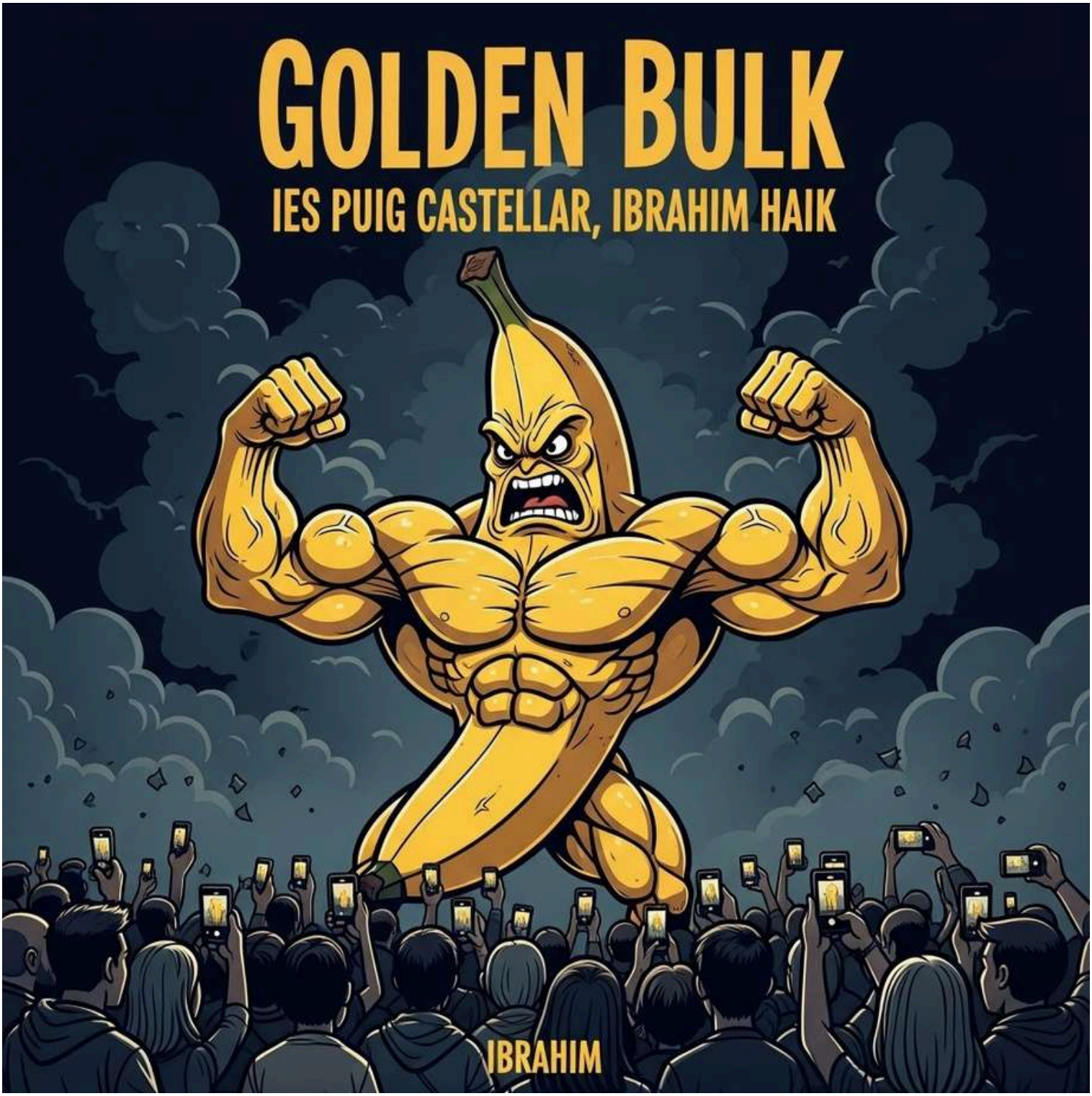

GOLDEN BULK
IES PUIG CASTELLAR, IBRAHIM HAIK
IBRAHIM

Resumen del proyecto

Golden Bulk es una aplicación de fitness diseñada para crear una comunidad positiva y cohesionada entre usuarios de toda España. La aplicación tiene como objetivo unir a personas con intereses similares en el mundo del fitness, promoviendo la comunicación, la motivación y la mejora continua a través de retos semanales, interacción social y recompensas. El proyecto se divide en varios bloques funcionales que integran tecnologías como React Native, Node, PostgreSQL y Firebase.

Cada usuario, al registrarse, elige una comunidad basada en su ciudad. Esta selección lo integra automáticamente en un chat global con otros miembros de esa localidad, fomentando el contacto y el intercambio de información. Este chat incluye funcionalidades como auto-desplazamiento, notificaciones de usuarios en línea e indicadores de usuario escribiendo, todo gestionado mediante Firebase.

En la pantalla principal, los administradores pueden publicar vídeos e imágenes relacionados con entrenamientos, suplementos y estilos de vida saludables. Estas publicaciones se muestran en un feed dinámico, donde los usuarios pueden interactuar mediante “me gusta” y comentarios. Los archivos multimedia se gestionan mediante Cloudinary y las interacciones sociales se registran con PostgreSQL.

Los retos semanales son una pieza clave: los administradores crean retos específicos por comunidades, y los usuarios pueden grabarse completándolos mediante la cámara integrada de la app. Los vídeos se revisan desde el portal del administrador, y si son aprobados, se conceden puntos virtuales que pueden canjearse en la tienda de la aplicación.

El proyecto ha sido desarrollado en solitario, con una fuerte inversión de tiempo y aprendizaje tecnológico, buscando siempre un enfoque realista y escalable para su despliegue. La plataforma se ha concebido como un espacio social saludable, abierto a colaboraciones con marcas y entrenadores, y con una clara orientación hacia la comunidad y la motivación colectiva.

Palabras clave:

Fitness, Comunidad, React Native, Firebase, PostgreSQL, Cloudinary, Retos, Chat en tiempo real.

Abstract

Golden Bulk is a fitness application designed to create a positive and cohesive community among users across Spain. The app aims to bring together people with similar interests in the fitness world, promoting communication, motivation, and continuous improvement through weekly challenges, social interaction, and rewards. The project is divided into several functional blocks that integrate technologies such as React Native, Firebase, PostgreSQL, and Node.

Upon registration, each user selects a community based on their city. This selection automatically integrates them into a global chat with other members from that area, encouraging contact and the exchange of information. This chat includes features such as auto-scrolling, online user notifications, and typing indicators, all managed through Firebase.

On the main screen, administrators can post videos and images related to workouts, supplements, and healthy lifestyles. These posts appear in a dynamic feed, where users can interact through “likes” and comments. Multimedia files are managed using Cloudinary, and social interactions are recorded with PostgreSQL.

Weekly challenges are a key feature: administrators create specific challenges for each community, and users can record themselves completing them using the app's built-in camera. Videos are reviewed through the admin portal, and if approved, users are awarded virtual points that can be redeemed in the app’s store.

The project has been developed solo, with a strong investment in time and technological learning, always aiming for a realistic and scalable deployment. The platform has been conceived as a healthy social space, open to collaborations with brands and trainers, and clearly focused on community and collective motivation.

Keywords

Fitness, Community, React Native, Firebase, PostgreSQL, Cloudinary, Challenges, Real-time Chat

1. Introducción

En la actualidad, el crecimiento del uso de dispositivos móviles y aplicaciones orientadas a la salud y el bienestar ha generado una nueva tendencia en la manera en que las personas gestionan su entrenamiento físico y su relación con la comunidad. A pesar de este crecimiento, muchas aplicaciones se limitan a ofrecer rutinas o dietas sin establecer vínculos sociales sólidos ni fomentar la motivación colectiva. Esta carencia se vuelve especialmente significativa en entornos en los que la constancia y la motivación son esenciales para alcanzar objetivos personales relacionados con el fitness.

Para responder a esta necesidad, se propone el desarrollo de Golden Bulk, una aplicación móvil de fitness basada en la comunidad que promueve la interacción entre usuarios mediante chats locales, retos semanales y recompensas por logros. El objetivo es ofrecer una plataforma donde los usuarios no solo puedan seguir rutinas de entrenamiento, sino también sentirse acompañados, compartir experiencias y competir sanamente con otros miembros de su ciudad.
La aplicación se desarrolla con React Native para multiplataforma (Android/iOS), utilizando Firebase para la autenticación y el sistema de mensajería en tiempo real. Además, se hace uso de PostgreSQL para gestionar la persistencia de datos y Cloudinary para el almacenamiento y gestión de contenido multimedia. La aplicación integra funcionalidades como el registro y acceso del usuario, feed de publicaciones, mensajería por comunidades locales, retos audiovisuales semanales, tienda de recompensas y un portal de administración.

A diferencia de muchas aplicaciones del sector, Golden Bulk apuesta por una experiencia comunitaria completa, donde cada usuario tiene un papel activo dentro de su entorno local, participando e interactuando constantemente. Esta combinación de comunidad, gamificación y tecnología hace que el proyecto no solo sea una herramienta de seguimiento de entrenamientos, sino una red social motivadora centrada en el fitness.

1.1. Contexto

Durante el desarrollo del proyecto, uno de los miembros del equipo vivió una experiencia significativa relacionada con la motivación y el seguimiento del entrenamiento físico en entornos comunitarios. En la ciudad donde reside, observó que muchos usuarios de aplicaciones deportivas convencionales no lograban mantener una constancia adecuada ni un sentimiento de pertenencia a un grupo, lo que afectaba negativamente sus resultados y compromiso.

Esta situación motivó al equipo a profundizar en el estudio de plataformas móviles, sistemas de gamificación y redes sociales locales, con el fin de diseñar una solución que unificara estos elementos para potenciar la motivación y el soporte entre usuarios. La experiencia acumulada en el análisis de estas tecnologías y comportamientos sociales sentó las bases para la creación de ***Golden Bulk***, una aplicación enfocada en la interacción comunitaria y el estímulo de la constancia mediante retos, recompensas y comunicación directa.
Este contexto ha orientado el desarrollo del proyecto hacia una solución innovadora que combina funcionalidades tecnológicas avanzadas con un enfoque centrado en la experiencia humana y social del usuario.

1.2. Justificación

El desarrollo de ***Golden Bulk*** se justifica por diversas razones. En primer lugar, permite al equipo aplicar y profundizar en conocimientos tecnológicos actuales en áreas como desarrollo móvil multiplataforma, bases de datos en la nube, autenticación segura y sistemas de mensajería en tiempo real.

En segundo lugar, se plantea como una oportunidad para crear un producto útil y diferenciador que atienda una necesidad real de la sociedad: la motivación y el acompañamiento en el fitness a través de una comunidad local.

Además, el proyecto fomenta el trabajo en equipo y la integración de diversas competencias técnicas y creativas, promoviendo el aprendizaje colaborativo y la resolución de problemas complejos.

Finalmente, ***Golden Bulk*** representa un reto profesional que permite explorar nuevas tendencias tecnológicas y de diseño centrado en el usuario, abriendo posibilidades para futuras mejoras y aplicaciones en ámbitos similares.

1.3. Objetivos

Para la realización del proyecto se han definido varios objetivos que guiarán su desarrollo, divididos en dos categorías principales:

1.3.1. Objetivos Generales

- Implementar metodologías ágiles que permitan una gestión eficiente del desarrollo, facilitando la planificación, seguimiento y entrega de resultados.
- Desarrollar una aplicación móvil multiplataforma que facilite la interacción entre usuarios para el seguimiento de entrenamientos, retos y comunicación en tiempo real, accesible desde cualquier dispositivo.
- Integrar funcionalidades que potencien la motivación y la colaboración comunitaria mediante sistemas de gamificación, mensajería y gestión de retos semanales.
- Garantizar una arquitectura modular y escalable que permita incorporar futuras mejoras y mantener un rendimiento óptimo.

1.4. Estrategia y Planificación

La estrategia de desarrollo se basa en la construcción desde cero de una aplicación móvil utilizando tecnologías modernas, combinando herramientas y frameworks conocidos, así como la adopción de nuevas tecnologías específicas para mensajería en tiempo real, almacenamiento en la nube y gestión de usuarios.

Se seguirá un enfoque de aprendizaje autodidacta para dominar estas tecnologías, permitiendo así obtener un producto final con un nivel profesional alto, adaptable y robusto. El proyecto se organizará en fases claramente definidas, que abarcarán desde el análisis y diseño hasta la implementación y pruebas, siempre con revisiones periódicas para asegurar la calidad y cumplimiento de los objetivos.

1.5. Metodología de Trabajo

Al tratarse de un proyecto desarrollado de forma individual, la gestión del tiempo y la organización personal son aspectos clave para asegurar la finalización exitosa del trabajo sin retrasos ni estrés innecesario.

Para ello, se establece un plan de trabajo estructurado con objetivos claros y plazos autoimpuestos, utilizando herramientas simples como calendarios personales y listas de tareas para mantener el foco en las prioridades.

Se prioriza un enfoque incremental e iterativo, realizando entregas parciales y revisiones continuas del progreso, lo que permite identificar problemas a tiempo y ajustar la dirección del proyecto de manera autónoma y eficiente.

1.5.1 Análisis de Tareas

El desarrollo del proyecto se ha organizado en tres grandes fases, adaptadas a la naturaleza individual del proyecto y su complejidad tecnológica.

- **Planificación y análisis inicial:** En esta fase se realizó un estudio profundo de las necesidades de usuarios en el sector fitness, detectando la carencia de aplicaciones que fomenten una comunidad real y cercana. Se definieron las funcionalidades clave: comunidades basadas en ciudades, chat en tiempo real, feed multimedia, retos con videos y sistema de recompensas. Debido al desarrollo individual, la planificación fue flexible y adaptativa, sin el uso de herramientas formales de gestión de tareas.

- **Estudio y selección tecnológica:** Esta etapa implicó un proceso de exploración y pruebas de distintas tecnologías para cubrir cada necesidad del proyecto. Se descartaron opciones como Socket.io por su complejidad y limitaciones para el chat, y Android Studio por no ajustarse a los objetivos profesionales. Se optó por React Native con Expo para la app móvil ya que ofrecía una vista en tiempo real, Firebase para el chat en tiempo real, Cloudinary para el almacenamiento multimedia, y PostgreSQL para la gestión de datos estructurados como usuarios, retos, puntos y tienda. Esta fase fue clave para entender la integración entre sistemas y garantizar la escalabilidad futura.

- **Desarrollo e integración:** En esta fase se implementaron las distintas funcionalidades de la app siguiendo la arquitectura definida. Se comenzó por backend (aproximadamente 3 meses de dedicación intensiva) para crear las API y gestionar la lógica de negocio, seguido por el desarrollo del frontend en React Native. La integración entre Firebase, Cloudinary y PostgreSQL se gestionó cuidadosamente para asegurar coherencia en los datos y una experiencia fluida.

Listado de tareas clave:

1. Definir y modelar las entidades principales: usuarios, comunidades, mensajes, retos, puntos, tienda, etc.

2. Desarrollar la API backend en Node.js con Express para gestionar datos y lógica de negocio.

3. Implementar el chat en tiempo real usando Firebase, con funcionalidades de usuarios en línea, auto-scroll y notificaciones de escritura.

4. Configurar Cloudinary para el almacenamiento y recuperación eficiente de imágenes y vídeos.

5. Programar el feed de contenidos administrados por los administradores y habilitar interacciones (likes, comentarios).

6. Crear el sistema de retos con carga de videos, aprobación y asignación de puntos.

7. Diseñar y programar la tienda virtual con sistema de puntos para canje.

8. Desarrollar la gestión del perfil de usuario con actualización de datos y consulta de pedidos.

9. Realizar pruebas funcionales y de integración usando Postman y herramientas propias.

10. Evaluar mejoras futuras como integración de IA y escalabilidad del sistema.

1.6 Estudio económico y presupuesto

Determinar los costes estimados para desarrollar y mantener la aplicación, teniendo en cuenta la infraestructura tecnológica, el desarrollo frontend y backend, servicios externos y los recursos humanos necesarios.

Categoría	Coste Inicial	Coste Mensual	Justificación
Desarrollo de la aplicación	18.000 - 25.000 **€**	N/A	El coste inicial se calcula en función del salario, el número de trabajadores y la tarifa por hora, estimando un desarrollo de aproximadamente 6 meses. Se tiene en cuenta el salario y precio por hora del desarrollador
Diseño UI/UX	3.000 - 5.000 **€**	N/A	Se destina una inversión inicial para contratar diseñadores especializados que creen una interfaz intuitiva y atractiva, mejorando la experiencia de usuario y la coherencia visual de la aplicación.
Servidores y Hosting	N/A	200- 500 **€**	No se requiere un coste inicial significativo; sin embargo, se establecerá una membresía mensual para mantener el servidor, cuyo valor variará según el software y la infraestructura elegida para asegurar la disponibilidad y escalabilidad del sistema.
Base de Datos	N/A	50 - 200 **€**	Dado el manejo de numerosos archivos e información,

			incluidos vídeos de gran tamaño, se contratará un servicio de base de datos con amplia capacidad de almacenamiento y rendimiento óptimo, garantizando la seguridad y eficiencia en el procesamiento de los datos.
Sistema de Mensajería	N/A	100 - 300 **€**	Se integrará un software especializado que ofrezca un sistema de mensajería robusto y de alta calidad, capaz de soportar múltiples conversaciones simultáneas en diversas comunidades, asegurando una comunicación fluida y sin latencia.
Mantenimiento E-commerce	N/A	50 - 300 **€**	El mantenimiento se centrará en la gestión y actualización de plugins de pago y otros sistemas críticos, garantizando el correcto funcionamiento y la seguridad de la tienda online, de acuerdo a las necesidades que se presenten.
Marketing y Publicidad	3.000 **€**	500 - 2,000 **€**	Se realizará una inversión inicial importante para atraer clientes potenciales mediante campañas estratégicas, seguida de campañas mensuales continuas que

			aseguren el crecimiento y la fidelización de la base de usuarios.
Creación Branding Suplementos	2.000 - 5.000 €	2,000 €	La inversión comprende la contratación de diseñadores para desarrollar la identidad visual de la marca, la selección de proveedores especializados y la producción del producto final.
AppStore y PlayStore	124 €	8.25 €	Se incluyen los costes de publicación en ambas plataformas, con un coste anual de 99 € para la App Store y un coste único de 24 € para la Play Store.
Sistema de Gamificación	N/A	500 - 3,000 €	El presupuesto se destina al desarrollo de funcionalidades que permitan premiar a los usuarios por canjear sus puntos, integrando retos, logros y recompensas.
Soporte y Mantenimiento	N/A	1,000 - 2,000 €	Se asigna un presupuesto para cubrir el salario de un desarrollador encargado de solucionar incidencias, corregir bugs y mantener la operatividad continua de la aplicación, asegurando su mejora y actualización constante.

2.1. Análisis de requisitos

2.1.1 Requisitos funcionales

Los requisitos funcionales detallan las funciones esenciales que debe realizar la aplicación para cumplir con sus objetivos.

Registro de usuario	El sistema debe permitir registrar nuevos usuarios, asociados a una comunidad específica.
Inicio de sesión	El sistema debe permitir iniciar sesión con email y contraseña, devolviendo un token JWT.
Creación de retos	Los administradores deben poder crear retos semanales filtrados por comunidad
Participación en retos	Los usuarios deben poder grabarse completando un reto desde la app y enviar el vídeo
Validación de retos	Los administradores deben poder aprobar o suspender vídeos subidos por los usuarios
Asignación de puntos	El sistema debe asignar puntos a los usuarios cuando se aprueba un reto
Visualización de puntos	Los usuarios deben poder consultar su balance total de puntos acumulados
Compra en tienda	Los usuarios deben poder canjear puntos por productos disponibles en la tienda virtual
Chat en tiempo real	Los usuarios deben poder enviar y recibir mensajes instantáneos en un chat por comunidad
Presencia online	El sistema debe indicar qué usuarios están conectados al chat en tiempo real
Feed de publicaciones	Los administradores deben poder publicar contenido multimedia visible por los usuarios
Interacción social	Los usuarios deben poder dar “likes” y comentar publicaciones del feed
Gestión de perfil	Los usuarios deben poder consultar su perfil y actualizar algunos datos
Portal administrador	El administrador debe acceder a un panel con funcionalidades especiales (estadísticas, gestión de retos, vídeos y publicaciones)

2.1.2 Requisitos no funcionales

Seguridad de contraseñas	Las contraseñas deben almacenarse cifradas mediante hashing seguro con bcrypt
Autenticación segura	Toda ruta protegida debe validar el token JWT y restringir el acceso a usuarios no autenticados
Persistencia de datos	El sistema debe almacenar la información clave de forma persistente en PostgreSQL
Respuesta rápida	Las operaciones más frecuentes (login, carga de feed, puntos) deben responder en menos de 2 segundos
Multiplataforma	La aplicación debe funcionar correctamente en dispositivos Android e iOS
Disponibilidad	El sistema debe estar disponible un mínimo del 99% del tiempo
Subida eficiente de multimedia	Las imágenes y vídeos deben subirse y servirse mediante una CDN como Cloudinary
Experiencia de usuario	La app debe ser intuitiva, visualmente coherente y fácil de navegar
Actualización en tiempo real	El sistema de chat debe reflejar los mensajes y presencia online sin necesidad de recargar

2.2. Tecnologías

Para cubrir los diversos requisitos de **Golden Bulk** se seleccionaron tecnologías y herramientas específicas que ofrecen el equilibrio entre funcionalidad, eficiencia y aprendizaje profesional.

- **React Native con Expo:** Framework elegido para el desarrollo móvil multiplataforma. Permite un desarrollo ágil, reutilización de código y acceso a funcionalidades nativas, la razón principal de su elección se debe a que muestra una vista en tiempo real del estado de la app.

- **Node.js:** Motor backend utilizado para construir la API REST que gestiona usuarios, retos, puntos y tienda. Su ecosistema facilita la integración con bases de datos y servicios externos.

- **Express**: Framework minimalista para Node.js utilizado para estructurar y simplificar la creación de rutas y middleware en el servidor. Facilita el manejo de peticiones HTTP, la modularidad del código y la escalabilidad del backend

- **PostgreSQL:** Base de datos relacional seleccionada para almacenar datos estructurados que requieren relaciones complejas, como usuarios, retos, puntuaciones y registros de compras.

- **Firebase:** Servicio en la nube para el chat en tiempo real. Proporciona sincronización instantánea, usuarios en línea, notificaciones y almacenamiento de mensajes sin necesidad de construir un backend específico para esta funcionalidad, reduciendo la complejidad y coste inicial.

- **Cloudinary:** Plataforma de almacenamiento y gestión de imágenes y videos. Utilizada para alojar el contenido multimedia del feed y los videos de retos, facilitando la carga, optimización y entrega eficiente, aunque con ciertas limitaciones de velocidad.

- **Ngrok:** Herramienta para exponer localmente el servidor Node.js a Internet durante desarrollo y pruebas, permitiendo que la app móvil pueda comunicarse con el backend desde cualquier lugar.

- **Postman:** Software para testeo y validación de los endpoints de la API, garantizando que las comunicaciones entre frontend y backend funcionen correctamente antes y durante el desarrollo.

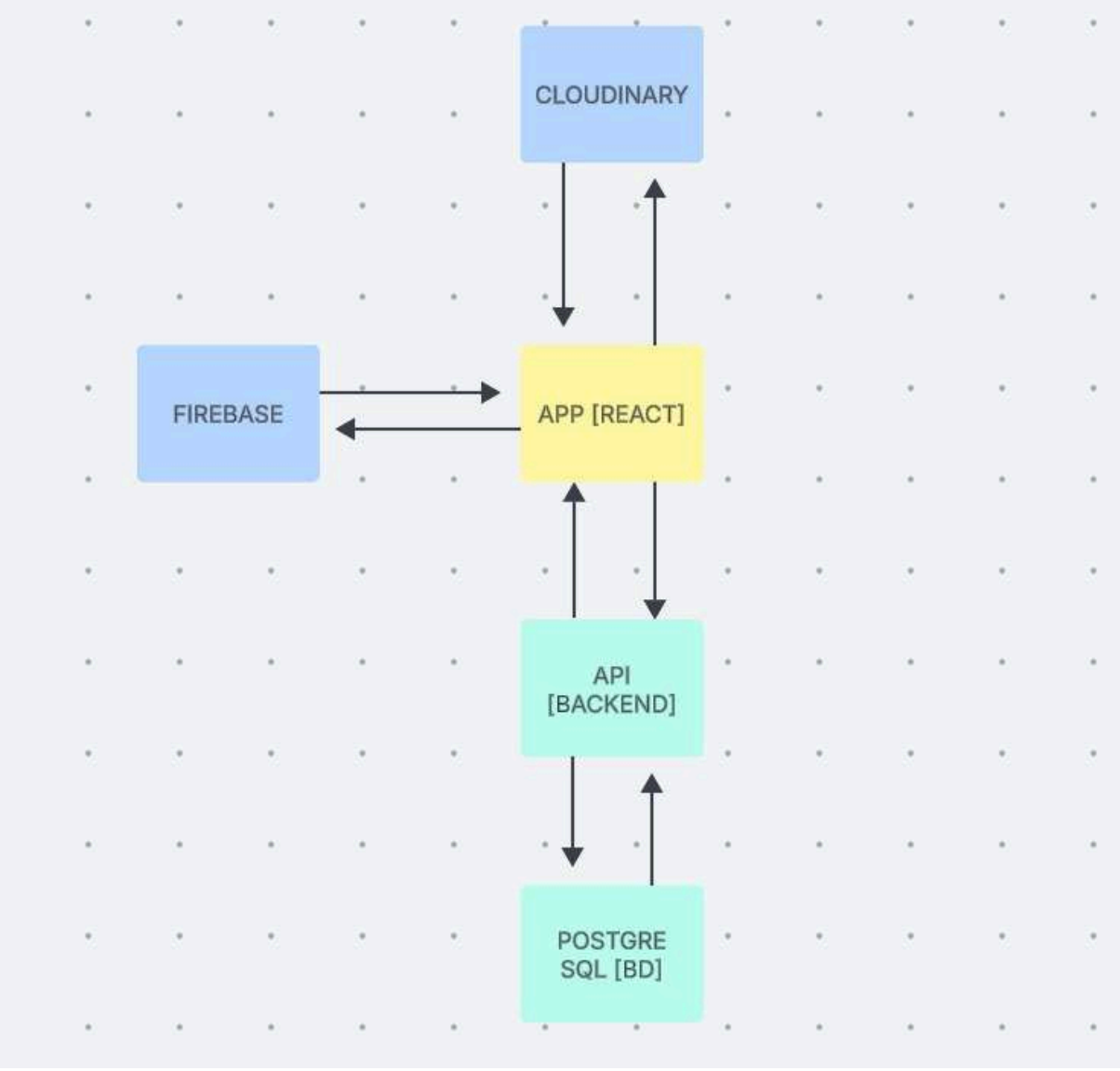
Tecnologías examinadas descartadas:

Socket.io , React , React Native (Base) , Android Studio, Supabase, AppWrite

2.3 Estructura del proyecto

La arquitectura general de la aplicación se basa en una estructura modular compuesta por diversos componentes interconectados que permiten el funcionamiento fluido y escalable del sistema. El sistema combina una aplicación móvil desarrollada en React Native (Expo), un servidor backend construido en Node.js con Express, una base de datos relacional PostgreSQL, Firebase Realtime Database para la mensajería en tiempo real y Cloudinary como proveedor de almacenamiento multimedia.

A continuación, se presenta un esquema conceptual de la estructura:



- La app interactúa directamente con el backend el cual insertar y devuelve información de la base de datos (Login, registro, obtener feed, obtener retos,
- obtener productos, obtener información del usuario, editar perfil, obtener cantidad de puntos, realizar compras..)

- Firebase se comunica con la app para el chat en tiempo real mediante listeners y notificaciones en vivo.
- Cloudinary recibe los ficheros multimedia desde la app y devuelve URLs que se almacenan en la base de datos.

2.4 Descripción de los componentes

2.4.1 Aplicación (Frontend)

Descripción:
La interfaz de nuestra aplicación móvil ha sido desarrollada utilizando React Native con gestión mediante Expo, ofreciendo una experiencia fluida y multiplataforma (Android/iOS). El sistema de navegación se basa en una combinación de navegación por pestañas (TabNavigator) y navegación por pila (StackNavigator), para permitir una jerarquía clara entre pantallas y una navegación intuitiva.
Se ha implementado **AsyncStorage** para gestionar la persistencia del token JWT de sesión de manera segura en el almacenamiento local. La comunicación con el backend se realiza mediante **Axios**, y se utiliza **Firebase** para funcionalidades en tiempo real como el chat y la presencia de usuarios.
Además, la aplicación incorpora **Cloudinary** para la gestión y subida de imágenes y vídeos. También cuenta con un estilo visual oscuro modernizado, con elementos de diseño como **LinearGradient**, iconos de **Ionicons** y componentes de diseño responsivo que mejoran la experiencia visual.
Tecnologías:

React Native, Expo, Axios, AsyncStorage, React Navigation, Firebase SDK, Cloudinary, React Native Paper, Expo Camera / Media Library, jwt-decode

Funcionalidades destacadas:

Usuario:

1. Registro e inicio de sesión con formularios validados y gestión de errores.
2. Visualización de retos activos y participación mediante subida de vídeos.
3. Sistema de puntuación por logros y visualización de puntos.
4. Muro de publicaciones con likes, comentarios e interacción social.
5. Acceso a una tienda virtual con compra de productos mediante puntos.
6. Sistema de chat en tiempo real integrado con Firebase Realtime Database.
7. Menú lateral con acceso al perfil, puntos y opción de cerrar sesión.

Administrador:

1. Inicio de sesión con rol de administrador.
2. Creación y actualización de retos con selección de comunidad y fechas.
3. Revisión y aprobación/suspensión de vídeos subidos por los usuarios.
4. Creación de publicaciones y productos para la tienda virtual.
5. Panel de estadísticas: número de usuarios, mensajes y retos.
6. Gestión de chats comunitarios con visualización de usuarios en línea.
7. Interfaz adaptada para gestiones administrativas, con acceso directo a funcionalidades desde la pantalla principal.

Arquitectura de componentes:

- Componentes como ***HomeHeader***, ***DrawerMenu***, ***TabNavigator*** y ***LoadingScreen*** mejoran la organización visual y la reutilización del código.
- Separación clara entre pantallas (screens/) y componentes reutilizables (`components/`).

Otras características:

- Navegación segura entre pantallas con React Navigation.
- Gestión de errores y pantallas de carga (***ErrorScreen***, ***LoadingScreen***).
- Soporte para subida de imágenes y vídeos con permisos nativos.
- Estilos personalizados y adaptados a un tema oscuro moderno.
- El proyecto incluye un archivo api.js donde se define la URL base de conexión con el backend, la cual debe actualizarse diariamente con el enlace generado por **ngrok** para garantizar el acceso remoto desde cualquier ubicación.

2.4.2 Backend (API REST)

Descripció:
Servidor desarrollado con Node.js y Express, que expone endpoints REST para autenticar usuarios, gestionar retos, puntos, compras, y manejar publicaciones.

Tecnologies: Node.js, Express, JWT, Bcrypt, PostgreSQL (pg), Multer

Usuarios, endpoint encargado de gestionar la información de los usuarios

GET /usuarios	**Listar todos los usuarios.**
GET /usuarios/:id	**Obtiene un usuario por id, necesario para la mayoría de funcionalidades de la aplicación.**
POST /usuarios	**Registra un nuevo usuario.**
DELETE /usuarios/:id	**Elimina un usuario.**
PUT /usuarios	**Actualiza la información de un usuario.**

Comunidades, endpoint encargado de gestionar las comunidades a las que los usuarios se registran

GET /comunidades	**Listar todas las comunidades, se utiliza en el registro para que el usuario elija su comunidad.**
GET /comunidades/id	**Se utiliza para filtrar información de interés dependiendo de la comunidad elegida por el usuario.**
POST /comunidades	**Crea una nueva comunidad.**
PUT /comunidades/id	**Actualiza una comunidad.**
DELETE /comunidades/:id	**Elimina una comunidad.**

Retos, endpoint encargado de gestionar los retos que se muestran a los usuarios en los cuales los usuarios se graban y suben sus vídeos para obtener una cantidad de puntos.

GET /retos	**Lista todos los retos, se utiliza en la pantalla de retos filtrando los retos que coincidan con comunidad_id del usuario.**
GET /retos/:id	**Obtener reto por ID.**
POST /retos	**Crear reto.**
PUT /retos/:id	**Actualizar reto.**
DELETE /retos/:id	**Eliminar reto.**

Video, endpoint encargado de gestionar los vídeos que se suben en el apartado de retos

GET /videos	**Lista todos los vídeo de retos, se utiliza en el portal del admin.**
POST /videos	**Sube un nuevo vídeo, se llama cuando se recibe la URL de Cloudinary.**
PUT /videos/:id	**Desde el portal del admin, cambia el estado del vídeo (aprobado, suspendido).**
DELETE /videos/:id	**Eliminar vídeo.**

Puntos, endpoint encargado de gestionar los puntos del usuario

GET /puntos	**Lista todos los registros de puntos**
GET /puntos/:usuario_id	**Lista el balance total de puntos de un usuario mediante consultas, se utiliza en el UseEffect de Home.js.**
POST /puntos	**Asignar puntos a un usuario (al completar un reto en aprobado) si se devuelve aprobado se llama a esta ruta.**

Productos, endpoint encargado de gestionar los productos de la tienda

GET /productos	Lista todos los productos disponibles
GET /productos/:id	Obtiene producto por id
POST /productos	Crea un producto nuevo
PUT /productos/:id	Actualiza el producto
DELETE /productos/:id	Elimina el producto

Posts, endpoint encargado de gestionar los posts que se suben al feed (vídeos, imágenes)

GET /posts	Lista todos los posts
GET /posts/:id	Obtener post por ID
POST /posts	Crear un nuevo post
DELETE /posts/:id	Eliminar post

Likes y comentarios, endpoints encargados de los likes y comentarios de un post

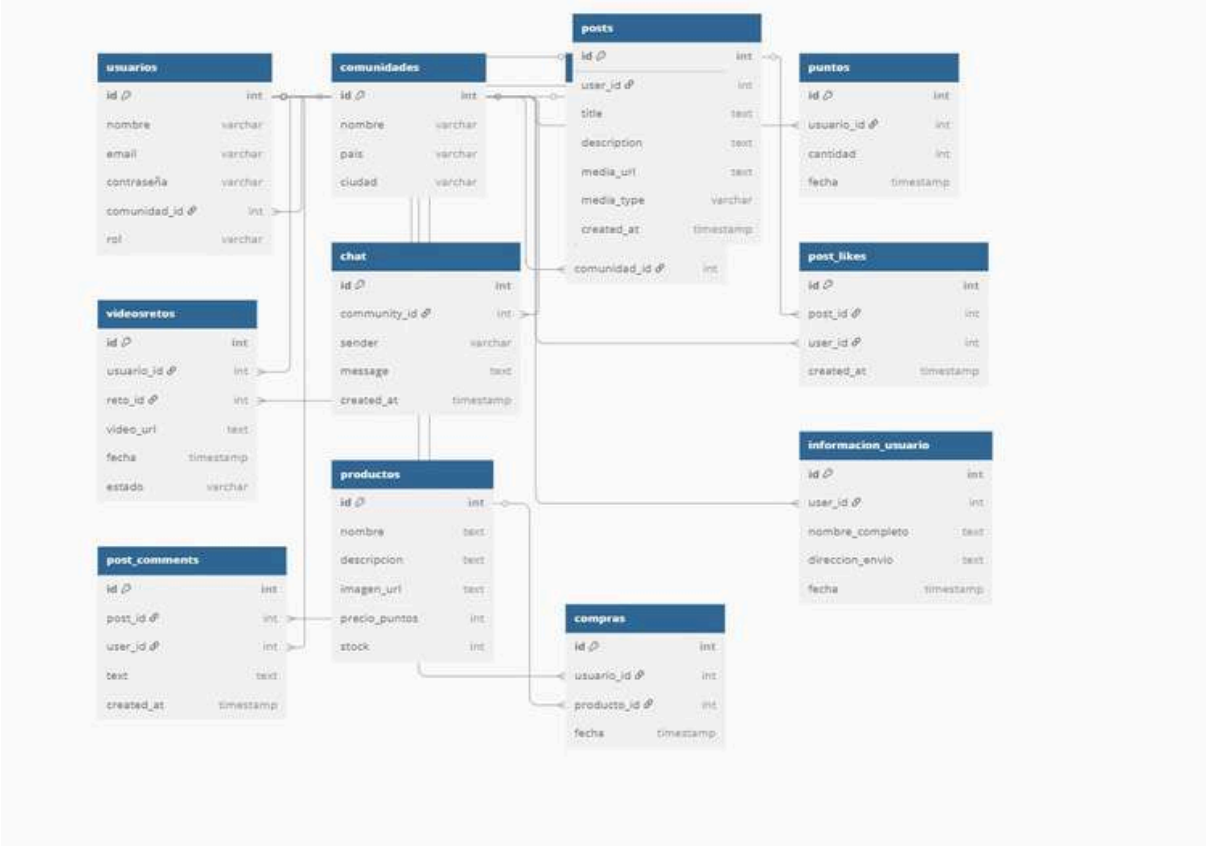
POST /likes	Dar like a un post
GET /likes/:postId	Obtener número de likes
DELETE /likes	Quitar un like (por usuario y post)
GET /comments/:postId	Obtener comentarios de un post
POST /comments	Añadir comentarios a un post

2.4.3 Base de Datos (PostgreSQL)

Descripción:
Sistema relacional donde se almacena toda la información persistente: usuarios, vídeos, imágenes, retos, puntos, productos, compras, comunidades, publicaciones, likes y comentarios.

En las tablas de la base de datos no se guardan directamente los vídeos ni las imágenes en binario. En su lugar, se suben a Cloudinary, y lo que se almacena en la base de datos son solo las URLs de acceso a esos archivos. Esto hace que la app sea más rápida, ligera y fácil de mantener.

Tecnologies: PostgreSQL, pgAdmin.



2.4.4 Almacenamiento Multimédia (Cloudinary)

Descripció:
Servicio en la nube encargado de recibir, almacenar y servir imágenes y vídeos desde la app móvil, optimizando su tamaño y accesibilidad.

Tecnologías: Cloudinary SDK, CDN de Cloudinary.

Funcionalidades: Subida directa desde la app, generación de URLs seguras, transformación automática de los archivos.

Cloudinary

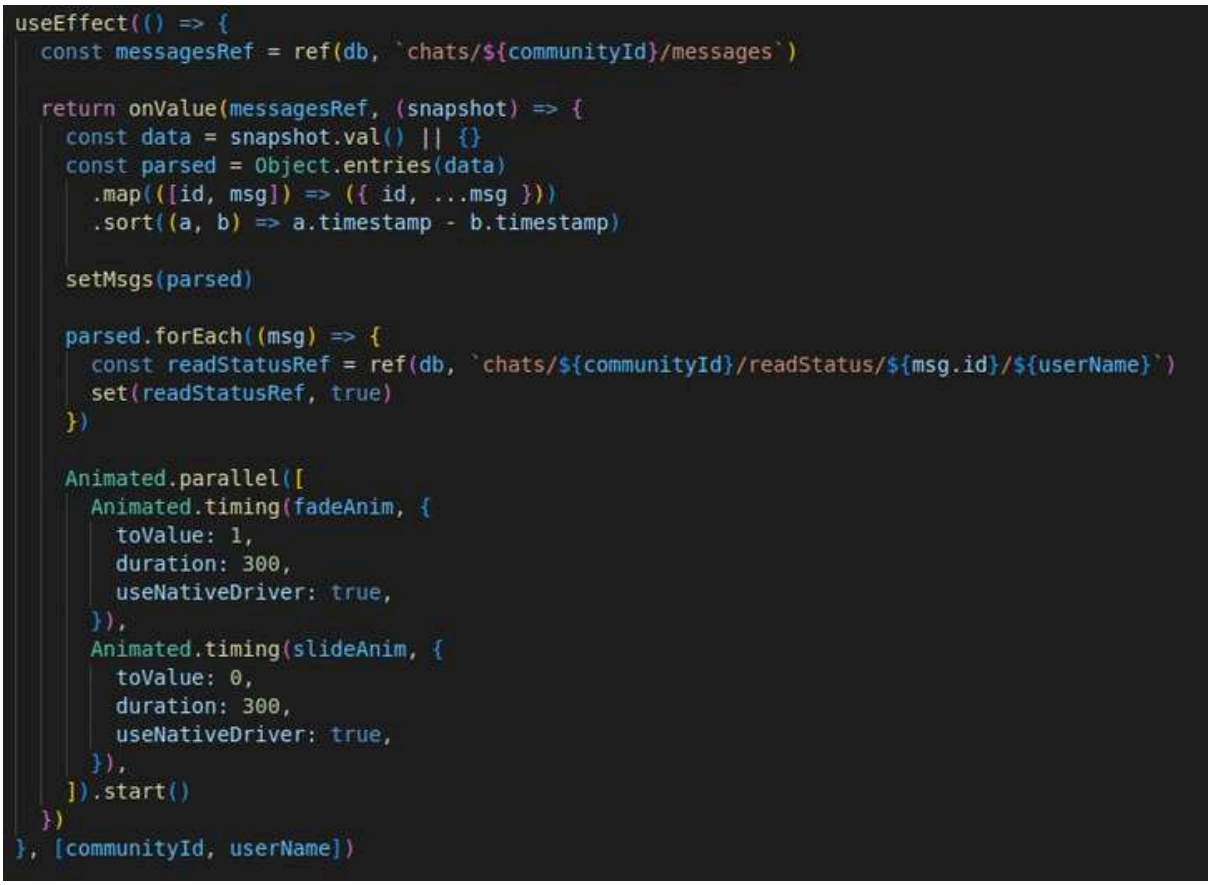
2.4.5 Chat en Tiempo Real (Firebase Realtime Database)

Descripción: Motor de mensajería en tiempo real usado para los chats de comunidad. Cada comunidad tiene un nodo de chat donde los usuarios intercambian mensajes.

Tecnologías: Firebase Realtime Database, Firebase Auth (si se usa), Firebase SDK.

Funcionalidades: Envío y recepción instantánea de mensajes, gestión del estado de presencia (usuarios online/offline), escucha de nuevos mensajes mediante listeners.

Cuando el componente de chat se monta, se conecta a la base de datos para leer los mensajes en tiempo real de una comunidad específica



```
useEffect(() => {
  const messagesRef = ref(db, `chats/${communityId}/messages`)

  return onValue(messagesRef, (snapshot) => {
    const data = snapshot.val() || {}
    const parsed = Object.entries(data)
      .map(([id, msg]) => ({ id, ...msg }))
      .sort((a, b) => a.timestamp - b.timestamp)

    setMsgs(parsed)

    parsed.forEach((msg) => {
      const readStatusRef = ref(db, `chats/${communityId}/readStatus/${msg.id}/${userName}`)
      set(readStatusRef, true)
    })

    Animated.parallel([
      Animated.timing(fadeAnim, {
        toValue: 1,
        duration: 300,
        useNativeDriver: true,
      }),
      Animated.timing(slideAnim, {
        toValue: 0,
        duration: 300,
        useNativeDriver: true,
      }),
    ]).start()
  })
}, [communityId, userName])
```

Se usa Firebase para mantener un estado online/offline, (Marca al usuario como online, usa onDisconnect para marcarlo como offline automáticamente, escucha los cambios en presencia de otros usuarios)

```
useEffect(() => {
  const userStatusRef = ref(db, `communities/${communityId}/presence/${userName}`)

  set(userStatusRef, {
    online: true,
    lastSeen: serverTimestamp(),
  })

  onDisconnect(userStatusRef).update({
    online: false,
    lastSeen: serverTimestamp(),
  })

  const presenceRef = ref(db, `communities/${communityId}/presence`)
  return onValue(presenceRef, (snapshot) => {
    const data = snapshot.val() || {}
    const online = Object.entries(data)
      .filter(([_, status]) => status.online)
      .map(([username]) => username)

    setOnlineUsers(online)
  })
}, [communityId, userName])
```

Cuando el usuario presiona “Enviar”, se guarda el mensaje en FireBase, utiliza push para agregar un nuevo nodo a la lista de mensajes, guarda los campos necesarios y marca el mensaje como leído por el emisor

```
const send = () => {
  if (!text.trim()) return

  setIsTyping(true)

  const messagesRef = ref(db, `chats/${communityId}/messages`)
  const newMessageRef = push(messagesRef)

  set(newMessageRef, {
    text,
    userName,
    timestamp: serverTimestamp(),
  }).then(() => {
    const readStatusRef = ref(db, `chats/${communityId}/readStatus/${newMessageRef.key}/${userName}`)
    set(readStatusRef, true)
  })

  setText("")
  setIsTyping(false)
}
```

https://fitpluschat-default-rtdb.europe-west1.firebasedatabase.app

https://fitpluschat-default-rtdb.europe-west1.firebasedatabase.app/

- chats
- communities
- presence

2.4.6 Gestor de Reto i Puntuación

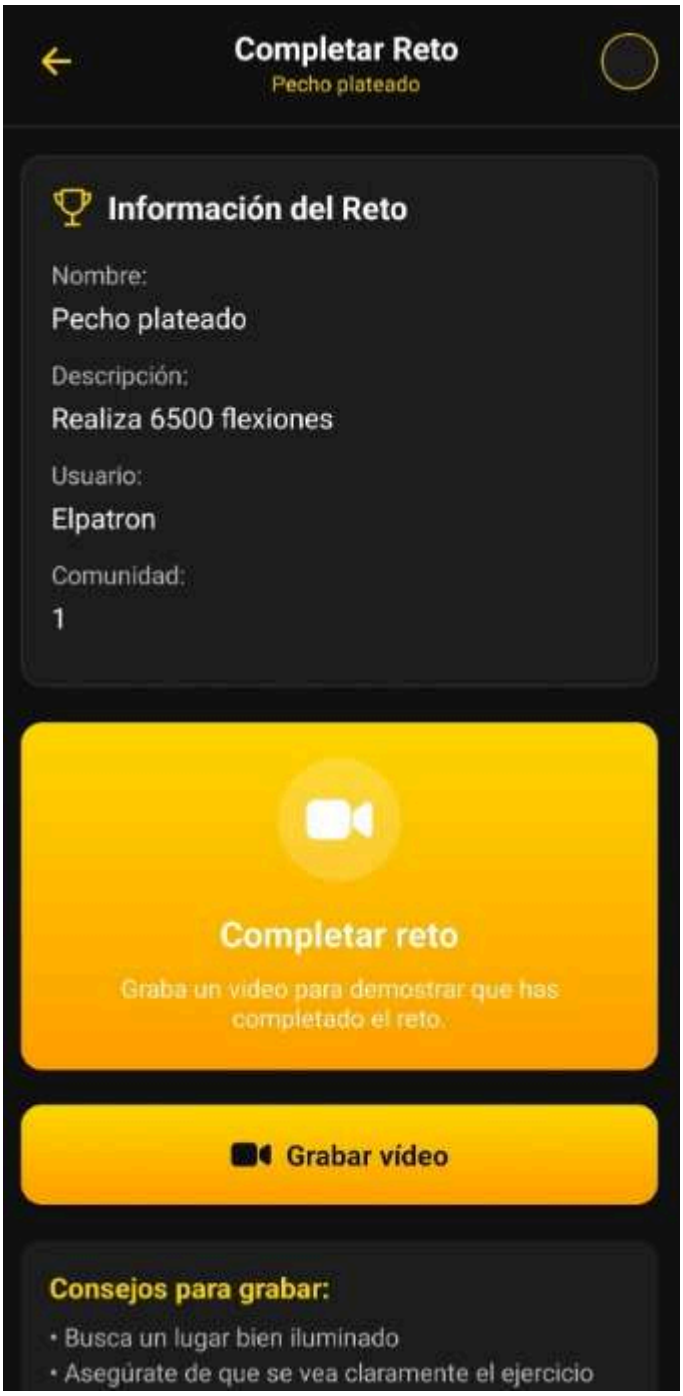
Descripción: Módulo interno del backend encargado de validar las participaciones de los usuarios en los retos, gestionar el flujo de aprobación por parte del administrador y, en caso afirmativo, asignar puntos al perfil del usuario. Este sistema es esencial para garantizar la equidad y la correcta valoración de los retos dentro de la comunidad.
Tecnologías:

Backend Node.js, PostgreSQL.

Funcionalidades: Registro automático de la participación una vez el usuario sube el vídeo del reto, visualización de los retos pendientes de aprobación por parte del administrador, posibilidad de aprobar o suspender cada vídeo, asignación automática de puntos al usuario si el vídeo es aprobado, actualización de la base de datos con el estado y la fecha de la participación.

Primero el administrador crea un reto desde el panel de administración y selecciona la comunidad a la que va dirigida el reto:

El usuario consulta su sección de retos y revisa que hay un reto disponible para su comunidad, se le ofrece la opción de grabar un vídeo, el usuario no puede subir vídeos desde la galería, se le abre la cámara de la aplicación con Expo Camera y debe completar el reto en ese momento.



Cuando el usuario sube el vídeo se prepara el body y se hace un post a Cloudinary, almacenando el vídeo ahí, luego se obtiene la url que devuelve Cloudinary y se realiza un post la tabla videos, con los campos necesarios.

```
try {
  const formData = new FormData();
  formData.append('file', { uri, name: filename, type: 'video/mp4' });
  formData.append('upload_preset', CLOUDINARY_UPLOAD_PRESET);
  formData.append('public_id', `videos/${comunidadId}/${filename}`);

  setUploading(true);

  const response = await fetch(
    `https://api.cloudinary.com/v1_1/${CLOUDINARY_CLOUD_NAME}/video/upload`,
    { method: 'POST', body: formData }
  );

  const data = await response.json();
  if (!data.secure_url) {
    console.error('Error al subir a Cloudinary:', data);
    Alert.alert('Error', 'Error al subir el vídeo a Cloudinary');
    setUploading(false);
    return;
  }

  const videoData = {
    usuario_id: usuarioId,
    reto_id: retoId,
    video_url: data.secure_url,
    fecha: new Date().toISOString(),
  };

  await axios.post('/videos', videoData);
```

Al entrar en la pantalla se obtiene el valor de la columna estado del reto actual y del usuario actual, ahora mismo el estado es igual a pendiente por lo que al usuario le aparecerá el vídeo en revisión.

```
useEffect(() => {
  const fetchEstado = async () => {
    try {
      const response = await axios.get(`/videos/estado/${retoId}/${usuarioId}`);
      setEstado(response.data.estado);
    } catch (error) {
      console.error('Error al obtener estado:', error);
    } finally {
      setLoading(false);
    }
  };
  fetchEstado();
}, [retoId, usuarioId]);
```

El administrador entra a la sección de “Revisar vídeos”, se obtienen todos los vídeos subidos, el administrador reproduce el vídeo y decide si aprobar o suspender el vídeo, en este caso el administrador aprobará el vídeo y el valor de la columna estado asociado al reto y al usuario cambiará a “aprobado”.

El usuario consulta el estado de su vídeo, se vuelve a ejecutar el UseEffect y se obtiene el valor de estado, el usuario ve que ha completado el reto. Además en su balance se ha sumado la cantidad de puntos que daba el reto.

```
useEffect(() => {
  const fetchEstado = async () => {
    try {
      const response = await axios.get(`/videos/estado/${retoId}/${usuarioId}`);
      setEstado(response.data.estado);
    } catch (error) {
      console.error('Error al obtener estado:', error);
    } finally {
      setLoading(false);
    }
  };
  fetchEstado();
}, [retoId, usuarioId]);
```

Con createPunto creamos un nuevo registro de puntos asociados a un usuario ya sea en negativo o en positivo.

```
exports.createPunto = async (req, res) => {
  const { usuario_id, cantidad } = req.body;
  try {
    const resultado = await client.query(
      `INSERT INTO puntos (usuario_id, cantidad, fecha)
       VALUES ($1, $2, NOW()) RETURNING *`,
      [usuario_id, cantidad]
    );
    res.status(201).json(resultado.rows[0]);
  } catch (err) {
    console.error('Error al asignar puntos:', err);
    res.status(500).send('Error al asignar puntos');
  }
};
```

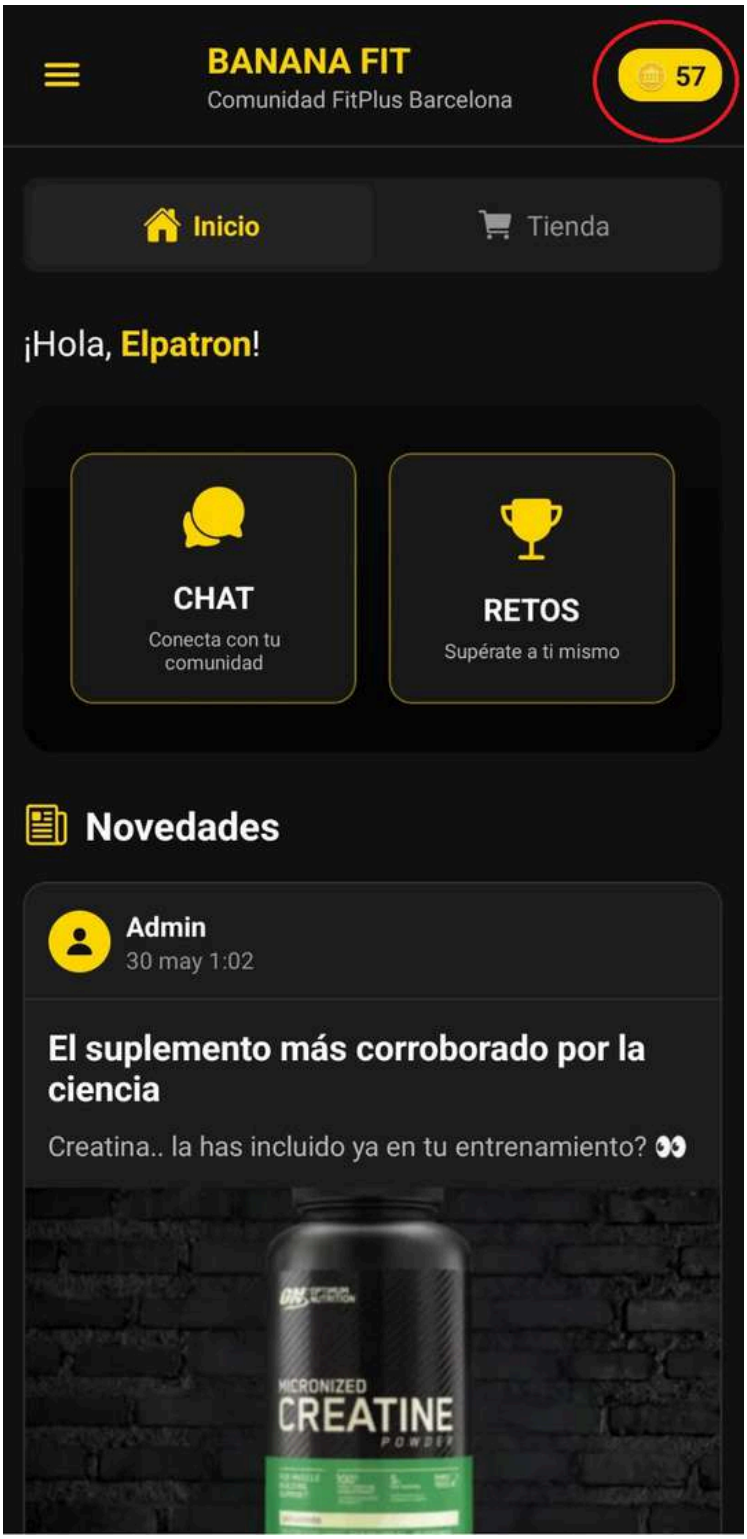
El nuevo registro aparece de esta forma en la tabla puntos.

```
id | usuario_id | cantidad |           fecha
---+------------+----------+----------------------------
 6 |         47 |      100 | 2025-05-09 16:41:16.456995
 7 |         47 |      250 | 2025-05-09 16:41:22.945055
 8 |         47 |      -50 | 2025-05-09 16:41:26.925862
```

Con getBalance obtenemos el balance total sumando todos los valores de cantidad del usuario en cuestión.

```
exports.getBalance = async (req, res) => {
  const { usuario_id } = req.params;
  try {
    const { rows } = await client.query(
      `SELECT COALESCE(SUM(cantidad), 0) AS total_puntos
       FROM puntos
       WHERE usuario_id = $1`,
      [usuario_id]
    );
    res.json({ usuario_id, total_puntos: rows[0].total_puntos });
  } catch (err) {
    console.error('Error al obtener balance de puntos:', err);
    res.status(500).send('Error al obtener balance de puntos');
  }
};
```

Finalmente el usuario obtiene el balance total de sus puntos



2.4.7 Sistema de Publicaciones

Descripción: Sistema que permite a los administradores crear y compartir publicaciones con contenido multimedia. Los usuarios pueden interactuar con estas publicaciones. Fomenta la interacción social mediante likes y comentarios. El administrador puede crear contenido destacado visible para todos los miembros.
Tecnologías:

Backend Node.js, PostgreSQL, Cloudinary.

Funcionalidades: Crear, listar y eliminar publicaciones (posts) con imágenes o vídeos (solo para administradores), añadir likes y comentarios a cada publicación, asociar cada post a un usuario y una comunidad, cargador de imágenes y vídeos integrado con Cloudinary, acceso rápido a los datos del autor, fecha y comunidad de cada post, eliminación segura con control de permisos según el autor o administrador.

Con getAllPosts obtenemos todos los posts, se llama a este endpoint en Home.js

```
exports.getAllPosts = async (req, res) => {
  try {
    const result = await client.query('SELECT * FROM posts ORDER BY created_at DESC');
    res.json(result.rows);
  } catch (err) {
    console.error('Error al obtener posts:', err);
    res.status(500).send('Error al obtener posts');
  }
};
```

Con createPost se crea un nuevo post, se llama desde CreatePost.js (solo administradores)

```
exports.createPost = async (req, res) => {
  const { user_id, title, description, media_url, media_type } = req.body;
  try {
    const result = await client.query(
      'INSERT INTO posts (user_id, title, description, media_url, media_type) VALUES ($1, $2, $3, $4, $5) RETURNING *',
      [user_id, title, description, media_url, media_type]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error('Error al crear el post:', err);
    res.status(500).send('Error al crear el post');
  }
};
```

2.4.8 Sistema de Tienda Virtual

Descripción:
Módulo destinado a la gestión y compra de productos mediante el sistema de puntos acumulados por el usuario. Los productos pueden ser creados y administrados por el administrador, y los usuarios pueden adquirirlos desde el frontend de manera intuitiva.
Tecnologías:

Backend Node.js, PostgreSQL, React Native, Axios. **Funcionalidades:** Visualización de productos disponibles con imágenes, descripción y precio en puntos, compra de productos desde la aplicación móvil, con deducción automática de los puntos del usuario, validación de saldo suficiente antes de permitir la compra, creación y edición de productos por parte del administrador (con subida de imagen y descripción), página de "Pasarela de compra" para confirmar y validar la adquisición, listado de compras realizadas por cada usuario (a futuro, puede ampliarse con historial o estados de pedidos), integración con Cloudinary para las imágenes de los productos.

En HomeScreen.js, cuando navegamos por tab al componente TiendaContent.js nos aparecen los productos disponibles, cuando intentamos realizar una compra se realiza un post a /compras que llama a comprarProducto, en este controlador primero se verifica que el producto exista y tenga stock, si el producto existe se obtienen los puntos del usuario de la mismo forma que en puntos.GetBalance, si el total de puntos es inferior al precio del producto impide la compra lanzando un error, en caso contrario se registra la compra en la tabla compra de manera parecida a como ocurre con la tabla puntos, luego se hace un insert en negativo del precio del producto para descontar esa cantidad del balance total del usuario, posteriormente se reduce el stock de la tabla productos.

```
exports.comprarProducto = async (req, res) => {
    const { usuario_id, producto_id } = req.body;

    try {
      await client.query('BEGIN');

      const { rows: productoRows } = await client.query(
        `SELECT * FROM productos WHERE id = $1 FOR UPDATE`,
        [producto_id]
      );
      const producto = productoRows[0];

      if (!producto || producto.stock <= 0) {
        throw new Error('Producto no disponible');
      }
      const { rows: puntosRows } = await client.query(
        `SELECT COALESCE(SUM(cantidad), 0) AS total_puntos
        FROM puntos
        WHERE usuario_id = $1`,
       [usuario_id]
      );
      const totalPuntos = puntosRows[0].total_puntos;
```

```
if (totalPuntos < producto.precio_puntos) {
  throw new Error('No tienes suficientes puntos');
}

await client.query(
  `INSERT INTO compras (usuario_id, producto_id) VALUES ($1, $2)`,
  [usuario_id, producto_id]
);

await client.query(
  `INSERT INTO puntos (usuario_id, cantidad) VALUES ($1, $2)`,
  [usuario_id, -producto.precio_puntos]
);

await client.query(
  `UPDATE productos SET stock = stock - 1 WHERE id = $1`,
  [producto_id]
);
```

Tabla compras

id	usuario_id	producto_id	fecha
1	52	2	2025-05-12 16:51:30.824259
2	52	3	2025-05-12 16:52:47.203738
3	52	1	2025-05-12 16:52:51.229096
4	47	2	2025-05-13 15:46:16.652828

Aquest sistema transforma els punts obtinguts en una motivació tangible, augmentant la fidelització i la implicació dels usuaris.

2.4.9 Sistema de Autenticación

Descripción: El sistema de autenticación es el componente encargado de garantizar la seguridad de acceso a los recursos protegidos de la aplicación. Implementa un modelo basado en JWT (JSON Web Tokens), permitiendo la autenticación sin estado (stateless) tanto para peticiones HTTP como para conexiones en tiempo real.. Las contraseñas de los usuarios se protegen mediante hashing con bcrypt, garantizando que la información sensible nunca se almacene en texto plano.

Este sistema asegura que sólo los usuarios registrados y autenticados puedan acceder a funcionalidades clave como la participación en comunidades, retos, chat o recompensas
Tecnologías: Node.js + Express, PostgreSQL, bcrypt, jsonwebtoken (JWT), Middleware

Express

Funcionalidades

Registro seguro de usuarios: Encriptación con bcrypt, inserción en base de datos con asociación a una comunidad.

Inicio de sesión (login): Validación de credenciales, generación de un token JWT con expiración.

Middleware de autenticación: Protege rutas sensibles, valida la existencia, integridad y vigencia del token.
Obtención del usuario autenticado: Recupera los datos del usuario desde el token, devuelve datos personales y nombre de la comunidad asociada.

2.4.9.1 - Registro de usuario [controllers/usuarioController.js]

Se recibe el nombre, email, contraseña y comunidad_id, la contraseña se encripta con bcrypt y se hace un insert a la tabla usuarios.

```
const crearUsuario = async (req, res) => {
    const { nombre, email, contraseña, comunidad_id } = req.body;
    try {
        const contraseñaEncriptada = await encriptarContraseña(contraseña);
        const resultado = await client.query(
            'INSERT INTO usuarios (nombre, email, contraseña, comunidad_id) VALUES ($1, $2, $3, $4) RETURNING *',
            [nombre, email, contraseñaEncriptada, comunidad_id]
        );
        res.status(201).json(resultado.rows[0]);
    } catch (err) {
        console.error('Error al crear usuario:', err);
        res.status(500).send('Error al crear usuario');
    }
};
```

2.4.9.2 - Login [controllers/usuarioControllers.js]

Se filtra por el email insertado, se compara la contraseña ingresada con el hash en la base de datos (bcrypt.compare), si es correcta genera un token JWT.

```
const login = async (req, res) => {
    const { email, contraseña } = req.body;
    try {
        const resultado = await client.query('SELECT * FROM usuarios WHERE email = $1', [email]);
        if (resultado.rows.length === 0) {
            return res.status(400).json({ message: 'Usuario no encontrado' });
        }
        const usuario = resultado.rows[0];
        const esValido = await verificarContraseña(contraseña, usuario.contraseña);
        if (!esValido) {
            return res.status(400).json({ message: 'Contraseña incorrecta' });
        }
        const token = jwt.sign(
            { id: usuario.id, rol: usuario.rol },
            'Walking_33',
            { expiresIn: '1h' }
        );
        res.json({ token });
    } catch (error) {
        console.error('Error en login:', error);
        res.status(500).json({ message: 'Error en el servidor' });
    }
};
```

2.4.9.3 - Middleware de autenticación [middlewares/auth.js]

Este middleware protege rutas verificando el token JWT, si el token no existe o es inválido saltará (403, 401)

```
const verificarToken = (req, res, next) => {
    const token = req.headers['authorization']?.split(' ')[1];

    if (!token) {
        return res.status(403).json({ message: 'Token no proporcionado' });
    }

    try {
        const decoded = jwt.verify(token, 'Walking_33');
        req.usuario = decoded;
        next();
    } catch (error) {
        return res.status(401).json({ message: 'Token inválido o expirado' });
    }
};
```

2.4.9.4 - Obtención de datos del usuario actual [ruta protegida /usuario/me]

Usa el id decodificado desde el token para buscar datos del usuario, además trae el nombre de la comunidad a la que pertenece.

```
router.post('/login', usuarioController.login);
router.post('/registro', usuarioController.crearUsuario);
router.post('/verificar', usuarioController.verificarUsuario);
router.get('/', usuarioController.obtenerUsuarios);
router.get('/me', verificarToken, usuarioController.obtenerUsuarioActual);
router.get('/:id', usuarioController.obtenerUsuarioPorId);
```

```
const obtenerUsuarioActual = async (req, res) => {
    const usuarioId = req.usuario.id;

    try {
        const resultadoUsuario = await client.query('SELECT id, nombre, email, comunidad_id, rol FROM usuarios WHERE id = $1', [usuarioId]);

        if (resultadoUsuario.rows.length === 0) {
            return res.status(404).json({ mensaje: 'Usuario no encontrado' });
        }

        const usuario = resultadoUsuario.rows[0];
        console.log('Datos del usuario:', usuario);

        const resultadoComunidad = await client.query('SELECT nombre FROM comunidades WHERE id = $1', [usuario.comunidad_id]);

        if (resultadoComunidad.rows.length === 0) {
            return res.status(404).json({ mensaje: 'Comunidad no encontrada' });
        }

        usuario.comunidad_nombre = resultadoComunidad.rows[0].nombre;

        res.status(200).json(usuario);
    } catch (err) {
        console.error('Error al obtener el usuario actual:', err);
        res.status(500).json({ mensaje: 'Error en el servidor' });
    }
};
```

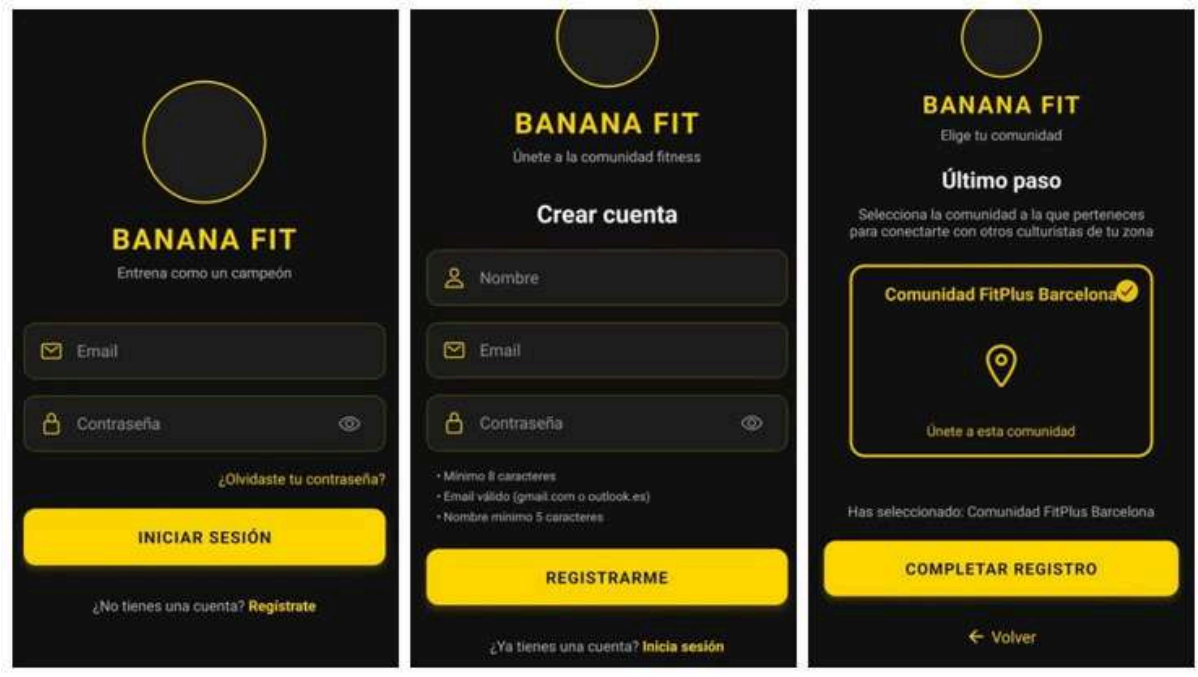
2.5 Definición de las funcionalidades

Aunque en el apartado 2.4 Descripción de los componentes ya se ha realizado una explicación técnica y modular de las diferentes partes que componen la solución, en este bloque se pretende recoger una visión más funcional, centrada en qué permite hacer la aplicación, cómo lo hace y a quién va dirigida cada funcionalidad.

2.5.1 Registro e inicio de sesión

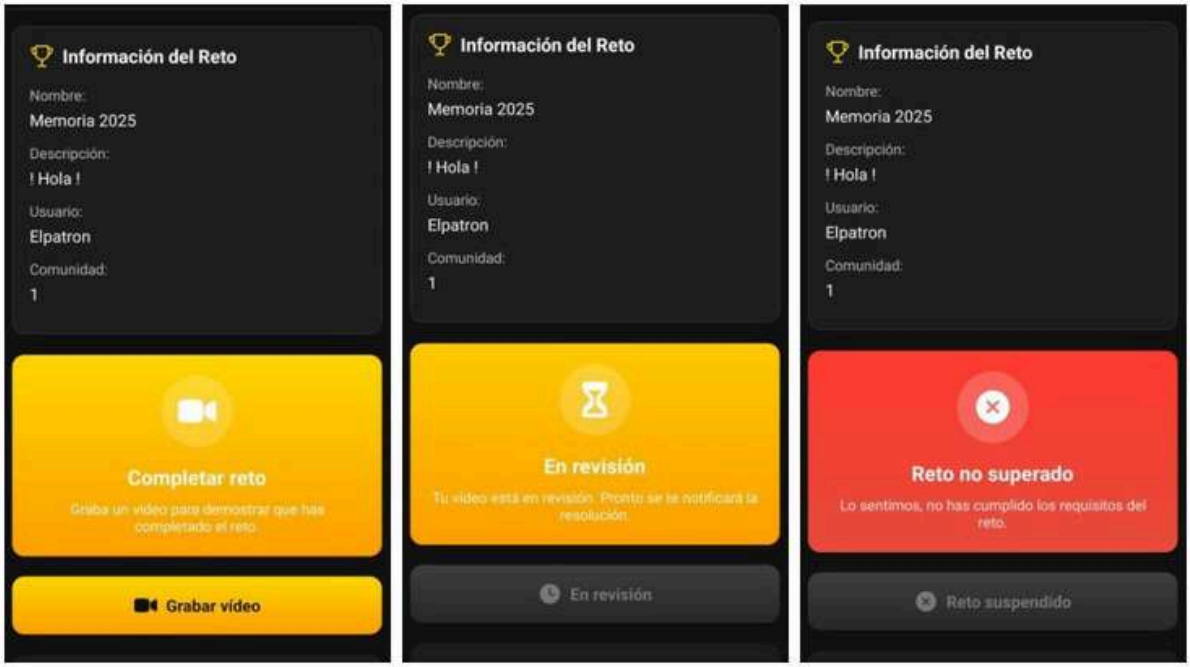
La aplicación permite que nuevos usuarios se registren introduciendo su nombre, correo electrónico, contraseña y la comunidad a la que desean unirse. Esta funcionalidad asegura que cada usuario esté correctamente identificado y asociado a su comunidad correspondiente.

Una vez registrado, el usuario puede iniciar sesión con sus credenciales. Si los datos son válidos, el sistema genera un token JWT, que se almacena localmente y sirve para autenticar futuras acciones dentro de la plataforma.
Este sistema garantiza una experiencia fluida y segura, protegiendo el acceso a funciones privadas.



2.5.2 Participación en retos mediante vídeos

Una de las funcionalidades principales es la posibilidad de participar en retos semanales asignados por la comunidad. Cada usuario puede acceder a un reto, grabarse en vídeo ejecutándolo y subir esa grabación a la plataforma.
Este vídeo es enviado al servidor a través de la app, almacenado en un sistema de almacenamiento en la nube (Cloudinary), y asociado al reto y usuario correspondientes.



2.5.3 Validación de retos por parte del administrador

Los vídeos enviados no se aprueban automáticamente. Un administrador debe revisar cada uno y decidir si el reto ha sido cumplido correctamente o no. Para ello, se ha implementado un panel de validación donde el administrador puede:

- Visualizar el vídeo del usuario
- Ver los detalles del reto
- Aprobar o rechazar la participación

En caso de ser aprobado, el sistema activa automáticamente la asignación de puntos al usuario.

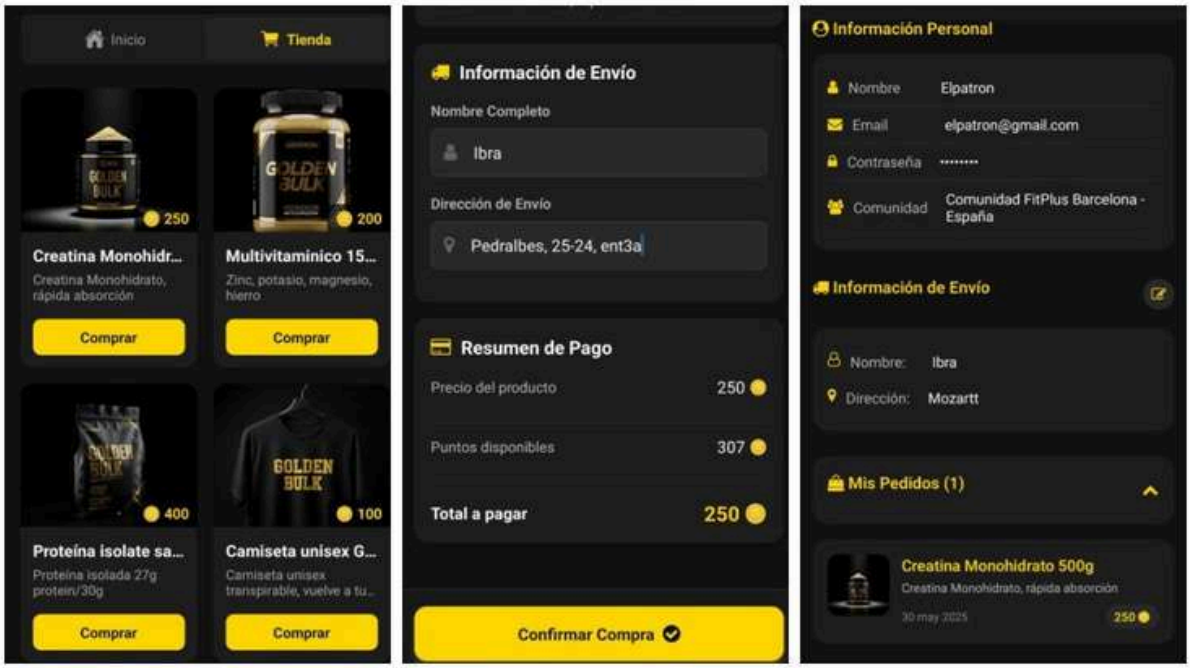
2.5.4 Sistema de puntuación

El sistema de puntos permite gamificar la experiencia de los usuarios. Cada reto tiene una cantidad de puntos predefinida.
Cuando un administrador aprueba un vídeo, los puntos correspondientes se asignan al usuario automáticamente.
Además, el usuario puede consultar en todo momento su balance de puntos acumulados

.

2.5.5 Tienda virtual y canje de puntos

Los usuarios pueden acceder a una tienda virtual dentro de la app. Desde ahí, pueden ver una lista de productos y canjear sus puntos por ellos. El sistema valida que el usuario tenga los puntos necesarios, actualiza el stock del producto y registra la transacción.



2.5.6 Feed de publicaciones sociales

La plataforma incluye un muro de publicaciones gestionado por los administradores.
Estos pueden subir imágenes o vídeos acompañados de texto, para motivar a los usuarios o compartir novedades.
Los usuarios, por su parte, pueden interactuar con estas publicaciones mediante:

- Likes
- Comentarios

Esta funcionalidad fomenta la interacción social dentro de la app.

2.5.7 Chat en tiempo real por comunidad

Cada comunidad dispone de un chat exclusivo que permite a sus miembros comunicarse en tiempo real.
Este sistema incluye:

- Envío de mensajes instantáneos
- Indicadores de usuario en línea
- Confirmaciones de lectura (✓ / ✓✓)

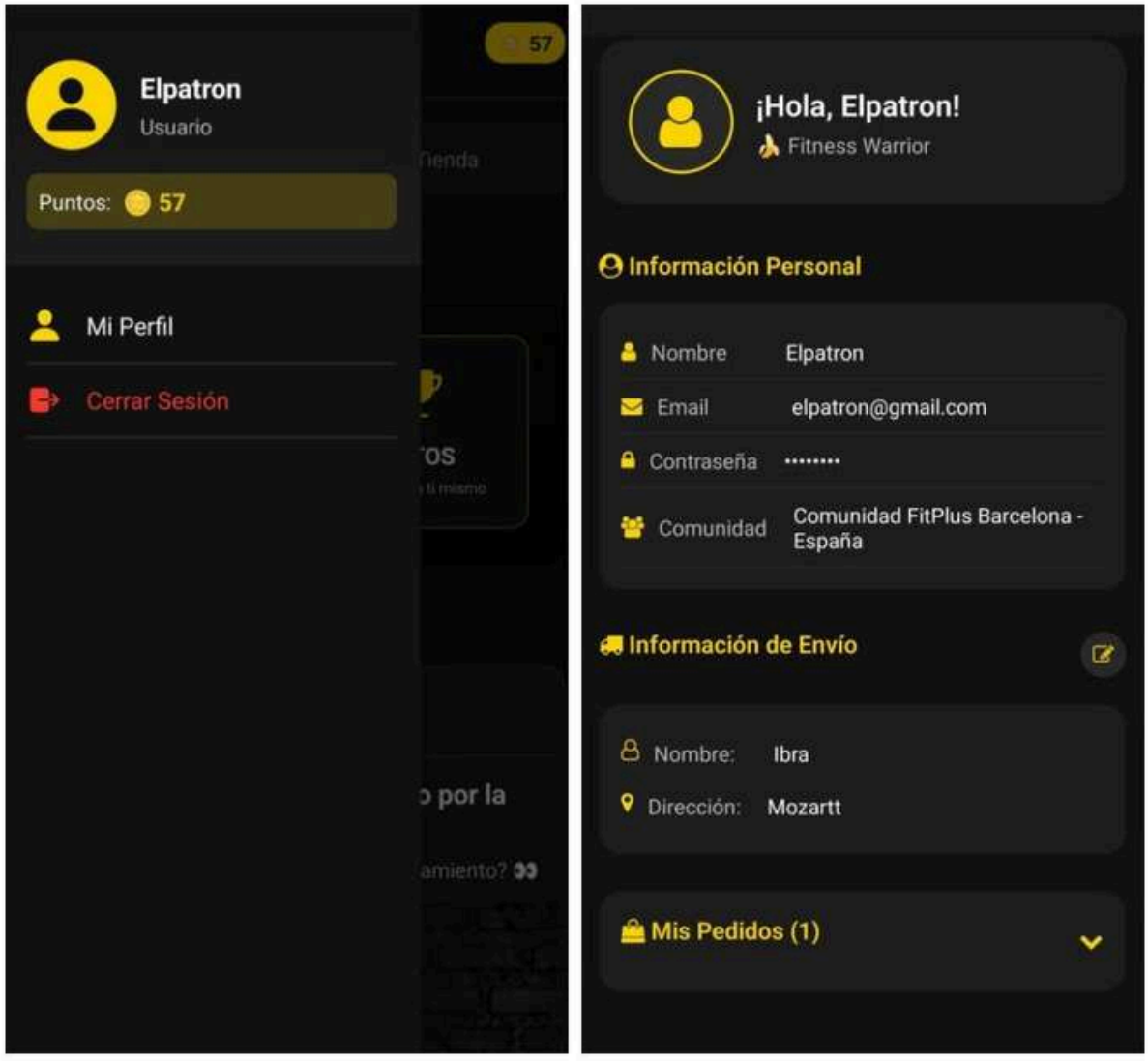
Gracias a la integración con Firebase Realtime Database, los mensajes se actualizan de forma automática sin necesidad de recargar la app.

2.5.8 Gestión del perfil personal

Cada usuario puede acceder a su perfil desde el menú lateral. Desde ahí puede:

- Consultar su nombre y puntos
- Ver la comunidad a la que pertenece
- Cerrar sesión

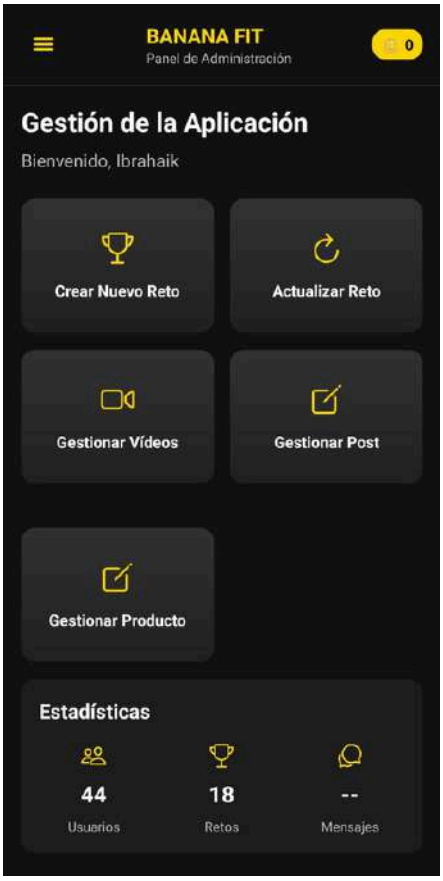
Esta sección es básica pero funcional, y podría ser ampliada en versiones futuras para incluir más personalización.



2.5.9 Panel de administración

Los administradores disponen de un panel avanzado que permite gestionar toda la plataforma desde una interfaz web.
Desde este panel pueden:

- Crear y actualizar retos
- Validar o rechazar vídeos
- Subir publicaciones al feed
- Añadir productos a la tienda
- Consultar estadísticas globales (usuarios, puntos otorgados, vídeos enviados, etc.)



4. Conclusiones

4.1 Conclusiones generales del proyecto

El desarrollo de Golden Bulk ha supuesto un reto integral tanto a nivel técnico como personal. La aplicación no solo ha permitido aplicar los conocimientos adquiridos a lo largo del ciclo formativo, sino que también ha sido una oportunidad para profundizar de manera autodidacta en tecnologías avanzadas como React Native, Firebase, Node, Express y JavaScript, que son actualmente relevantes en el ámbito profesional.

El objetivo principal del proyecto se ha cumplido con creces. Golden Bulk ofrece un ecosistema completo que combina interacción social, gamificación y recompensas, todo ello integrado en una arquitectura modular, escalable y adaptable. Se ha priorizado en todo momento la experiencia del usuario, construyendo una solución robusta, fluida y funcional. Además, el proyecto ha evidenciado la capacidad de planificar, desarrollar e integrar un sistema complejo de forma individual, afrontando las dificultades técnicas con criterio y resolución. El resultado final no es solo un producto académico, sino una plataforma real y funcional con posibilidades de evolución y despliegue comercial.

4.2 Consecución de los objetivos

Los objetivos planteados al inicio del proyecto, tanto generales como específicos, se han cumplido en su totalidad:

Se ha desarrollado una aplicación móvil multiplataforma funcional, utilizando React Native con Expo para maximizar compatibilidad y eficiencia.

Se ha construido una arquitectura modular que facilita la escalabilidad y el mantenimiento del sistema, con separación clara entre frontend, backend y servicios externos.

Se ha implementado un sistema completo de autenticación basado en JWT, garantizando seguridad en el acceso y manejo de datos sensibles.

Se han integrado funcionalidades clave como: retos semanales, subida de vídeos, validación, puntuación, feed social, chat en tiempo real y tienda virtual.

Se ha diseñado un panel de administración eficaz, desde el que se gestionan todos los elementos críticos de la plataforma (retos, publicaciones, usuarios, vídeos y estadísticas).

Se ha conseguido una experiencia de usuario cuidada y coherente, compatible con Android e iOS.

4.3 Valoración de la metodología y planificación

La metodología seguida ha sido **adaptativa e iterativa**, combinando planificación estratégica inicial con una ejecución flexible y dinámica. A pesar de no haber utilizado herramientas formales como Gantt o Trello, la organización del trabajo en fases (análisis, selección tecnológica, desarrollo e integración) ha resultado efectiva.

El enfoque autodidacta ha sido especialmente relevante: se han abordado tecnologías nuevas en tiempo real durante el proceso de construcción del proyecto, enfrentando barreras técnicas y aprendiendo a resolverlas mediante documentación oficial, foros y pruebas locales. Esto no solo ha enriquecido el proyecto, sino también el perfil profesional. La dedicación constante, la toma de decisiones fundamentadas y la validación de cada módulo mediante pruebas funcionales y de integración han sido claves para garantizar la calidad del producto final.

4.4 Visión de futuro

Golden Bulk se ha concebido desde el inicio como una plataforma escalable y adaptable. Su diseño técnico permite evolucionar el sistema en diversas direcciones, tanto a nivel de producto como de negocio. A continuación, se plantean algunas posibles líneas de mejora y expansión:

- **Membresías premium:** Incluir planes de pago con ventajas exclusivas como retos avanzados, recompensas adicionales o entrenadores personales.
- **Sistema de notificaciones push:** Usar Firebase Cloud Messaging para alertar a los usuarios de nuevos retos, publicaciones o mensajes.
- **Inteligencia artificial aplicada a la validación de retos:** Integrar un modelo de reconocimiento de vídeo que detecte automáticamente si el reto ha sido completado correctamente.
- **Expansión a versión web:** Desarrollar una plataforma web complementaria para dar acceso desde navegadores y aumentar el alcance de la comunidad.
- **Gamificación avanzada:** Incorporar logros, rankings por comunidades y recompensas dinámicas según el nivel de actividad.
- **Colaboraciones con marcas:** Establecer alianzas con marcas del sector fitness para integrar promociones o productos exclusivos en la tienda.

Estas mejoras no solo potenciarían la experiencia del usuario, sino que abrirían la puerta a la monetización y profesionalización del producto, acercando el proyecto a una solución comercial real.

5. Glosario

Término	Definición
API REST (Representational State Transfer)	Conjunto de reglas que permiten la comunicación entre sistemas utilizando el protocolo HTTP. En Golden Bulk, las API REST permiten la comunicación entre la app móvil y el servidor backend para gestionar retos, usuarios, puntuaciones, etc.
Autenticación JWT (JSON Web Token)	Sistema de autenticación sin estado (stateless), que usa tokens firmados para validar la identidad del usuario. Una vez logueado, el usuario recibe un token que debe incluir en cada petición protegida.
Backend	Parte del sistema que se ejecuta en el servidor. Gestiona la lógica de negocio, conexión con bases de datos, almacenamiento de archivos, autenticación y enrutado de solicitudes HTTP: En Golden Bulk se ha desarrollado con ~~Node.js~~ y Express
Base de datos relacional (PostgreSQL)	Sistema de almacenamiento estructurado de datos que permite la creación de tablas relacionadas entre sí. PostgreSQL es el motor utilizado para almacenar usuarios, retos, puntuaciones, vídeos, comentarios y productos.
Bcrypt	Algoritmo de encriptación utilizado para proteger contraseñas de forma segura de ser almacenadas en la base de datos. Permite validar contraseñas sin necesidad desencriptarlas.
Chat en tiempo real	Funcionalidad que permite enviar y recibir mensajes instantáneamente. En Golden Bulk se ha implementado mediante Firebase Realtime Database, permitiendo presencia online y confirmaciones de lectura.
Cloudinary	Servicio en la nube especializado en la gestión de archivos multimedia. Se usa para almacenar, optimizar y servir imágenes y vídeos subidos por los usuarios y administradores.

Comunidad	Agrupación lógica de usuarios dentro de la app. Cada comunidad puede tener sus propios retos, tienda y feed social, lo que permite personalización por grupos.
Expo	Herramienta que simplifica el desarrollo de apps móviles con React Native, proporcionando un conjunto de APIs preconfiguradas para cámara, almacenamiento, notificaciones, etc.
Feed social	Sección de la app donde se muestran publicaciones de texto, imágenes o vídeos creadas por los administradores. Los usuarios pueden interactuar con ellas mediante "likes" y comentarios.
Firebase Realtime Database	Base de datos en la nube que sincroniza datos en tiempo real. Utilizada en Golden Bulk para el chat comunitario, permitiendo actualizaciones instantáneas entre usuarios.
Frontend	Parte visual del sistema con la que interactúa el usuario. En este proyecto, el frontend es una app móvil creada con React Native que comunica con el backend y muestra toda la lógica de la plataforma.

Gamificación	Técnica que aplica elementos de juego en contextos no lúdicos. Golden Bulk utiliza sistemas de puntos, retos semanales y recompensas para aumentar el compromiso del usuario.
Middleware	Componente que actúa como filtro entre la solicitud del cliente y el controlador del backend. En el caso de Golden Bulk, el middleware se utiliza para verificar tokens y proteger rutas.
Mobile First	Enfoque de diseño y desarrollo donde la versión móvil de una aplicación es prioritaria. Golden Bulk fue concebida desde el inicio como una app móvil centrada en la experiencia del usuario.

Node.js	Entorno de ejecución de JavaScript del lado del servidor. Utilizado para crear la lógica de backend en combinación con Express.
Notificaciones push	Sistema que permite enviar mensajes directamente al dispositivo móvil del usuario, incluso cuando la app está cerrada. Se plantea su inclusión en futuras versiones de Golden Bulk.
React Native	Framework de desarrollo móvil creado por Meta que permite crear apps nativas para Android e iOS con JavaScript. Utilizado para desarrollar el frontend de Golden Bulk.
Retos	Elemento central de la plataforma que consiste en actividades físicas semanales que deben ser completadas por los usuarios, normalmente mediante un vídeo grabado.
Socket	Canal de comunicación bidireccional que permite la transmisión continua de datos. Aunque se usó Firebase para el chat, se exploró también la opción de sockets tradicionales.
Token	Cadena alfanumérica generada tras un inicio de sesión válido que representa la identidad del usuario. En Golden Bulk, los tokens son de tipo JWT.
UX/UI (User Experience / User Interface)	Experiencia de usuario e interfaz gráfica. En Golden Bulk se priorizó una experiencia simple, visualmente clara y orientada a usuarios de cualquier nivel técnico.
Videovalidación	Proceso mediante el cual un administrador revisa manualmente un vídeo enviado por un usuario para validar si ha cumplido correctamente el reto asignado.
Zona privada	Sección de la aplicación accesible solo a usuarios autenticados. Contiene funcionalidades como retos, puntuación, tienda, perfil y chat.

6. Bibliografía

1. Firebase Documentation:
   https://firebase.google.com/docs
2. React Native Documentation:
   https://reactnative.dev/docs
3. PostgreSQL Official Docs:
   https://www.postgresql.org/docs/
4. Cloudinary Documentation:
   https://cloudinary.com/documentation
5. JWT.io – JSON Web Tokens Introduction:
   https://jwt.io/introduction
6. Node.js Documentation:
   https://nodejs.org/en/docs/
7. Express.js Guide:
   https://expressjs.com
8. Bcrypt NPM Docs:
   https://www.npmjs.com/package/bcrypt
9. Expo Documentation:
   https://docs.expo.dev
10. AsyncStorage (Expo):
    https://docs.expo.dev/versions/latest/sdk/async-storage/
11. YouTube – "React Native Firebase Chat App Tutorial" (Fireship):
    https://www.youtube.com/watch?v=2V1FtfBDsLU
12. YouTube – "React Native Camera & Video Recording with Expo" (CodeWithChris):
    https://www.youtube.com/watch?v=7hWCeKgK6d4
13. YouTube – "Cloudinary Tutorial: Image Upload & Optimization" (The Net Ninja):
    https://www.youtube.com/watch?v=2kvePe4bUAs
14. YouTube – "JWT Authentication in Node.js" (Academind):
    https://www.youtube.com/watch?v=7nafaH9SddU
15. LogRocket Blog – "React Native performance tips":
    https://blog.logrocket.com/optimize-react-native-performance/
16. FreeCodeCamp – "Understanding REST APIs":
    https://www.freecodecamp.org/news/rest-api-tutorial/
17. CSS Tricks – "Flexbox Guide":
    https://css-tricks.com/snippets/css/a-guide-to-flexbox/
18. ChatGPT – OpenAI:
    https://chat.openai.com

7. Problemas técnicos

Durante el desarrollo de Golden Bulk se identificaron diversos errores técnicos relacionados con la gestión del estado, la navegación entre pantallas, la sincronización de datos y la integración de servicios externos. A continuación, se describen algunos de los más representativos y cómo fueron solucionados.

1. community_id nulo en la navegación tras el login

Incidencia:
Tras iniciar sesión correctamente, la aplicación permitía acceder a las diferentes pantallas del sistema. Sin embargo, se detectó que el valor de community_id —clave para filtrar datos según la comunidad— llegaba como null en algunas pantallas al navegar, aunque sí estaba presente en el token JWT.
Análisis:

El problema radicaba en que el valor de community_id se pasaba por props o por parámetros de navegación (route.params), lo cual perdía persistencia al moverse entre pantallas sin volver a pasar explícitamente dichos datos.
Solución aplicada:
Se implementó un sistema centralizado para recuperar community_id directamente desde el token JWT almacenado en AsyncStorage, evitando depender de la navegación para acceder a este dato. Esto garantizó la disponibilidad del identificador de comunidad en todo el ciclo de vida de la sesión del usuario.

2. Mal funcionamiento del menú lateral tras navegar hacia atrás

Incidencia:
El menú lateral implementado en Home.js mediante DrawerNavigator presentaba comportamientos erráticos al navegar hacia atrás o salir de la aplicación. Los datos mostrados quedaban obsoletos o se producían errores de navegación.
Análisis:
React Navigation por defecto mantiene el estado montado de los navegadores incluso cuando el usuario cambia de pantalla. Al no volver a montar el componente, el contenido del menú no se actualizaba al reingresar a la pantalla Home.

Solución aplicada:
Se encapsuló la lógica del menú dentro de un hook useFocusEffect, que permite ejecutar código cada vez que una pantalla recibe el foco. De este modo, el menú se actualiza y remonta correctamente cada vez que el usuario vuelve a Home, garantizando una experiencia de navegación coherente.

3. Error al subir archivos de vídeo de gran tamaño

Incidencia:
Algunos usuarios grababan vídeos extensos para completar retos, lo que generaba archivos demasiado grandes. Cloudinary rechazaba estas peticiones con errores 413 `payload too large`.

Análisis:
La duración ilimitada de grabación generaba vídeos que superaban el límite permitido por la API gratuita de Cloudinary.
Solución aplicada:
Se impuso una duración máxima de grabación de 30 segundos en la app móvil mediante la configuración de la API de cámara de Expo. Adicionalmente, se habilitó la compresión automática del vídeo antes de ser subido, reduciendo considerablemente su tamaño.

4. Listeners de Firebase duplicados en el chat

Incidencia:
Al acceder repetidamente a la pantalla de chat, los mensajes se duplicaban o aparecían múltiples veces. Esto indicaba la presencia de listeners activos duplicados.
Análisis:

No se estaba cerrando correctamente la suscripción a onValue() en Firebase al desmontar la pantalla, por lo que cada acceso creaba un nuevo listener adicional. **Solución aplicada:**
Se encapsuló la lógica de suscripción dentro de un useEffect que devuelve una función de limpieza (return unsubscribe). Así se asegura que al desmontar el componente se cierre el listener correspondiente, evitando duplicación de datos en pantalla y consumo innecesario de recursos.

5. Expiración prematura del token JWT

Incidencia:
Se producía la desconexión inesperada de algunos usuarios pocos minutos después de iniciar sesión, obligándolos a autenticarse nuevamente.
Análisis:
La expiración del token JWT estaba configurada por defecto con una duración demasiado corta (1h), lo cual no se ajustaba al flujo real de uso de la aplicación.

Solución aplicada:
Se extendió el tiempo de expiración del token a 24 horas (expiresIn: '24h') y se añadió verificación de validez antes de cada acción sensible, mejorando así la experiencia de usuario sin comprometer la seguridad.

6. Desincronización del estado online en el chat

Incidencia:
El estado de conexión de los usuarios en el chat comunitario no reflejaba con precisión cuándo un usuario estaba realmente desconectado, lo cual generaba confusión en la interfaz.
Análisis:
La ausencia de un mecanismo de desconexión explícito provocaba que Firebase mantuviera el estado online de forma persistente incluso tras cerrar la aplicación o perder conexión.
Solución aplicada:
Se utilizó onDisconnect() para establecer automáticamente el estado offline cuando el usuario cierra la app o se desconecta de internet. Esta función nativa de Firebase permite una gestión confiable del estado en tiempo real.

7. Problemas de sincronización al asignar puntos tras validar retos

Incidencia:
En ciertos casos, la validación de retos no resultaba en la asignación efectiva de puntos al usuario, generando inconsistencias en la puntuación total.
Análisis:

El sistema ejecutaba las consultas de actualización de estado del reto y de asignación de puntos de forma paralela, sin garantizar el orden de ejecución correcto.
Solución aplicada:
Se refactorizó el flujo para utilizar async/await de manera secuencial, asegurando que la asignación de puntos solo se ejecute tras la validación exitosa del reto. Se añadió, además, control de errores y confirmación explícita de ambas operaciones.

8. Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todas las personas que me han acompañado y apoyado durante el desarrollo de este proyecto.

En primer lugar, agradezco especialmente al equipo docente del crédito de síntesi, por su constante apoyo, orientación y disponibilidad a lo largo del proceso y al resto de profesores que han formado parte del seguimiento y evaluación del trabajo, por sus observaciones técnicas y su implicación en cada fase del proyecto.

Este proyecto ha representado no solo un reto técnico, sino una experiencia de crecimiento personal y profesional, y me enorgullece haberlo completado con el acompañamiento de un equipo educativo comprometido y exigente.

9. Github

Link github código frontend: https://github.com/ibrahaik/FitPlus:

Link github código backend: https://github.com/ibrahaik/BACKEND