**0 €**

El proyecto es viable técnicamente ya que utiliza exclusivamente tecnologías open source y gratuitas. El despliegue en AWS se cubre con créditos educativos. En un escenario de producción real, los costes principales serían el servidor cloud (~15-30 €/mes) y un dominio (~10-15 €/año).

3.2. Temporalización del proyecto

- Diagrama de Gantt

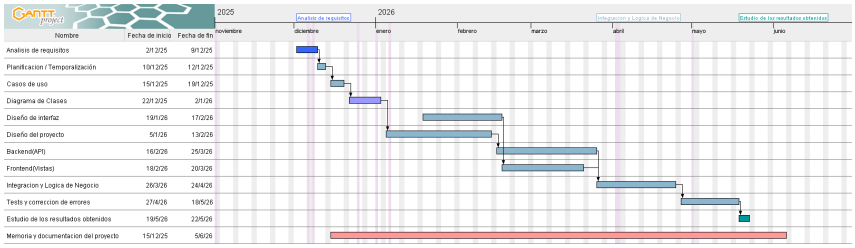


Figura 1: Diagrama de Gantt

Hitos del proyecto

Hito	Tarea	Fecha inicio	Fecha fin
1	Análisis de requisitos	02/12/2025	09/12/2025
2	Planificación / Temporalización	10/12/2025	12/12/2025
3	Casos de uso	15/12/2025	19/12/2025
4	Diagrama de clases	22/12/2025	02/01/2026
5	Diseño de interfaz	19/01/2026	17/02/2026
6	Diseño del proyecto	05/01/2026	13/02/2026
7	Backend (API)	16/02/2026	25/03/2026
8	Frontend (Vistas)	18/02/2026	20/03/2026
9	Integración y lógica de negocio	26/03/2026	24/04/2026
10	Tests y corrección de errores	27/04/2026	18/05/2026
11	Estudio de los resultados obtenidos	19/05/2026	22/05/2026
12	Memoria y documentación del proyecto	15/12/2025	05/06/2026

3.3. Casos de uso

- Diagrama de casos de uso
    - Herencia:

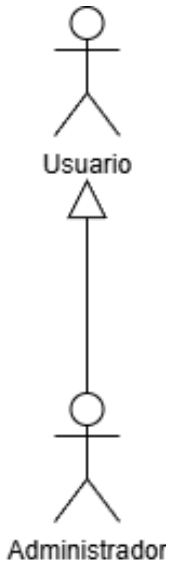


Figura 2: Diagrama de casos de uso - Herencia

    - Panel de Administrador:

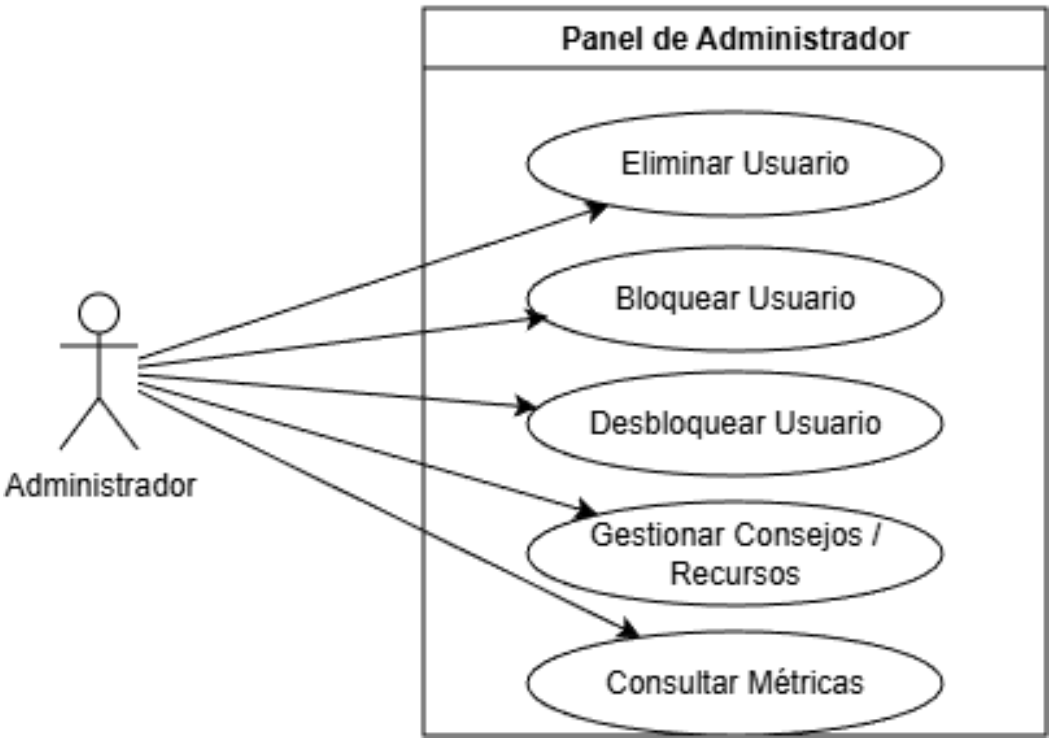


Figura 3: Diagrama de casos de uso - Panel de Administrador

- Panel de Usuario:

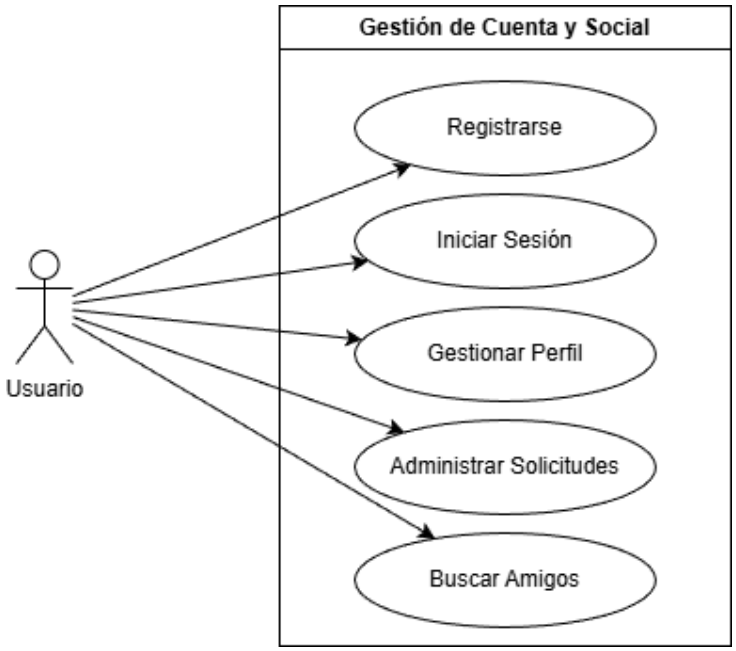


Figura 4: Diagrama de casos de uso - Panel de Usuario

- Panel de Gestión de Hábitos:

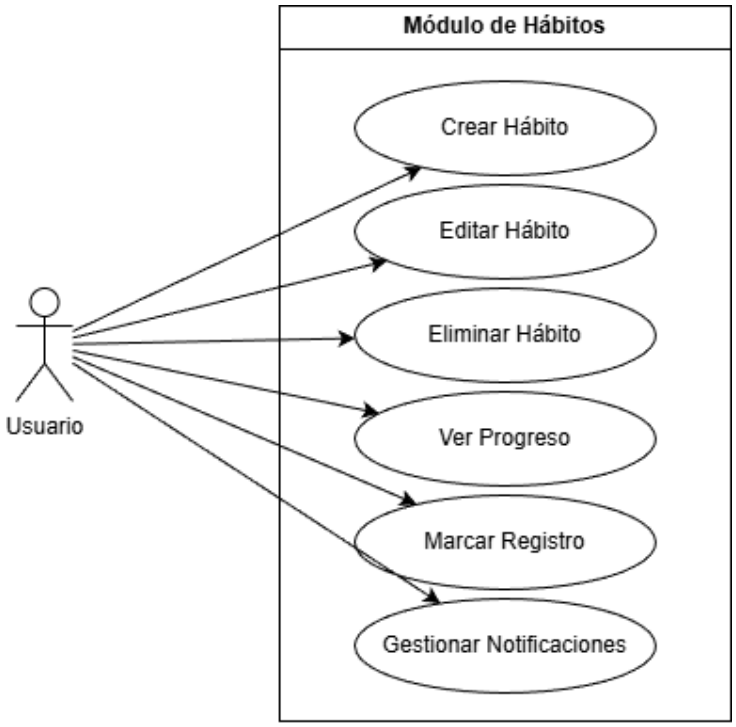


Figura 5: Diagrama de casos de uso - Panel de Hábitos

- Panel de Gestión de Desafíos:

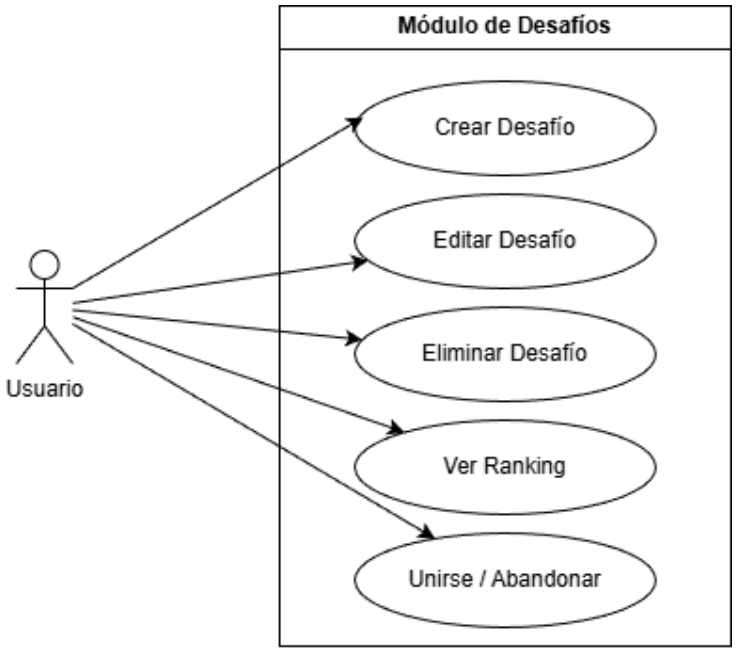


Figura 6: Diagrama de casos de uso - Panel de Desafíos

Casos de uso CU-01 a CU-04 - Gestión de Usuario

ID	Nombre	Actores	Precondiciones	Curso normal	Postcondiciones	Alternativas
CU-01	Registrarse	Usuario no registrado	No tiene cuenta activa.	1. Accede a registro. 2. Introduce nombre, email y password. 3. El sistema valida datos. 4. Crea perfil y redirige al login.	Usuario registrado en BD.	3a. Email ya existe: error. 3b. Password no valida: error.
CU-02	Iniciar sesion	Usuario registrado	Tiene cuenta registrada.	1. Introduce email y password. 2. El sistema valida credenciales. 3. Genera token JWT. 4. Redirige al dashboard.	Sesion activa con JWT.	2a. Credenciales incorrectas: error.
CU-03	Gestionar perfil	Usuario autenticado	Ha iniciado sesion.	1. Accede a perfil. 2. Modifica datos. 3. El sistema valida. 4. Guarda cambios.	Perfil actualizado.	3a. Nombre/email duplicado: error. 3b. Cambio password: requiere actual.
CU-04	Administra solicitudes	Usuario autenticado	Tiene solicitudes pendientes.	1. Accede a solicitudes. 2. Acepta o rechaza. 3. Actualiza lista de amigos.	Solicitud procesada.	Ninguna.

Casos de uso CU-05 a CU-08 - Amigos y Gestión de Hábitos

ID	Nombre	Actores	Precondiciones	Curso normal	Postcondiciones	Alternativas
CU-05	Buscar amigos	Usuario autenticado	Ha iniciado sesion.	1. Introduce nombre en buscador. 2. El sistema filtra usuarios. 3. Selecciona y envia solicitud. 4. Registra solicitud.	Solicitud enviada.	2a. Sin resultados: mensaje. 3a. Ya son amigos: aviso.
CU-06	Crear habito	Usuario autenticado	Ha iniciado sesion.	1. Pulsa crear habito. 2. Introduce nombre, descripcion, frecuencia. 3. Selecciona icono y tipo. 4. El sistema valida y guarda.	Habito creado y visible.	4a. Datos no validos: error.
CU-07	Editar habito	Usuario autenticado	Tiene al menos un habito.	1. Selecciona habito. 2. Accede a edicion. 3. Modifica campos. 4. El sistema valida y guarda.	Habito actualizado.	4a. Datos no validos: error.
CU-08	Eliminar habito	Usuario autenticado	Tiene al menos un habito.	1. Selecciona habito. 2. Pulsa eliminar y confirma. 3. Desactiva habito e historial.	Habito inactivo (soft delete).	2a. Cancela: sin cambios.

Casos de uso CU-09 a CU-12 - Hábitos y Desafíos

ID	Nombre	Actores	Precondiciones	Curso normal	Postcondiciones	Alternativas
CU-09	Ver progreso	Usuario autenticado	Ha iniciado sesion.	1. Accede a estadisticas. 2. Calcula rachas y porcentajes. 3. Muestra graficas.	Ninguna.	2a. Sin habitos: mensaje informativo.
CU-10	Marcar registro	Usuario autenticado	Tiene al menos un habito activo.	1. Selecciona dia en carrusel. 2. Pulsa check del habito. 3. Registra cumplimiento y actualiza racha.	Registro guardado, racha actualizada.	2a. Ya marcado: desmarca y recalcula.
CU-11	Gestionar notificaciones	Usuario autenticado	Tiene al menos un habito.	1. Accede a ajustes. 2. Selecciona hora y dias. 3. Programa notificaciones. 4. Confirma configuracion.	Notificaciones programadas.	2a. Permisos denegados: aviso.
CU-12	Crear desafio	Usuario autenticado	Ha iniciado sesion.	1. Accede a desafios. 2. Pulsa crear nuevo. 3. Define nombre, objetivo, fecha fin. 4. El sistema valida y crea.	Desafio creado y visible.	4a. Datos no validos o fecha pasada: error.

Casos de uso CU-13 a CU-16 - Gestión de Desafíos

ID	Nombre	Actores	Precondiciones	Curso normal	Postcondiciones	Alternativas
CU-13	Editar desafio	Creador del desafio	Es el creador.	1. Selecciona su desafio. 2. Modifica campos. 3. El sistema valida y guarda.	Desafio actualizado.	3a. Datos no validos: error.
CU-14	Eliminar desafio	Creador del desafio	Es el creador.	1. Selecciona su desafio. 2. Pulsa eliminar. 3. Pide confirmacion. 4. Confirma. 5. Desactiva desafio y participaciones.	Desafio inactivo (soft delete).	4a. Cancela: sin cambios.
CU-15	Ver ranking	Usuario autentica-do	Inscrito en el desafio.	1. Entra en desafio. 2. Ordena participantes por puntuacion. 3. Muestra ranking.	Ninguna.	2a. Un solo participante: ranking individual.
CU-16	Unirse / Abando-nar	Usuario autentica-do	Desafio activo.	1. Selecciona desafio. 2. Pulsa unirse o abandonar. 3. Registra inscripcion o baja.	Usuario inscrito o eliminado.	2a. Desafio finalizado: no permite unirse.

Casos de uso CU-17 a CU-21 - Panel de Administración

ID	Nombre	Actores	Precondiciones	Curso normal	Postcondiciones	Alternativas
CU-17	Eliminar usuario	Admin	Logueado.	1. Busca y selecciona usuario. 2. Confirma la eliminacion. 3. Desactiva cuenta.	Cuenta desactivada (soft delete).	2a. Cancela: sin cambios.
CU-18	Bloquear usuario	Admin	Usuario activo.	1. Busca usuario. 2. Selecciona bloquear y confirma. 3. Bloquea acceso.	Acceso bloqueado.	Ninguna.
CU-19	Desbloquear usuario	Admin	Usuario bloqueado.	1. Busca usuario. 2. Selecciona desbloquear y confirma. 3. Restaura acceso.	Acceso restaurado.	Ninguna.
CU-20	Gestionar consejos	Admin	Logueado.	1. Accede a contenidos. 2. Crea o edita consejo. 3. Guarda cambios.	Contenido publicado.	2a. Contenido vacio: error.
CU-21	Consultar metricas	Admin	Logueado.	1. Accede a metricas. 2. Procesa datos de uso. 3. Muestra estadisticas.	Ninguna.	2a. Sin datos: mensaje.

3.4. Diagrama de clases inicial

- **Diagrama de clases**

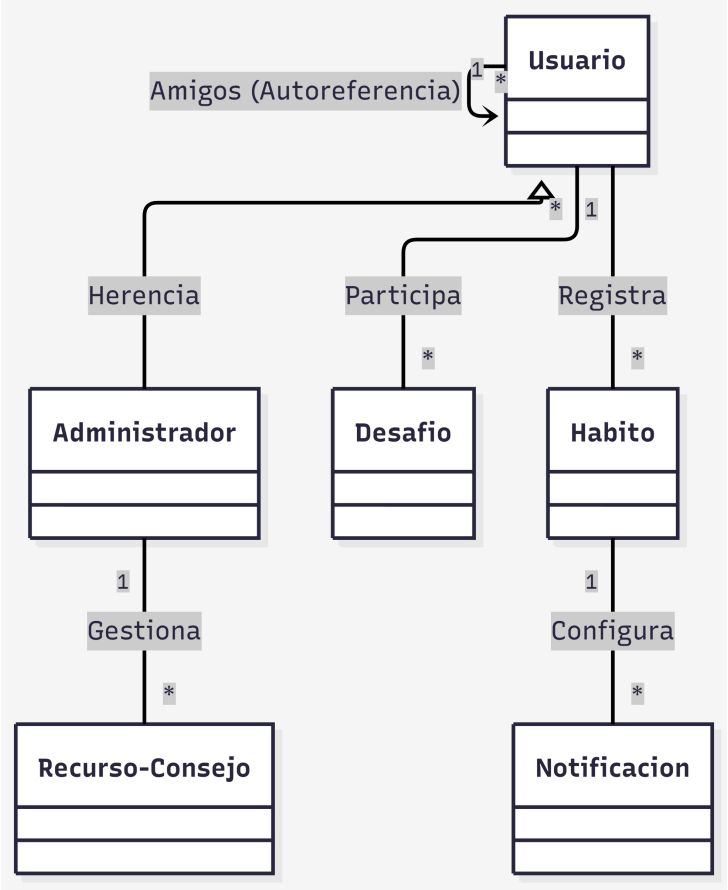


Figura 7: Diagrama de clases

- **Descripción de las clases**
  - **Usuario**: Perfil de la persona que usa la app. Guarda sus datos personales, su lista de amigos y su lista de hábitos. Tiene una relación de autoreferencia para gestionar amistades.
  - **Administrador**: Hereda de Usuario. Tiene permisos para eliminar usuarios, gestionar recursos y consultar métricas del sistema. *(En la implementación final se simplificó como atributo `rol` del usuario en lugar de subclase, véase el apartado 4.2.)*
  - **Hábito**: Actividad que el usuario quiere convertir en rutina (por ejemplo: “Beber agua” o “Ir al gimnasio”). Un usuario puede tener múltiples hábitos.
  - **Desafío**: Retos grupales con objetivo y fecha límite para motivar y competir entre usuarios. Un usuario puede participar en múltiples desafíos.
  - **Notificación**: Avisos configurables asociados a un hábito para recordar al usuario

su cumplimiento.

- **Recurso/Consejo**: Contenido publicado por los administradores (consejos, ideas de hábitos) disponible para todos los usuarios.

- **Diagrama entidad-relación**

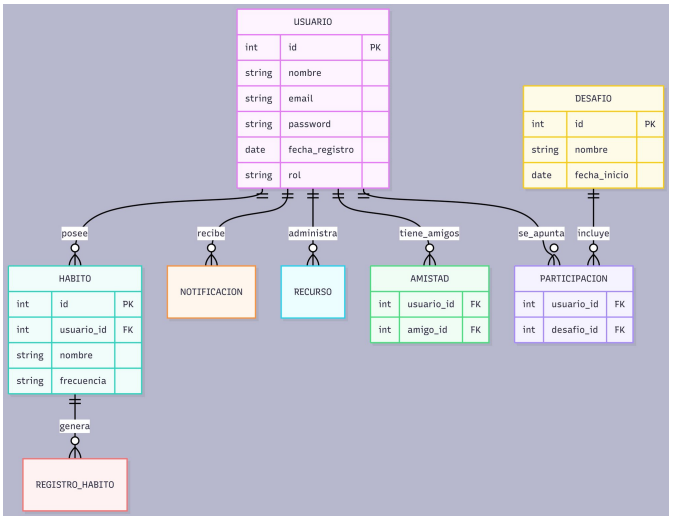


Figura 8: Diagrama entidad-relación

3.5. Wireframes de interfaces

- Bocetos a mano.

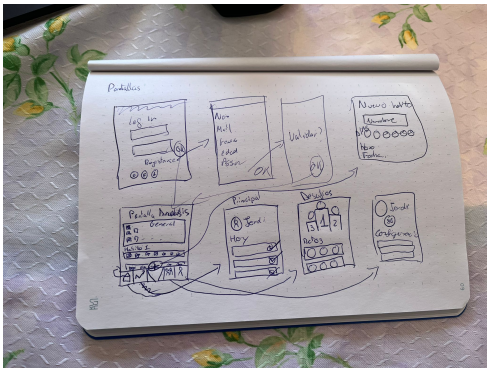


Figura 9: Boceto a mano

- Wireframes:
  - App:

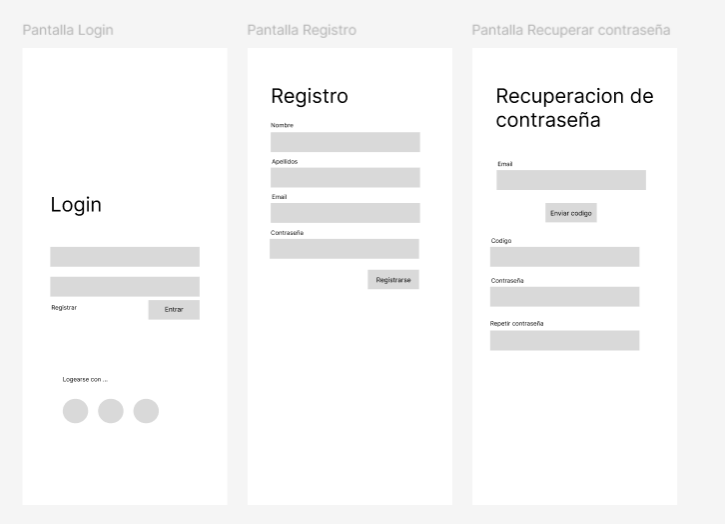


Figura 10: Pantalla de Login

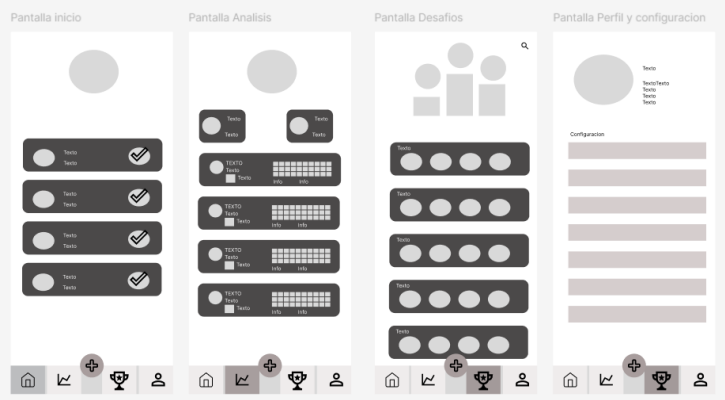


Figura 11: Pantallas Principales

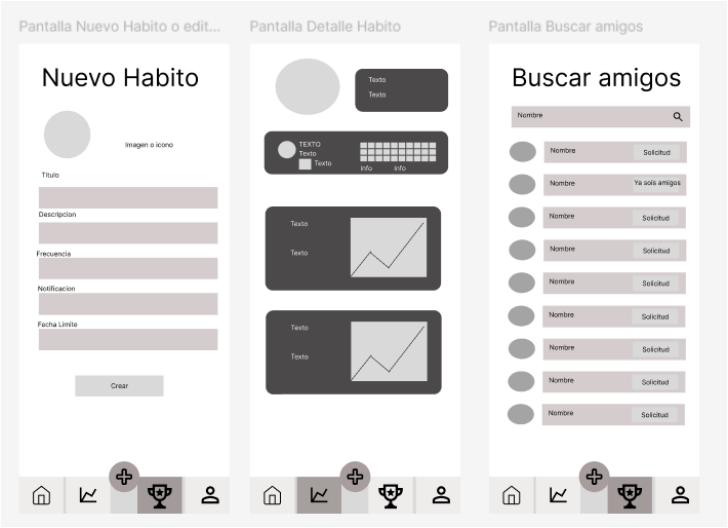


Figura 12: Pantallas Secundarias

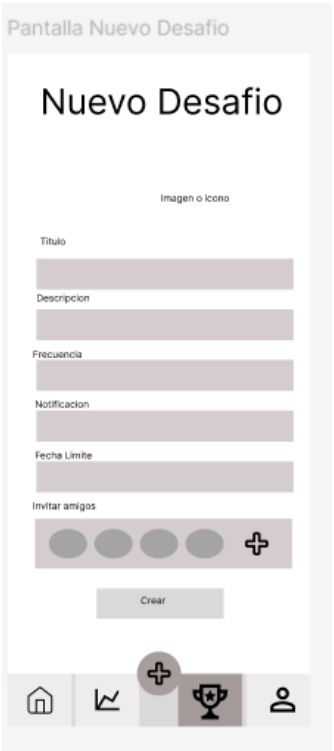


Figura 13: Pantalla de Desafío

4. Diseño del proyecto

4.1. Arquitectura del sistema

La arquitectura del sistema sigue un modelo **cliente-servidor de tres capas**:

- **Capa de cliente**: Dos aplicaciones desarrolladas con Flutter. La app móvil (Android) para los usuarios finales y una aplicación web para el panel de administración. Ambas se comunican con el backend a través de peticiones HTTP a la API REST.
- **Capa de backend**: API REST desarrollada con Spring Boot 3.3 (Java 17). Gestiona la lógica de negocio, la autenticación mediante JWT (Spring Security) y el acceso a datos mediante Spring Data JPA.
- **Capa de datos**: Base de datos PostgreSQL que almacena toda la información del sistema (usuarios, hábitos, desafíos, registros, recursos).

El despliegue se realiza en AWS, con el servidor backend en una instancia EC2 y la base de datos en el servicio gestionado RDS.

- **Diagrama de la arquitectura**

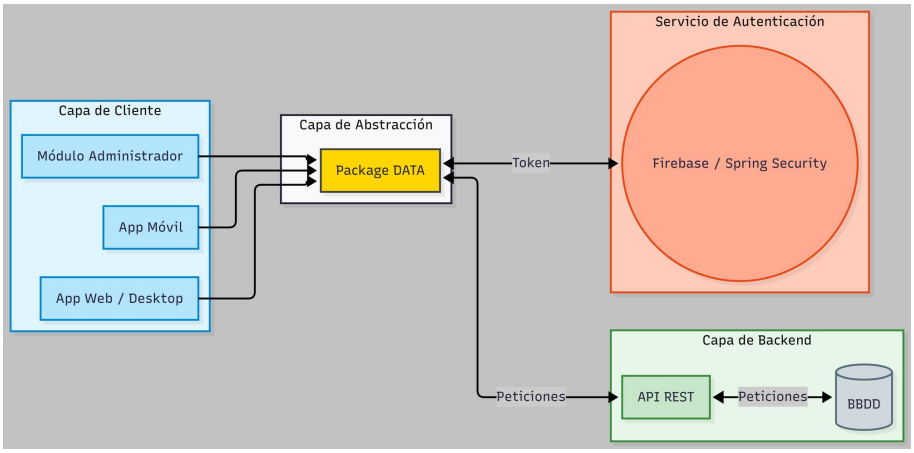


Figura 14: Diagrama de arquitectura

- **Diagrama de despliegue**

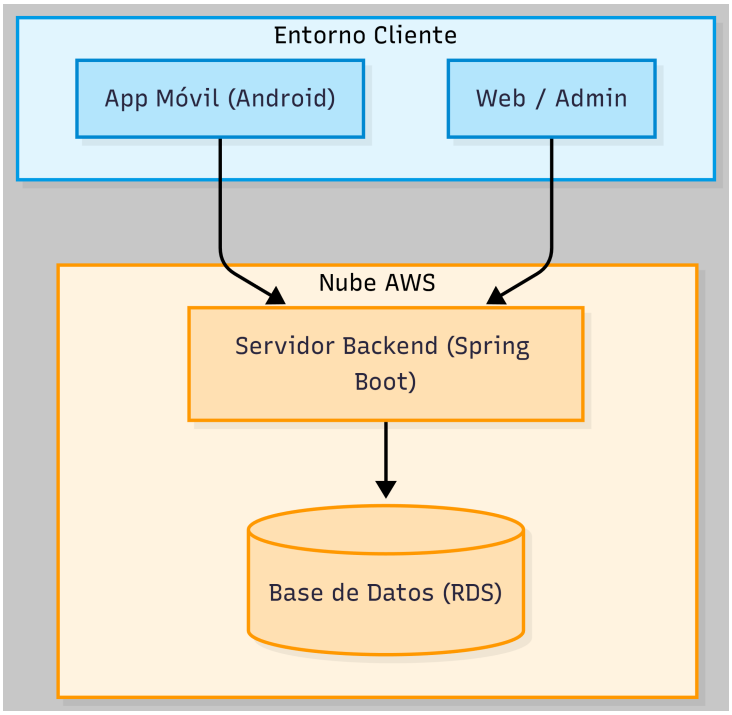


Figura 15: Diagrama de despliegue

El diagrama muestra los dos entornos: el entorno cliente (app móvil Android y web de administración) que se conecta a la nube AWS donde se aloja el servidor Spring Boot y la base de datos RDS.

4.2. Diagrama de clases definitivo

- Diagrama de clases

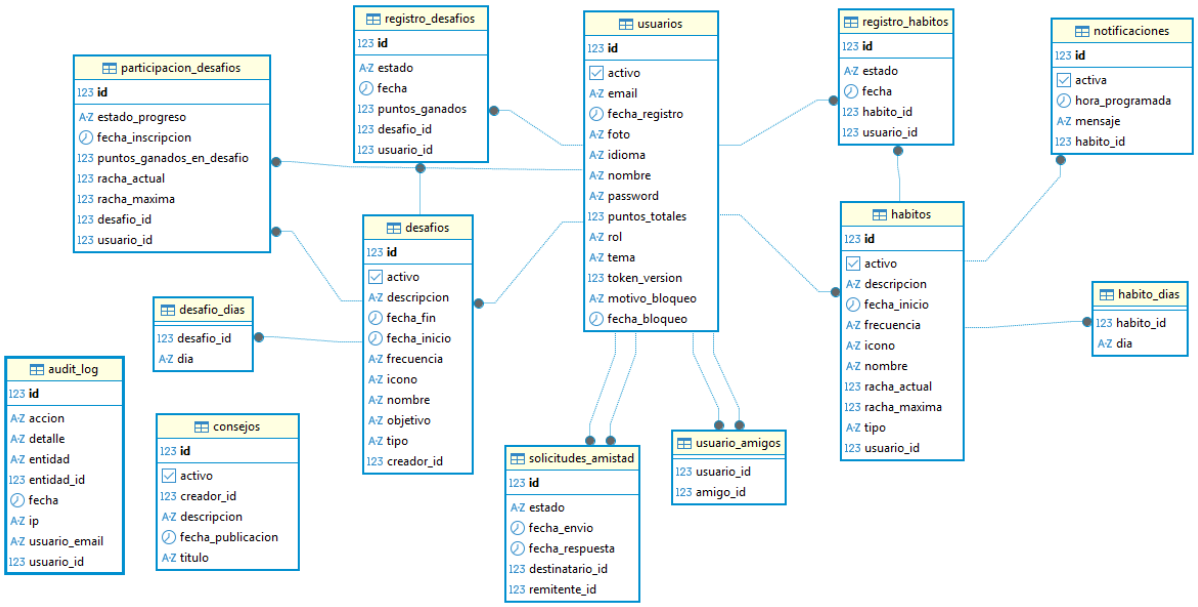


Figura 16: Diagrama de clases

- Descripción de las clases

Clase: Usuario (usuarios)

- **Descripción:** Representa a los usuarios finales y administradores de la plataforma.
- **Atributos:** `id`, `activo`, `email`, `fecha_registro`, `foto`, `idioma`, `nombre`, `password`, `puntos_totales`, `rol`, `tema`, `token_version`, `motivo_bloqueo`, `fecha_bloqueo`.
- **Relaciones:** Se relaciona con sí mismo a través de `usuario_amigos` y `solicitudes_amistad` . Se relaciona con `habitos`, `registro_habitos`, `desafios`, `participacion_desafios` , `registro_desafios`, `consejos` y `audit_log`.

Clase: Hábito (habitos)

- **Descripción:** Define la meta o rutina que el usuario desea seguir.
- **Atributos:** `id`, `activo`, `descripcion`, `fecha_inicio`, `frecuencia`, `icono`, `nombre`, `racha_actual`, `racha_maxima`, `tipo`.
- **Relaciones:** Pertenece a un `usuario_id`. Se relaciona con `registro_habitos`, `notificaciones` y `habito_dias`.

Clase: Registro de Hábito (registro_habitos)

- **Descripción:** Entidad que documenta el cumplimiento diario de un hábito.
- **Atributos:** `id`, `estado`, `fecha`.
- **Relaciones:** Asociado a un `habito_id` y a un `usuario_id`.

Clase: Días de Hábito (habito_dias)

- **Descripción:** Tabla auxiliar que almacena los días específicos de la semana asignados a un hábito.
- **Atributos:** `dia`.
- **Relaciones:** Asociado a un `habito_id`.

Clase: Notificación (notificaciones)

- **Descripción:** Sistema de alertas programadas para recordar hábitos.
- **Atributos:** `id`, `activa`, `hora_programada`, `mensaje`.
- **Relaciones:** Asociada a un `habito_id`.

Clase: Desafío (desafios)

- **Descripción:** Retos grupales con un objetivo y fechas límite.
- **Atributos:** `id`, `activo`, `descripcion`, `fecha_fin`, `fecha_inicio`, `frecuencia`, `icono`, `nombre`, `objetivo`, `tipo`.
- **Relaciones:** Creado por un `creador_id` (Usuario). Se relaciona con `participacion_desafios`, `registro_desafios` y `desafio_dias`.

Clase: Participación en Desafío (participacion_desafios)

- **Descripción:** Registro de la inscripción y progreso de un usuario en un desafío.
- **Atributos:** `id`, `estado_progreso`, `fecha_inscripcion`, `puntos_ganados_en_desafio`, `racha_actual`, `racha_maxima`.
- **Relaciones:** Asociada a un `desafio_id` y a un `usuario_id`.

Clase: Registro de Desafío (registro_desafios)

- **Descripción:** Documenta los avances o cumplimientos diarios dentro de un desafío.
- **Atributos:** `id`, `estado`, `fecha`, `puntos_ganados`.
- **Relaciones:** Asociado a un `desafio_id` y a un `usuario_id`.

Clase: Días de Desafío (desafio_dias)

- **Descripción:** Tabla auxiliar para los días específicos de la semana asignados a un desafío.

- **Atributos:** `dia`.
- **Relaciones:** Asociado a un `desafio_id`.

Clase: Consejo (consejos)

- **Descripción:** Material de apoyo y tips gestionados por los administradores.
- **Atributos:** `id`, `activo`, `descripcion`, `fecha_publicacion`, `titulo`.
- **Relaciones:** Creado por un `creador_id` (Usuario administrador).

Clase: Solicitud de Amistad (solicitudes_amistad)

- **Descripción:** Gestiona las peticiones de amistad entre usuarios.
- **Atributos:** `id`, `estado`, `fecha_envio`, `fecha_respuesta`.
- **Relaciones:** Vincula a un `remitente_id` y un `destinatario_id` (ambos Usuarios).

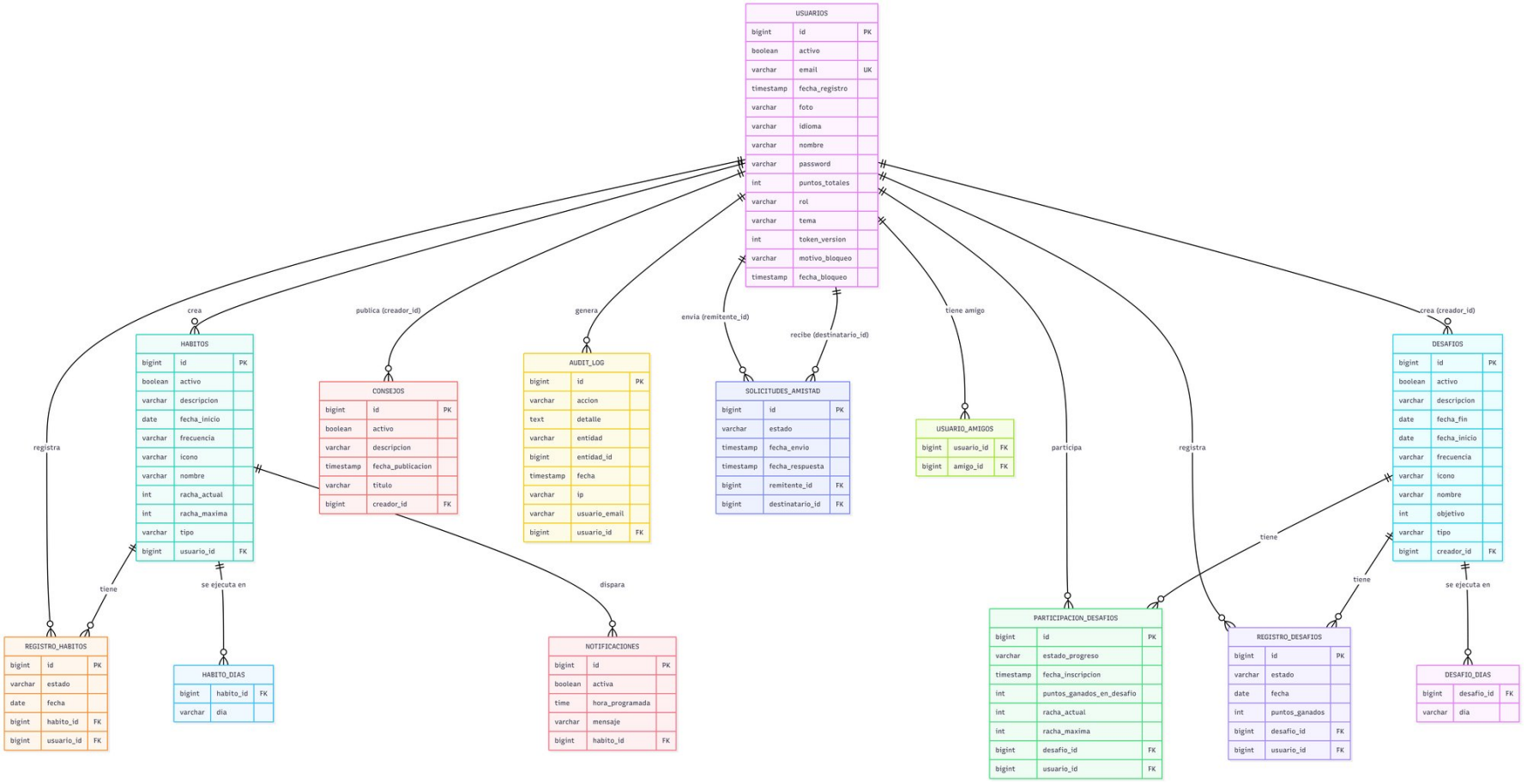
Clase: Amigos (usuario_amigos)

- **Descripción:** Tabla intermedia para la relación de amistad consolidada entre usuarios.
- **Atributos:** (Solo claves foráneas).
- **Relaciones:** Vincula un `usuario_id` con un `amigo_id`.

Clase: Registro de Auditoría (audit_log)

- **Descripción:** Historial de eventos y acciones relevantes en el sistema para trazabilidad.
- **Atributos:** `id`, `accion`, `detalle`, `entidad`, `entidad_id`, `fecha`, `ip`, `usuario_email`.
- **Relaciones:** Asociado a un `usuario_id` responsable de la acción.

Diagrama entidad-relación definitivo



4.3. Diseño de la interfaz de usuario

- **Mockups**

Los siguientes mockups se generaron con IA (Gemini) como referencia visual durante la fase de diseño, para validar los flujos de navegación y la estética de la app antes de la implementación.

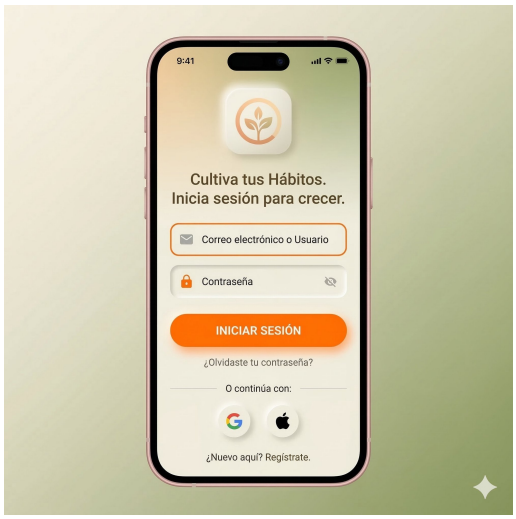


Figura 17: Login

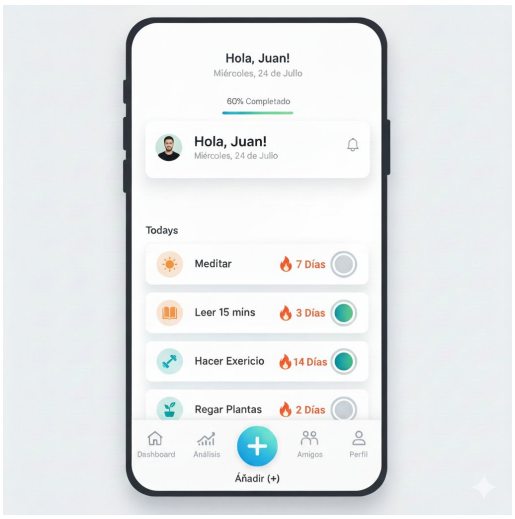


Figura 18: Dashboard

Figura 19: Estadísticas

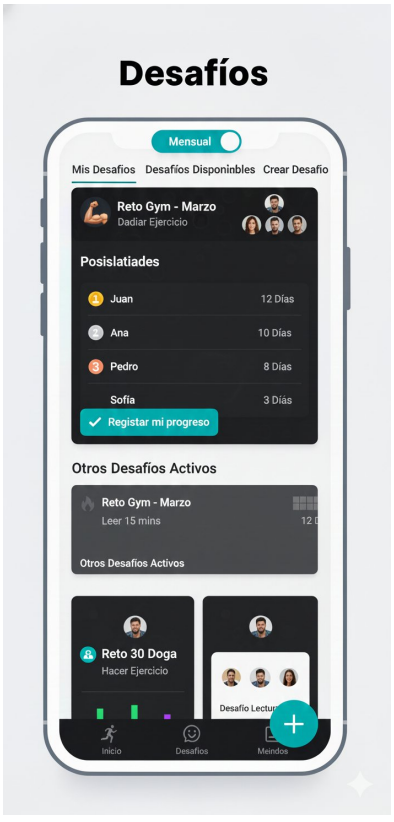


Figura 20: Desafíos

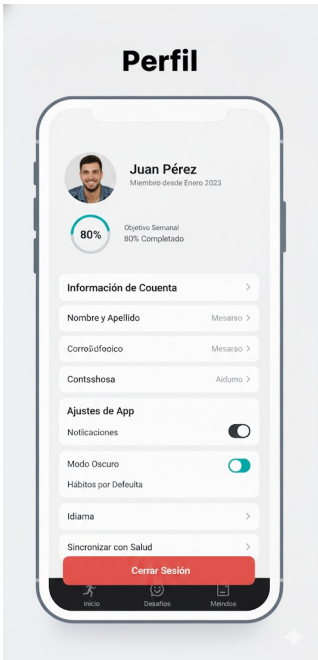


Figura 21: Perfil

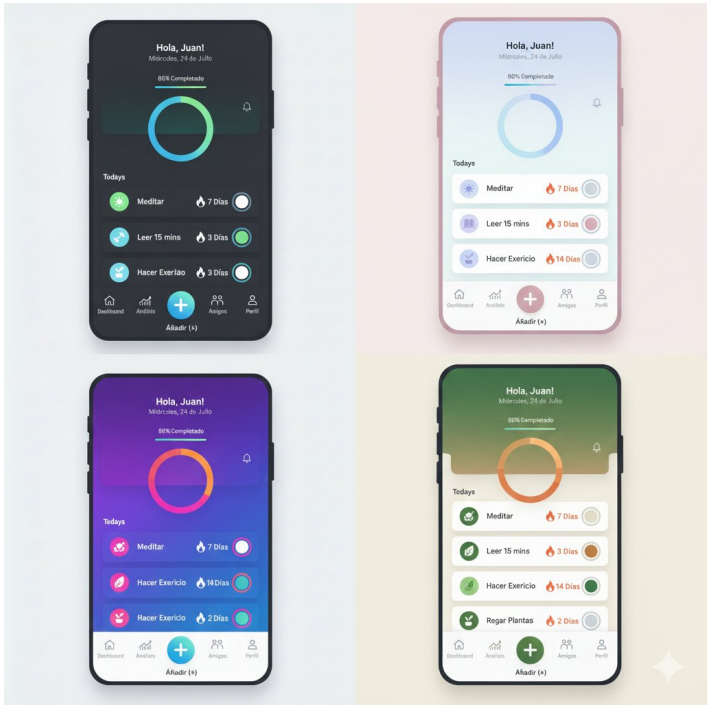


Figura 22: Dashboard con los 4 temas de color

- **Diagrama de navegación — App móvil**

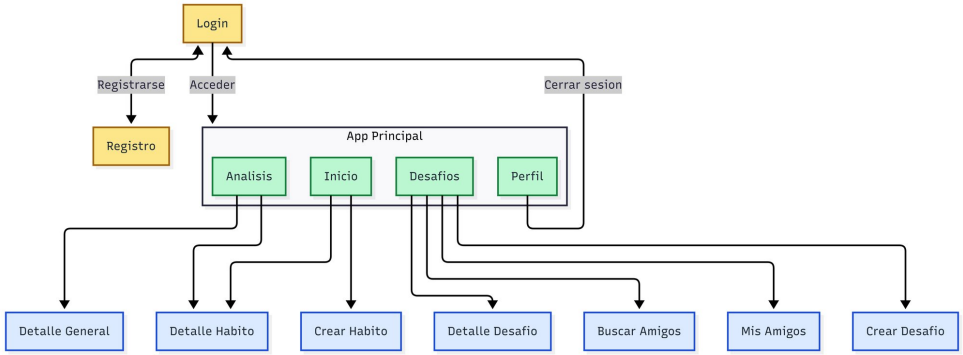


Figura 23: Diagrama de Navegación - App móvil

- **Diagrama de navegación — Panel de administración (Web)**

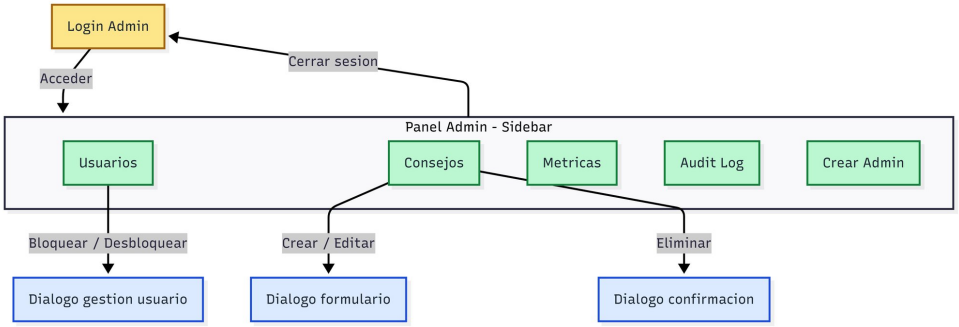
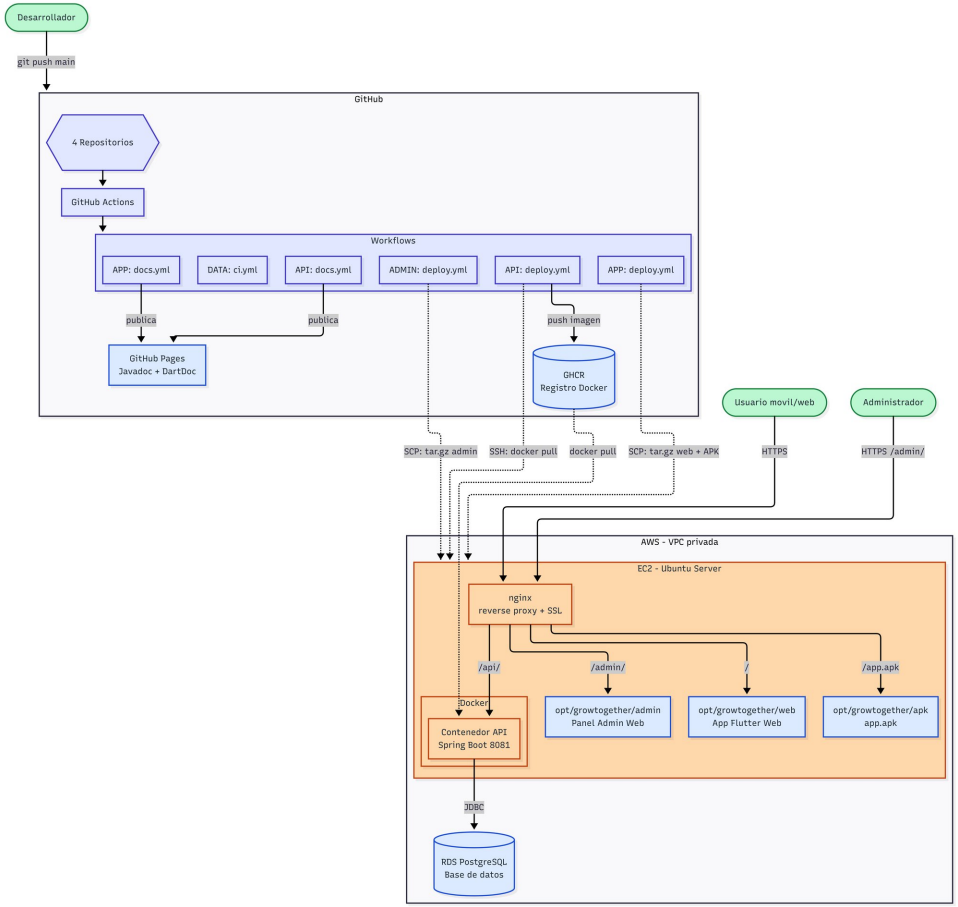


Figura 24: Diagrama de Navegación - App móvil

4.4. Otros diagramas y descripciones

Diagrama de despliegue GitHub - AWS



5. Implementación del proyecto

5.1. Estructura del proyecto

El proyecto se divide en cuatro subproyectos independientes dentro del repositorio raíz:

```
GrowTogether/
+-- GrowTogetherAPI/       # Backend - API REST (Spring Boot)
+-- GrowTogetherAPP/       # App móvil (Flutter)
+-- GrowTogetherADMIN/     # Panel de administración web (Flutter Web)
+-- GrowTogetherDATA/      # Paquete Dart compartido (modelos, repositorios
    , cliente HTTP)
```

La existencia de **GrowTogetherDATA** como paquete independiente es una decisión arquitectónica clave: tanto la app móvil como el panel de administración consumen los mismos modelos, repositorios y cliente HTTP, evitando duplicación de código y garantizando que cualquier cambio en el contrato de la API se propague de forma consistente.

Backend (GrowTogetherAPI) — Arquitectura por capas:

```
src/main/java/com/jordipatuel/GrowTogetherAPI/
+-- config/          # Configuración (seguridad, CORS, JWT)
+-- controller/      # Controladores REST (endpoints)
+-- dto/             # Objetos de transferencia de datos
+-- exception/       # Excepciones personalizadas
+-- model/           # Entidades JPA
|   +-- enums/       # Enumeraciones (roles, tipos, estados)
+-- repository/      # Repositorios Spring Data JPA
+-- service/         # Lógica de negocio
```

App móvil (GrowTogetherAPP) — Arquitectura por capas:

```
lib/
+-- core/
|   +-- feedback/    # Controlador y servicio de feedback háptico/sonoro
|   +-- l10n/        # Controlador de idioma (ValueNotifier)
|   +-- theme/       # 4 temas seleccionables y controlador (ValueNotifier)
|   +-- utils/       # Validadores, iconos, colores, snack helper,
    dashboard cache
+-- l10n/            # Archivos ARB de traducciones (es, ca, en)
+-- providers/       # Providers (ChangeNotifier) por feature
+-- screens/         # Pantallas y widgets reutilizables
+-- services/        # Servicios locales (notificaciones)
+-- main.dart        # Inyección de dependencias y arranque
```

Panel de administración (GrowTogetherADMIN) — Arquitectura por capas:

```
lib/
+-- core/            # Tema, configuración y utilidades
+-- providers/       # Providers del panel
+-- screens/
|   +-- admins/      # Crear administrador
|   +-- audit/       # Registro de auditoría
|   +-- consejos/    # Gestión de consejos
|   +-- metricas/    # Métricas del sistema
|   +-- usuarios/    # Gestión de usuarios
|   +-- login_admin_screen.dart
|   +-- main_layout.dart
+-- main.dart
```

Paquete compartido (GrowTogetherDATA):

```
lib/src/
+-- api/             # DioClient, ApiConfig, interceptores
+-- local/           # SecureStorage para tokens y preferencias
+-- models/          # Modelos de datos (Usuario, Habito, Desafio, etc.)
+-- repositories/    # Repositorios HTTP (Auth, Habito, Desafio, etc.)
```

5.2. Descripción de los módulos y componentes principales

Backend — Controladores (API REST)

Controlador	Descripción
AuthController	Registro, login y gestión de tokens JWT
UsuarioController	Gestión de perfil, búsqueda de usuarios y solicitudes de amistad
HabitoController	CRUD de hábitos y registro de cumplimiento diario
DesafioController	CRUD de desafíos, participación y ranking
NotificacionController	Gestión de recordatorios programados
AdminController	Gestión administrativa (usuarios, consejos, métricas, auditoría, alta de admins)
VersionController	Información de versión de la API para chequeo de compatibilidad

Backend — Modelos (Entidades JPA)

Entidad	Descripción
Usuario	Datos del usuario, credenciales, preferencias (tema, idioma) y estado de bloqueo
Habito	Hábito del usuario con tipo (positivo/negativo), icono y frecuencia
RegistroHabito	Registro diario de cumplimiento de un hábito
Desafio	Reto grupal con objetivo, fecha inicio/fin y creador
ParticipacionDesafio	Inscripción de un usuario en un desafío con puntuación y rachas
RegistroDesafio	Avance diario dentro de un desafío
Notificacion	Recordatorio programado asociado a un hábito
Consejo	Consejo publicado por administradores
SolicitudAmistad	Petición de amistad entre dos usuarios
AuditLog	Registro de auditoría de acciones relevantes en el sistema

App móvil — Pantallas

Pantalla	Descripción
LoginScreen	Inicio de sesión con email y contraseña
RegisterScreen	Registro de nuevo usuario
MainLayout	Layout principal con BottomNav de 4 pestañas y FAB contextual
DashboardScreen	Pantalla principal con carrusel de días, lista de hábitos y progreso
StatisticsScreen	Estadísticas y gráficas de progreso por hábito y globales
ChallengesScreen	Lista de desafíos, ranking y participación
ProfileScreen	Perfil del usuario, ajustes de tema, idioma y foto
CrearHabitoScreen	Formulario para crear un nuevo hábito (nombre, tipo, icono, frecuencia)
DetalleHabitoScreen	Detalle de un hábito con calendario de historial mensual
CrearDesafioScreen	Formulario para crear un nuevo desafío
DetalleDesafioScreen	Detalle de un desafío con ranking de participantes
DetalleGeneralScreen	Detalle global de progreso del usuario
AmigosScreen	Lista de amigos y gestión de solicitudes pendientes
BuscarAmigosScreen	Búsqueda de usuarios y envío de solicitudes

Panel de administración — Pantallas

Pantalla	Descripción
LoginAdminScreen	Inicio de sesión exclusivo para administradores
MainLayout	Layout con sidebar fijo y contenido conmutable
UsuariosScreen	Listado, búsqueda, bloqueo y desbloqueo de usuarios
ConsejosScreen	Alta, edición y eliminación de consejos
MetricasScreen	Métricas globales del sistema (usuarios activos, hábitos, desafíos)
AuditScreen	Consulta del registro de auditoría con filtros
CrearAdminScreen	Alta de nuevos administradores

Características implementadas

- 4 temas de color (oscuro, claro, morado, naturaleza)
- Internacionalización en 3 idiomas (castellano, inglés, valenciano)
- Foto de perfil desde cámara o galería
- Hábitos positivos y negativos con iconos personalizables
- Carrusel de días para marcar hábitos de días anteriores
- Consejo diario con temática de sostenibilidad
- Sistema de amigos con solicitudes pendientes, aceptación y rechazo
- Desafíos grupales con inscripción, ranking y sistema de puntos
- Gamificación mediante puntos totales, rachas actuales y rachas máximas
- Notificaciones locales programadas para recordar hábitos (pendiente)
- Detección automática de pérdida de conexión con banner persistente
- Autenticación JWT con cifrado BCrypt y versionado de tokens
- Panel de administración web con gestión completa de la plataforma
- Registro de auditoría de acciones relevantes en el sistema

5.3. Despliegue de la aplicación

El despliegue de GrowTogether está completamente automatizado mediante **GitHub Actions**, de forma que cualquier cambio en la rama `main` de cualquiera de los cuatro repositorios desencadena la construcción y publicación del componente afectado sin intervención manual.

5.3.1. Visión general

El flujo completo de despliegue es el siguiente:

1. El desarrollador realiza un `git push` a la rama `main`.
2. **GitHub Actions** detecta el cambio y arranca el workflow correspondiente.
3. El workflow construye el artefacto: imagen Docker para la API, build web Flutter para la app y el panel admin, o APK para Android.
4. La imagen Docker de la API se publica en **GitHub Container Registry (GHCR)**, el registro de imágenes de GitHub.
5. El workflow se conecta por **SSH** a la instancia **EC2** alojada en la **VPC** de AWS y despliega el nuevo artefacto.
6. **nginx**, instalado directamente en la EC2, recibe las peticiones HTTPS de internet y las enruta hacia el componente correspondiente según la URL.

5.3.2. Infraestructura en AWS

La infraestructura se compone de dos servicios principales de Amazon Web Services:

- **EC2 (Elastic Compute Cloud)**: instancia tipo `t3.micro` con Ubuntu Server que aloja nginx, el contenedor Docker de la API y los archivos estáticos de la app y el panel admin.
- **RDS (Relational Database Service)**: instancia gestionada de PostgreSQL que almacena toda la información de la plataforma.

Ambos recursos viven dentro de una **VPC** (Virtual Private Cloud), la red privada de AWS, y están protegidos por **Security Groups** que actúan como cortafuegos a nivel de red: la EC2 solo expone los puertos 80 y 443 a internet, y la RDS solo acepta conexiones desde la EC2.

El dominio público es `growtogether.jordipatuel.com`, gestionado con certificado SSL de **Let's Encrypt** mediante **certbot**, que renueva el certificado automáticamente.

- **5.3.3. Empaquetado con Docker**

La API Spring Boot se empaqueta en una imagen Docker utilizando un **Dockerfile multi-stage** que separa el proceso en dos fases:

1. **Fase de construcción**: utiliza una imagen con el JDK 17 completo y Gradle para compilar el código fuente y generar el JAR ejecutable.
2. **Fase de ejecución**: utiliza una imagen con solo el JRE (entorno de ejecución), mucho más ligera, y copia únicamente el JAR generado en la fase anterior.

- **5.3.4. Registro de imágenes: GHCR**

Las imágenes Docker construidas por GitHub Actions se publican en **GitHub Container Registry (GHCR)**, el servicio de registro de imágenes integrado en GitHub. Se ha escogido GHCR frente a alternativas como Docker Hub o Amazon ECR por su integración nativa con GitHub Actions (autenticación automática con `GITHUB_TOKEN`, sin necesidad de crear credenciales adicionales) y porque las imágenes publicadas como públicas disponen de almacenamiento y ancho de banda ilimitados de forma gratuita. Como ventaja adicional, al ser la imagen pública, la instancia EC2 puede descargarla con `docker pull` sin necesidad de autenticarse contra el registro.

Cada build genera dos etiquetas de la misma imagen:

- `:latest` — apunta siempre a la última versión y la utiliza la EC2 para el despliegue.
- `:sha-{commit}` — etiqueta inmutable asociada al commit concreto, que permite volver a una versión anterior en caso de fallo (rollback).

- **5.3.5. Workflows de GitHub Actions**

El sistema dispone de **ocho workflows** distribuidos entre los cuatro repositorios:

Repositorio	Workflow	Función
GrowTogetherAPI	`deploy.yml`	Construye la imagen Docker, la publica en GHCR y la despliega en EC2
GrowTogetherAPI	`docs.yml`	Genera la documentación Javadoc y la publica en GitHub Pages
GrowTogetherAPP	`deploy.yml`	Compila la app web Flutter y el APK Android, y los despliega en EC2
GrowTogetherAPP	`docs.yml`	Genera la documentación DartDoc y la publica en GitHub Pages
GrowTogetherADMIN	`deploy.yml`	Compila el panel admin web y lo despliega en EC2 bajo `/admin/`
GrowTogetherADMIN	`docs.yml`	Genera la DartDoc del panel admin y la publica en GitHub Pages
GrowTogetherDATA	`ci.yml`	Ejecuta los tests del paquete compartido en cada push
GrowTogetherDATA	`docs.yml`	Genera la DartDoc del paquete compartido y la publica en GitHub Pages

El workflow de la app móvil incluye además un job específico que compila el **APK Android** en modo release, lo sube como **artifact** descargable desde la interfaz de GitHub (con 30 días de retención) y también lo publica en la EC2 en la ruta pública `/app.apk` para que los usuarios puedan instalárselo directamente desde el navegador.

- **5.3.6. Servidor nginx y enrutamiento**

nginx corre **directamente en el host de la EC2** (no dentro de Docker) por dos razones: facilita la gestión de certificados SSL con certbot y permite servir archivos estáticos sin pasar por el contenedor.

La configuración de nginx actúa como **reverse proxy** y reparte las peticiones según la ruta solicitada:

URL	Destino
`https://growtogether.jordipatuel.com/`	App móvil Flutter Web (archivos en `/opt/growtogether/web/`)
`https://growtogether.jordipatuel.com/admin/`	Panel admin Flutter Web (archivos en `/opt/growtogether/admin/`)
`https://growtogether.jordipatuel.com/api/`	API Spring Boot (proxy a `127.0.0.1:8081`)
`https://growtogether.jordipatuel.com/app.apk`	Descarga directa del APK (archivo en `/opt/growtogether/apk/`)

El contenedor Docker de la API **solo expone su puerto al localhost de la EC2** (`127.0.0.1:8081`), no a internet. Esto significa que la única forma de llegar a la API es a través de nginx, lo cual añade una capa de seguridad y permite centralizar el SSL en un único punto.

- **5.3.7. Documentación automática**

Los workflows `docs.yml` presentes en los cuatro repositorios generan automáticamente la documentación técnica del código en cada push a `main`:

- **API**: ejecuta `./gradlew javadoc` para generar la **Javadoc** del código Java.
- **App, panel admin y paquete compartido**: ejecutan `dart doc` para generar la **DartDoc** del código Dart de cada uno.

En todos los casos, la documentación HTML resultante se publica como **artifact** descargable y, opcionalmente, se despliega en **GitHub Pages** para que sea accesible públicamente desde una URL pública. Esto garantiza que la documentación esté siempre sincronizada con el código sin tener que regenerarla manualmente.

- **5.3.8. Gestión de secretos**

Toda la información sensible (claves SSH, credenciales de la EC2, claves de repositorios privados) se gestiona mediante **GitHub Secrets**, que son variables cifradas accesibles únicamente desde los workflows. Los principales secretos configurados son:

Secreto	Uso
`EC2_HOST`	IP o dominio de la instancia EC2
`EC2_USER`	Usuario SSH de la EC2
`EC2_SSH_KEY`	Clave privada SSH para conectar a la EC2
`DATA_REPO_KEY`	Clave SSH (deploy key) del repositorio privado `GrowTogetherDATA`

Las credenciales de base de datos y la `SECRET_KEY` de JWT viven en un fichero `.env` que reside únicamente en la EC2, fuera de Git, y que `docker-compose` carga automáticamente al arrancar el contenedor de la API.

5.4. Capturas de pantalla

- **App Móvil**

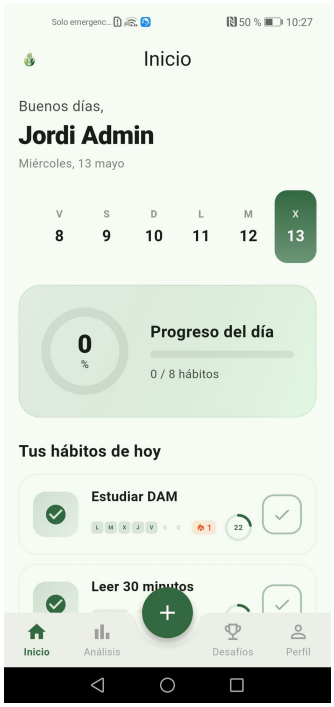


Figura 25: Pantalla de bienvenida al arrancar la aplicación

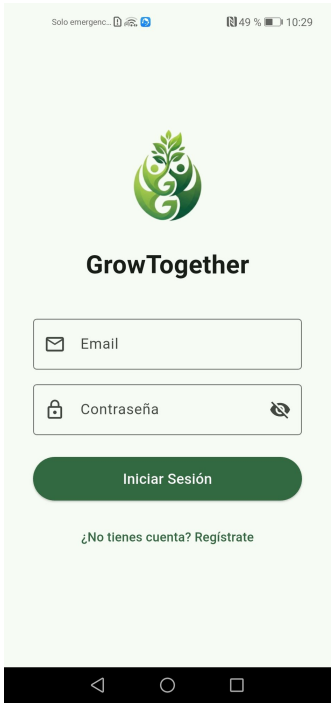


Figura 26: Inicio de sesión con email y contraseña

Figura 27: Dashboard con hábitos del día y vista de análisis

Figura 28: Formulario de creación de un nuevo hábito

Figura 29: Listado de desafíos activos y disponibles

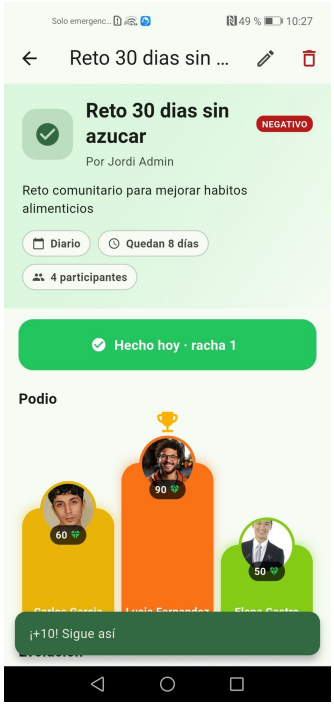


Figura 30: Ranking de participantes dentro de un desafío

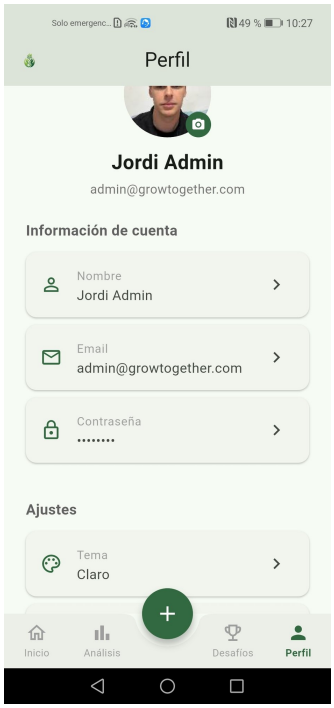


Figura 31: Perfil del usuario con datos y ajustes

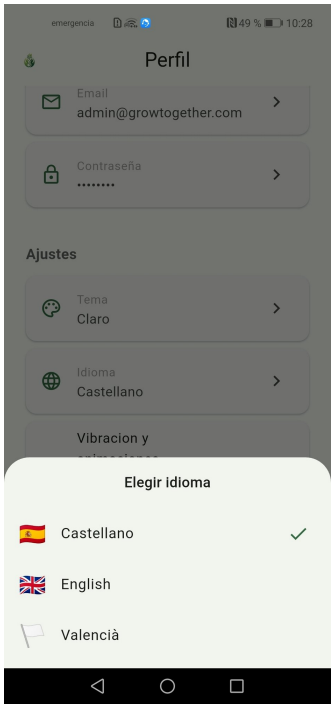


Figura 32: Selector de idioma (castellano, inglés y valenciano)

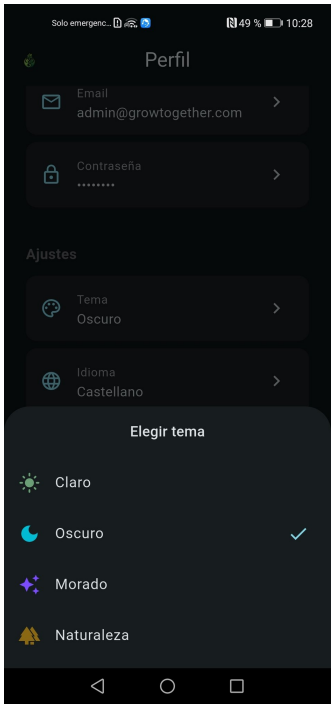


Figura 33: Selector de tema (claro, oscuro, morado y naturaleza)

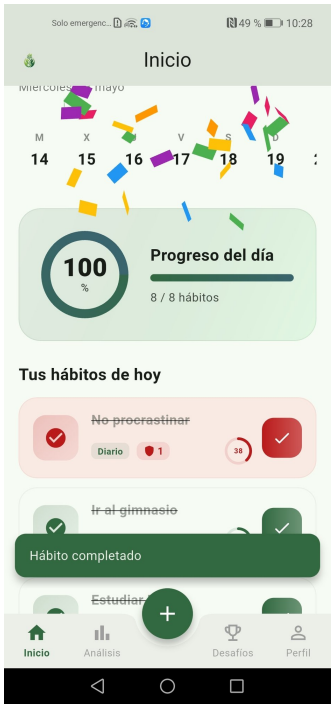


Figura 34: Animación de confeti al completar todos los hábitos del día

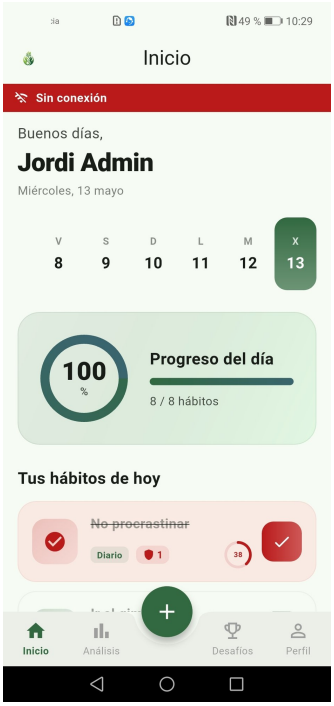


Figura 35: Banner informativo cuando se pierde la conexión a internet

- **Panel de Administración (Web)**

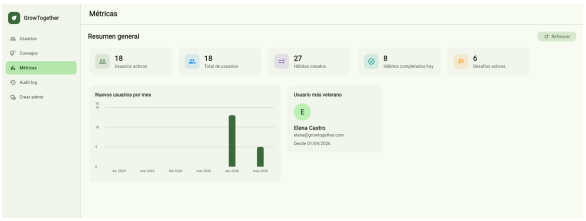


Figura 36: Pantalla de métricas globales del sistema

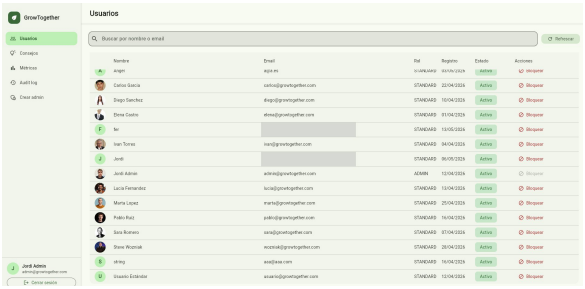


Figura 37: Gestión de usuarios con búsqueda, bloqueo y desbloqueo

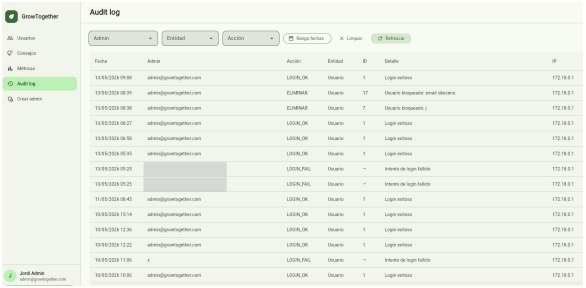


Figura 38: Registro de auditoría con filtros por admin, entidad y rango de fechas

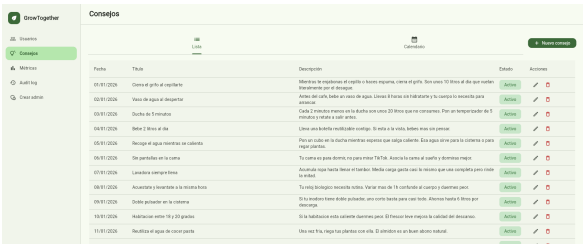


Figura 39: Gestión de consejos publicados desde el panel

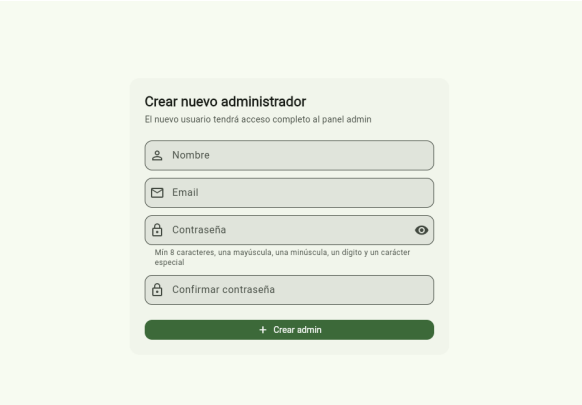


Figura 40: Formulario de alta de nuevos administradores

6. Estudio de los resultados obtenidos

6.1. Evaluación respecto a los objetivos iniciales

El proyecto ha cumplido en gran medida los objetivos planteados en el apartado 1.4. A continuación se repasa el grado de consecución de cada uno:

- **Sistema de hábitos con estadísticas y rachas**: completado. Incluye CRUD completo de hábitos, calendario mensual con el historial, cálculo de rachas actuales y máximas y un panel de estadísticas con métricas por hábito y globales.
- **Módulo de desafíos con ranking y puntuación**: completado. Los usuarios pueden crear desafíos, inscribirse, ver el ranking en tiempo real y ganar puntos que se integran con el sistema general de gamificación.
- **Sistema de amistad con solicitudes y búsqueda**: completado. Está implementada la búsqueda de usuarios, el envío y recepción de solicitudes, la aceptación o rechazo, y la gestión completa de la lista de amigos.
- **Panel de administración web**: completado. Permite gestionar usuarios (bloqueo y desbloqueo con motivo), publicar y editar consejos, consultar métricas globales, revisar el registro de auditoría y dar de alta a nuevos administradores.
- **Consejos diarios sobre sostenibilidad**: completado. Los administradores publican consejos desde el panel y la app móvil muestra el consejo del día.
- **Despliegue en infraestructura cloud**: completado. La aplicación está desplegada en AWS (EC2 + RDS) con un *pipeline* de CI/CD automatizado, dominio propio y certificado SSL.

6.2. Problemas encontrados y soluciones aplicadas

A lo largo del desarrollo han surgido diferentes problemas técnicos. Los más relevantes se documentan a continuación por su valor formativo:

Problemas de concurrencia y estado en la app móvil

- **Race condition en el cierre de sesión**: al pulsar "Cerrar sesión", el borrado del token era asíncrono y la navegación al login se hacía sin que terminase. La pantalla de login, comprobaba si había sesión activa y encontraba aún el token, y volvía a iniciar sesión automáticamente. La solución fue convertir el método de navegación a `async` y aplicar `await` a la operación de borrado.

Problemas de diseño y renderizado de widgets

- **Pantalla en blanco en el registro de auditoría**: la tabla del panel admin no se mostraba pese a recibir datos del backend. Tras descartar problemas de red y de parseo, se identificó la causa en un `Spacer()` colocado dentro de un `Wrap`, una combinación inválida en Flutter que rompía el árbol de widgets durante el render. Retirar el `Spacer` restauró el comportamiento esperado.
- **Posicionamiento y layout en Flutter**: a lo largo del desarrollo de las interfaces ha sido frecuente que los widgets no se comportaran exactamente como se esperaba en cuanto a alineación, tamaños relativos o espaciados.
- **Pixel overflow en pantallas pequeñas**: el aviso amarillo y negro de "RenderFlex overflowed" ha aparecido con frecuencia. Cada caso ha requerido envolver el contenido con `Expanded`, `Flexible`, `FittedBox` o `SingleChildScrollView` según la naturaleza del widget afectado.

Problemas de evolución del modelo de datos

- **Cambios continuos en la base de datos**: durante todo el desarrollo han ido apareciendo nuevos campos y modelos a medida que se concretaban requisitos (estado de bloqueo de usuarios, motivo de bloqueo, versionado del token JWT, registros de auditoría, etc.). Cada uno de estos cambios obligaba a modificar la entidad JPA, generar la migración correspondiente y propagar el cambio a los modelos del paquete `GrowTogetherDATA` y al panel admin.

Problemas de despliegue e infraestructura

- **Acumulación de imágenes Docker en la EC2**: la instancia `t3.micro` utilizada para producción tiene un disco limitado y los redeploys repetidos llenaban el espacio. Se añadió un paso de limpieza al workflow que conserva las tres últimas imágenes y elimina las más antiguas tras cada despliegue.
- **Sincronización de la base de datos en producción**: para evitar pérdidas accidentales, el perfil de producción usa `ddl-auto=validate` en lugar de `update`. Cualquier cambio en las entidades JPA exige aplicar manualmente la migración (`ALTER TABLE`) sobre RDS antes de hacer push, lo cual añade un paso operativo pero protege la integridad de los datos.
- **Configuración de red en AWS**: el primer despliegue requirió varios intentos para configurar correctamente la IP elástica, los *security groups* (puertos 22, 80, 443 abiertos selectivamente) y las reglas de acceso entre la instancia EC2 y la base de datos RDS. Cada error de configuración se traducía en *timeouts* o conexiones rechazadas que no siempre eran fáciles de diagnosticar.
- **Documentación automática con repositorio privado**: los workflows de GitHub Actions generaban correctamente la Javadoc y la DartDoc, pero la publicación en GitHub Pages no estaba disponible al tener los repositorios en modo privado sin plan de pago. Hasta que se hicieron públicos, la documentación solo era accesible como *artifact* descargable manualmente desde la interfaz de cada ejecución del workflow.

Funcionalidades parcialmente implementadas

- **Módulo de notificaciones locales**: el código que programa las notificaciones está implementado e integrado en la app, pero durante las pruebas se ha detectado un comportamiento inconsistente que falta por verificar y depurar. Se ha mantenido la funcionalidad en el código fuente y se documenta como pendiente de validación antes de considerarla operativa en producción.
- **Recuperación de contraseña**: el endpoint `POST /auth/recuperar` está expuesto en la API y se invoca correctamente desde la app, pero la implementación actual es un *stub* que solo devuelve un mensaje informativo sin enviar el email real al usuario. Falta integrar un servicio de envío de correos (SMTP, SendGrid o similar) que genere un enlace temporal de restablecimiento.

6.3. Futuras mejoras y ampliaciones

A continuación se enumeran las mejoras que se contemplan para una segunda versión del proyecto:

Funcionalidad nueva

- **Soporte para iOS**: actualmente la app móvil solo se distribuye como APK Android. Flutter permite compilar para iOS con cambios mínimos, pero se requiere una cuenta de Apple Developer (~99 €/año) y un equipo macOS para la generación del IPA.
- **Notificaciones push remotas**: el sistema actual usa notificaciones locales programadas en el dispositivo. Una mejora natural sería integrar **Firebase Cloud Messaging** para enviar recordatorios y avisos de desafíos directamente desde el backend.
- **Modo offline real**: aunque la app detecta la pérdida de conexión y muestra un banner informativo, no permite trabajar sin conexión. Implementar una capa de persistencia local (SQLite o Drift) con sincronización al reconectar mejoraría significativamente la experiencia.
- **Sistema de logros y niveles**: ampliar la gamificación con insignias, niveles de experiencia y recompensas visuales para reforzar la motivación a largo plazo.
- **Integraciones con plataformas de salud**: conectar con **Google Fit** y **Apple Health** para sincronizar automáticamente hábitos relacionados con actividad física y sueño.
- **OAuth y confirmación de cuenta**: incorporar inicio de sesión con proveedores externos (Google, Apple) y, alternativamente, un sistema de confirmación de email durante el registro para validar la titularidad de la cuenta.
- **Chat interno en los desafíos**: añadir una conversación grupal dentro de cada desafío para que los participantes puedan animarse, comentar sus avances y reforzar el componente social del proyecto.
- **Widgets para la pantalla de inicio**: crear widgets nativos de Android para visualizar el progreso del día y marcar hábitos como completados sin necesidad de abrir la aplicación.
- **Sistema de tickets de soporte**: incluir un formulario de contacto desde la propia app para que los usuarios puedan reportar incidencias o sugerencias, gestionado posteriormente desde el panel de administración.
- **Tutorial al primer uso**: incorporar un *onboarding* interactivo que se muestre la primera vez que el usuario abre la app tras registrarse, explicando cómo crear hábitos, marcarlos

completados, unirse a desafíos y añadir amigos. Reduciría la curva de aprendizaje inicial y la tasa de abandono temprano.

Mejoras de experiencia de usuario

- **Empty states ilustrados**: enriquecer todas las pantallas que pueden quedarse vacías (sin hábitos, sin amigos, sin desafíos) con ilustraciones y mensajes que orienten al usuario hacia la siguiente acción.
- **Animaciones y microinteracciones**: ampliar las animaciones existentes con transiciones más fluidas, efectos al completar hábitos y feedback visual en momentos clave como el cierre de un desafío o el desbloqueo de una racha.
- **Feedback sonoro**: incorporar sonidos sutiles para refuerzos positivos (hábito completado, racha conseguida, victoria en un desafío) con opción de silenciarlos desde los ajustes.

Arquitectura y mantenimiento

- **Cobertura de tests ampliada**: aumentar la cobertura de tests unitarios y de integración, especialmente en los flujos críticos (autenticación, registro de hábitos y cálculo de rachas).
- **Optimización del cálculo de rachas**: el recálculo actual hace una consulta a base de datos por cada día retrocedido, lo que se traduce en cientos de consultas para rachas largas. Cargar los registros del hábito en una sola consulta y recorrer en memoria reduciría la latencia al marcar hábitos.

7. Conclusiones

7.1. Relación con los contenidos de los módulos

Una de las cosas que más me ha llamado la atención al terminar este proyecto es que al final he acabado usando algo de prácticamente todos los módulos del ciclo. En clase cada cosa se ve por separado y a veces ni te imaginas cuándo lo vas a usar, pero juntando todo en un solo proyecto se nota un montón cómo todo encaja.

- **Acceso a datos**: básicamente todo el backend. Las entidades con JPA, las relaciones entre tablas (`@OneToMany`, `@ManyToOne` y la auto-relación de los amigos con `@ManyToMany`), las validaciones y los repositorios de Spring Data. Mover la base de datos del PC a la de Amazon (RDS) fue básicamente hacer un *backup* en local y subirlo, pero me sirvió para ver cómo se gestiona una base de datos cuando ya no la tienes en tu máquina.
- **Desarrollo de interfaces**: aquí tengo dos productos distintos pensados para situaciones distintas. La app móvil está pensada para usar todos los días en el teléfono, rápido y a una mano, con cuatro temas de color y tres idiomas. El panel de administración es de escritorio, con tablas y filtros, pensado para alguien que entra de vez en cuando a gestionar cosas. Las dos consumen los mismos modelos y repositorios del paquete `GrowTogetherDATA`, así no tienes que mantener el mismo código dos veces.
- **Programación multimedia y dispositivos móviles**: toda la app está hecha con Flutter para Android, siguiendo una arquitectura por capas con *providers* (`ChangeNotifier`) para gestionar el estado, *services* para las funcionalidades del dispositivo y *screens* para las pantallas. He usado cosas típicas de móvil como la cámara y la galería para la foto de perfil, el almacenamiento seguro para guardar el token, las notificaciones locales y el detectar si hay internet o no.
- **Sistemas de gestión empresarial**: el panel de admin es lo que sería el “back-office” de cualquier app comercial, y es algo que también he visto en mi periodo de prácticas. Puedes gestionar usuarios (bloquearlos, desbloquearlos), publicar consejos para todos, ver métricas de uso y, lo más importante, todo lo que hace un admin queda guardado en un registro de auditoría con la IP y el momento exacto. Eso último lo añadí porque en una app real es algo obligatorio.
- **Programación de servicios y procesos**: en el backend hay tareas que se ejecutan solas todos los días (con `@Scheduled` de Spring) para mantener el historial de hábitos limpio.

En el frontend he tenido que aprender a manejar bien todo lo asíncrono con `async/await`, y de hecho uno de los bugs más raros del proyecto fue una *race condition* al cerrar sesión que me costó pillar.

- **Digitalización**: aquí entra todo el tema cloud. Tener la app desplegada en AWS con su dominio, su certificado SSL y un *pipeline* automático de despliegue en GitHub Actions me ha hecho ver lo que cuesta de verdad poner algo en producción. También he tocado Docker para empaquetar la API y subirla al registro de imágenes de GitHub.
- **Sostenibilidad**: los consejos que ven los usuarios cada día van sobre temas de medio ambiente (reciclar, ahorrar luz, moverse en bici, etc.). Aprovechando que la app ya se basa en cambiar hábitos del día a día, encajaba bastante bien empujar al usuario hacia hábitos que también sean buenos para el planeta.

7.2. Valoración personal del proyecto

Después de haber trabajado en el proyecto, mi visión sobre lo que he aprendido ha aumentado mucho. Creo que es aquí donde realmente se aprende: cuando empiezas a tener problemas y tienes que buscar la forma de solucionarlos. Desde mi punto de vista, la idea de la app ha sido lo de menos. Sí que es algo que sentía que necesitaba y por eso tiré con ella, pero todo lo que hay detrás es lo que más me ha aportado: pensar cómo desplegar la app automáticamente después de cada push a GitHub, estar atento a los posibles fallos de seguridad que me han enseñado en las prácticas a tener en cuenta (como el *rate limit*, por ejemplo), y asegurarme de cosas como que un usuario no pueda entrar a la cuenta de otro o de hacer bien el *soft delete*. Estoy contento con lo que voy a entregar por todo lo aprendido. Sinceramente, pensaba que era un trámite más del ciclo, pero en estos meses mi aprendizaje ha sido mucho mayor de lo que esperaba, supongo que por ponerlo todo en práctica a la vez y juntar los módulos de clase en un único proyecto.

Aun con lo contento que estoy del resultado, hay un par de cosas que haría distinto si tuviera que repetir el proyecto. La primera es dedicar más tiempo a planificar, pero no un plan general de fases, sino sobre todo las microtareas del día a día: muchas veces me ponía a trabajar y saltaba de una cosa a otra sin haber decidido antes “hoy toca esta tarea concreta”, y eso me hacía dispersarme. La segunda es mejorar mi uso de Git con las ramas, porque en varios momentos me habrían ahorrado algún disgusto al poder probar cambios sin tocar `main` directamente. Y la tercera, escribir la memoria en paralelo al desarrollo y no dejarla para el final, ya que al hacerla

ahora me he encontrado intentando recordar decisiones que tomé hace meses y que ya no tenía tan frescas.

Me gustaría mencionar también mi punto de vista sobre la IA y sobre cómo creo que debemos afrontarla los *juniors*. Es algo que está avanzando de forma indiscutible, y debemos estar a su lado para crecer con ella, pero teniendo cuidado de no delegarlo todo, porque si lo hacemos no aprenderemos. Me he dado cuenta de que a largo plazo el saber —o mejor dicho, memorizar— la sintaxis de un lenguaje ya no será tan necesario. Esto traerá nuevos retos y un peldaño más en el esfuerzo de aprender arquitectura y estructuras de datos: pasaremos a ser más bien orquestadores que dirigen hacia dónde va la herramienta y la guían para que, con nuestro conocimiento, construya. No es que vayamos a crear una app o una solución milagrosa multimillonaria en dos días, como venden muchos por internet, pero sí creo que a los experimentados les da un empujón y les quita la fricción de escribir; en consecuencia, podrán dedicar más tiempo a pensar en la solución.

Llevaba años con ganas de estudiar programación y, ahora que termino el ciclo, me voy con la sensación de haber acertado al dar el paso. Una de las cosas que más me gusta de este mundo es que el aprendizaje no se acaba nunca: no solo de código, también de redes, dispositivos, nuevas tecnologías o incluso de cómo funcionan las empresas, algo que descubrí en las prácticas al trabajar con ERPs y CRMs, donde para construir una buena solución primero hay que entender el negocio que hay detrás. Esa curiosidad por seguir aprendiendo de todo es lo que me llevo del ciclo.

8. Bibliografía y recursos utilizados

8.1. Libros

- **James Clear** — *Hábitos Atómicos*. Inspiración inicial para el proyecto.

8.2. Documentación oficial

- **Flutter documentation** https://docs.flutter.dev
- **Dart language tour** https://dart.dev/language
- **Spring Boot Reference Documentation**
  https://docs.spring.io/spring-boot/docs/current/reference/html/
- **Spring Security Reference**
  https://docs.spring.io/spring-security/reference/
- **Spring Data JPA Reference**
  https://docs.spring.io/spring-data/jpa/reference/
- **PostgreSQL 16 Documentation**
  https://www.postgresql.org/docs/16/
- **AWS EC2 User Guide**
  https://docs.aws.amazon.com/ec2/
- **AWS RDS for PostgreSQL**
  https://docs.aws.amazon.com/rds/
- **Docker Documentation**
  https://docs.docker.com
- **GitHub Actions documentation**
  https://docs.github.com/actions
- **Working with the GitHub Container Registry**
  https://docs.github.com/packages/working-with-a-github-packages-registry/working-with-the-container-registry
- **Certbot documentation (Let's Encrypt)** https://eff-certbot.readthedocs.io
- **nginx documentation** https://nginx.org/en/docs/
- **Material Design 3 Guidelines** https://m3.material.io

8.3. Paquetes Dart y librerías Java

- **pub.dev**: provider, dio, flutter_secure_storage, image_picker, connectivity_plus, flutter_local_notifications, data_table_2, intl, fl_chart, confetti, permission_handler.
  https://pub.dev
- **Maven Central**: jjwt (jsonwebtoken), Lombok, SpringDoc OpenAPI, BCrypt.
  https://central.sonatype.com

8.4. Recursos formativos y comunidad

- **MoureDev** (Brais Moure). Canal de YouTube con tutoriales sobre Flutter, Spring Boot y despliegue en AWS.
  https://www.youtube.com/@MoureDev
- **Widget Wisdom**. Canal de YouTube sobre Flutter y diseño de interfaces.
  https://www.youtube.com/@widgetwisdom
- **Flutter Mapp**. Canal de YouTube con tutoriales prácticos de Flutter.
  https://www.youtube.com/@FlutterMapp
- **Ales Dev**. Canal de YouTube sobre desarrollo Flutter y diseño UI con Figma.
  https://www.youtube.com/@alesdevstudio
- **HeyFlutter**. Canal de YouTube con cursos completos de Flutter.
  https://www.youtube.com/@HeyFlutter

8.5. Herramientas

- **IntelliJ IDEA Community** — IDE para el backend Java.
- **Visual Studio Code** — Editor para el desarrollo Flutter.
- **DBeaver** — Cliente gráfico para PostgreSQL y RDS.
- **Figma** — Diseño y prototipado de la interfaz de usuario.
- **draw.io** — Edición de los diagramas UML, ER y de despliegue.
- **Mermaid** — Generación de diagramas de navegación y arquitectura embebidos como código.
- **Canva** — Diseño de la portada y recursos gráficos complementarios.
- **Git** y **GitHub Desktop** — Control de versiones y colaboración.
- **Pandoc + LaTeX (XeLaTeX)** con plantilla *Eisvogel* — Generación del PDF de esta memoria a partir de Markdown.
- **Gemini** y **Claude** — Asistentes de IA usados para consultas técnicas y apoyo en la redacción de documentación y tests.

9. Anexos

9.1. Código fuente

El código fuente completo del proyecto está disponible en los siguientes repositorios públicos de GitHub:

- **GrowTogetherAPI** — Backend Spring Boot
  https://github.com/devPatuel/GrowTogetherAPI
- **GrowTogetherAPP** — App móvil Flutter
  https://github.com/devPatuel/GrowTogetherAPP
- **GrowTogetherADMIN** — Panel de administración web
  https://github.com/devPatuel/GrowTogetherADMIN
- **GrowTogetherDATA** — Paquete Dart compartido
  https://github.com/devPatuel/GrowTogetherDATA

9.2. Guía de instalación y uso

La aplicación móvil está disponible como APK Android descargable directamente desde el dominio del proyecto:

- **Descarga del APK**
  https://growtogether.jordipatuel.com/app.apk

Para instalarlo es necesario habilitar la opción “Permitir instalación desde fuentes desconocidas” en los ajustes de Android, ya que la app no está publicada en Google Play.

Como alternativa, si no se desea instalar el APK, la misma aplicación está accesible como aplicación web desde cualquier navegador, sin necesidad de descargar nada:

- **Versión web de la app**
  https://growtogether.jordipatuel.com
- **Panel de administración**
  https://growtogether.jordipatuel.com/admin/

El **panel de administración** se ha diseñado pensando para su uso en escritorio, por lo que se recomienda acceder a él desde un ordenador. En dispositivos móviles la interfaz queda comprimida y algunas tablas pierden legibilidad.

9.3. Documentación adicional

La documentación técnica del código se genera y publica automáticamente en cada despliegue mediante GitHub Actions, y está disponible públicamente en GitHub Pages:

- **Javadoc del backend (GrowTogetherAPI)**
  https://devpatuel.github.io/GrowTogetherAPI/
- **DartDoc de la app móvil (GrowTogetherAPP)**
  https://devpatuel.github.io/GrowTogetherAPP/
- **DartDoc del panel de administración (GrowTogetherADMIN)**
  https://devpatuel.github.io/GrowTogetherADMIN/
- **DartDoc del paquete compartido (GrowTogetherDATA)**
  https://devpatuel.github.io/GrowTogetherDATA/

Adicionalmente, la API expone su especificación OpenAPI con interfaz interactiva **Swagger UI**, que permite probar todos los endpoints desde el navegador sin necesidad de herramientas externas:

https://growtogether.jordipatuel.com/swagger-ui/index.html