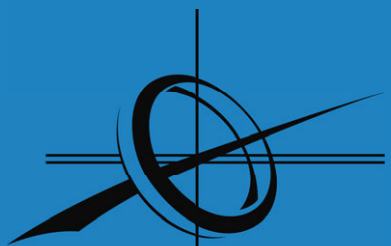
POLITÉCNICA



Universidad Politécnica de Madrid

ETSI SISTEMAS INFORMÁTICOS

Diseño y Programación de una Aplicación Web para la Gestión Personal

Proyecto Fin de Grado

Grado en Ingeniería del Software

Autores:
Xusheng Qiu Huang
Eduardo Segarra Ledesma

Tutor:
Borja Bordel Sánchez

Septiembre 2025

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Diseño y Programación de una Aplicación Web para la Gestión Personal

Proyecto Fin de Grado

Grado en Ingeniería del Software

Curso académico 2025-2026

Autores:
Xusheng Qiu Huang
Eduardo Segarra Ledesma

Tutor:
Borja Bordel Sánchez

Queremos agradecer a la universidad por la formación y el apoyo brindados.
A nuestro tutor, Borja Bordel Sánchez, por su guía, disponibilidad y confianza.
A nuestras familias por su apoyo incondicional.
Y a nuestros compañeros de carrera, Andrés, Alicia, Daniel y Andriy, cuyo acompañamiento y ánimo han sido determinantes.
Sin ellos no estaríamos donde estamos hoy en términos académicos.

Resumen

Este trabajo tiene como objetivo el diseño, desarrollo e implementación de una aplicación web para el seguimiento de metas personales, con el objetivo de ofrecer una experiencia clara y accesible desde el navegador para crear, editar, actualizar y eliminar metas, visualizar el progreso y desplegar la solución íntegramente en la nube, validando su uso con personas usuarias. La propuesta combina simplicidad de producto y una base tecnológica robusta.

La metodología de desarrollo sigue un proceso ágil con Kanban y apoyo de Gantt: columnas Backlog → Ready → In Progress → Testing → Done, límites WIP y políticas explícitas de entrada/salida para asegurar calidad antes de dar una tarea por terminada. El backlog cubre back end, front end, autenticación, tests y evaluación. La evaluación incluye tests con usuarios reales mediante un cuestionario en Google Forms para medir usabilidad y validar criterios de aceptación.

El proyecto entrega GoLife, una aplicación web accesible desde el navegador de escritorio que materializa un flujo completo de autenticación, dashboard y CRUD de metas y registros. La interfaz, construida como SPA, ofrece vistas de trabajo con acciones por meta (crear registro, editar, finalizar y eliminar metas) y gráficos con filtros de periodo y rango para analizar la evolución; los formularios incorporan validación, estados deshabilitados y confirmaciones para prevenir errores, con actualización inmediata de tablas y estadísticas.

En la infraestructura cloud utilizada, el front end se sirve desde Firebase Hosting y su red CDN; la autenticación se delega en Firebase Authentication; la API se ejecuta en Google App Engine con versionado y enrutado gestionados; los datos residen en MongoDB Atlas; y la seguridad se refuerza con Cloud KMS para cifrado y verificación de campos sensibles. Esta arquitectura 100 % cloud y gestionada reduce la carga operativa y habilita escalado y alta disponibilidad sin reingeniería. El despliegue y la operación se automatizan mediante pipelines CI/CD con GitHub Actions, inyectando secrets de forma controlada y publicando versiones reproducibles en App Engine y Hosting.

En conclusión, el proyecto cumple los objetivos: materializa una plataforma esencial y usable, desplegada 100 % en la nube con servicios gestionados, y establece una base sólida para evolucionar. Destaca el aprendizaje de que “menos es más” y que la nube actúa como acelerador de producto sin comprometer el crecimiento futuro.

Palabras clave: gestión personal; metas; aplicación web; despliegue en la nube; arquitectura en la nube.

Abstract

The objective of this project is to design, develop, and implement a web application for tracking personal goals, with the aim of offering a clear and accessible browser experience to create, edit, update, and delete goals, view progress, and deploy the solution entirely in the cloud, validating its use with users. The proposal combines product simplicity and a robust technological base.

The development methodology follows an agile process with Kanban and Gantt support: Backlog → Ready → In Progress → Testing → Done columns, WIP limits, and explicit entry/exit policies to ensure quality before completing a task. The backlog covers back end, front end, authentication, testing, and evaluation. Evaluation includes testing with real users using a Google Forms questionnaire to measure usability and validate acceptance criteria.

The project delivers GoLife, a web application accessible from a desktop browser that provides a complete authentication flow, dashboard, and CRUD for goals and records. The interface, built as an SPA, offers work views with actions per goal (create record, edit, finalize, and delete goals) and graphs with period and range filters to analyze progress; the forms incorporate validation, disabled states, and confirmations to prevent errors, with immediate updating of tables and statistics.

In the cloud infrastructure used, the front end is served from Firebase Hosting and its CDN network; authentication is delegated to Firebase Authentication; the API runs on Google App Engine with managed versioning and routing; data resides in MongoDB Atlas; and security is reinforced with Cloud KMS for encryption and verification of sensitive fields. This 100 % cloud-based and managed architecture reduces operational overhead and enables scaling and high availability without reengineering. Deployment and operation are automated using CI/CD pipelines with GitHub Actions, injecting secrets in a controlled manner and publishing reproducible versions to App Engine and Hosting.

In conclusion, the project meets its objectives: it delivers an essential and usable platform, deployed 100 % in the cloud with managed services, and establishes a solid foundation for future evolution. The key lessons learned are that “less is more” and that the cloud acts as a product accelerator without compromising future growth.

Keywords: personal management; goals; web application; cloud deployment; cloud architecture.

Glosario

Entradas ordenadas alfabéticamente:

A

ACID: (Atomicity, Consistency, Isolation, Durability) propiedades transaccionales que garantizan fiabilidad en operaciones de base de datos.

API: (Application Programming Interface) interfaz de programación que define cómo se comunican aplicaciones y servicios.

Aplicación web: software accesible desde el navegador que se ejecuta en un servidor remoto.

App Engine: servicio PaaS de Google para desplegar aplicaciones con infraestructura, escalado y enrutado gestionados.

B

BaaS: (Backend as a Service) plataforma que ofrece servicios de backend listos (autenticación, base de datos, notificaciones, etc.).

Back end: capa del sistema que corre en el servidor y gestiona lógica de negocio, datos y APIs.

Backlog: lista priorizada de tareas o requisitos pendientes de desarrollo.

BASE: (Basically Available, Soft state, Eventual consistency) principios alternativos a ACID para sistemas distribuidos.

Bearer: esquema de autorización HTTP que transmite un *token* en la cabecera `Authorization: Bearer <token>`.

BSON: (Binary JSON) formato binario basado en JSON utilizado, entre otros, por MongoDB.

C

CDN: (Content Delivery Network) red de servidores que distribuye contenido estático acercándolo al usuario para reducir latencia.

CI/CD: (Continuous Integration / Continuous Delivery) prácticas y canalizaciones para integrar, probar y desplegar cambios de forma continua.

Cloud: computación en la nube; provisión de recursos y servicios informáticos a través de Internet bajo demanda.

Cloud KMS / KMS: (Key Management Service) servicio de gestión de claves criptográficas para cifrar y firmar datos de forma segura.

Cluster: conjunto de máquinas o instancias que trabajan coordinadamente como un único sistema.

CORS: (Cross-Origin Resource Sharing) mecanismo de HTTP que controla solicitudes entre orígenes distintos.

CRUD: (Create, Read, Update, Delete) operaciones básicas de persistencia sobre datos.

Cycle time: tiempo desde que una tarea comienza a desarrollarse hasta que se completa.

D

DAO: (Data Access Object) patrón que encapsula el acceso a datos y las operaciones CRUD.

Dashboard: panel de control con indicadores, tablas o gráficos para seguimiento y análisis.

DBaaS: (Database as a Service) servicio gestionado de base de datos operado por un proveedor.

Deployment: despliegue; proceso de publicar o actualizar una aplicación en un entorno de ejecución.

Docker: plataforma de contenedores que empaqueta aplicación y dependencias en imágenes portables, ejecutadas como contenedores ligeros y aislados.

DoD: (Definition of Done) criterios que deben cumplirse para considerar una tarea terminada.

DoR: (Definition of Ready) criterios que debe cumplir una tarea para entrar en desarrollo.

DTO: (Data Transfer Object) objeto sin lógica que transporta datos entre capas o servicios.

E

E2E: (End to End) recorrido de extremo a extremo que validan el flujo real completo: cliente, API, base de datos.

Endpoints: URLs o rutas de una API que exponen recursos u operaciones.

F

Feedback: retroalimentación; comentarios que ayudan a mejorar producto o proceso.

Firebase Authentication: servicio gestionado de identidad y autenticación de Firebase (inicio de sesión y gestión de usuarios).

Firebase Hosting: servicio de alojamiento estático con CDN integrado para publicar sitios y SPAs.

Front end: capa de presentación que se ejecuta en el cliente (navegador), interfaz con el usuario.

G

GCP: (Google Cloud Platform) plataforma de servicios en la nube de Google.

GoLife: aplicación desarrollada en este PFG para el seguimiento de metas personales.

gRPC: marco de comunicación de alto rendimiento basado en HTTP/2 y Protocol Buffers para llamadas a procedimiento remoto.

H

HTTPS: (HyperText Transfer Protocol Secure) versión segura de HTTP que cifra el canal mediante TLS.

I

IaaS: (Infrastructure as a Service) provisión de infraestructura virtual (VMs, redes, almacenamiento) como servicio.

IAM: (Identity and Access Management) gestión de identidades y accesos de GCP: define usuarios, roles y permisos para controlar recursos y servicios.

J

Joins: operaciones en SQL que combinan filas de múltiples tablas según relaciones.

JSON: (JavaScript Object Notation) formato ligero de intercambio de datos basado en pares clave–valor.

JWT: (JSON Web Token) token firmado (y opcionalmente cifrado) para autenticación y autorización.

K

Kanban: método ágil de gestión del trabajo en flujo continuo con límites WIP.

L

Lead time: tiempo desde que se solicita una tarea hasta que se entrega al usuario.

M

MAC: (Message Authentication Code) código de autenticación que verifica integridad y autenticidad de un mensaje con clave.

MongoDB Atlas: servicio DBaaS de MongoDB para bases de datos administradas en la nube.

N

NoSQL: familias de bases de datos no relacionales (clave–valor, documento, columna, grafo), "Not Only SQL".

P

PaaS: (Platform as a Service) plataforma gestionada para ejecutar aplicaciones sin administrar servidores subyacentes.

PFG: Proyecto Fin de Grado; trabajo académico final para la obtención del título de grado.

PII: (Personally Identifiable Information) información de identificación personal (datos que identifican a una persona).

POJO: (Plain Old Java Object) objeto Java simple, sin dependencias ni comportamientos especiales de frameworks.

R

Rate limiting: límite de peticiones por usuario o IP durante un intervalo, previniendo abuso y protegiendo el servicio.

REST / RESTful: (Representational State Transfer) estilo arquitectónico para sistemas distribuidos, común en APIs HTTP.

RGPD: Reglamento General de Protección de Datos (GDPR); normativa europea de protección de datos personales.

S

SaaS: (Software as a Service) software ofrecido como servicio accesible por Internet.

SDK: (Software Development Kit) conjunto de librerías, herramientas y documentación para desarrollar sobre una plataforma.

Seudonimización: reemplazo de identificadores personales por seudónimos, reduciendo exposición de datos sensibles.

SPA: (Single-Page Application) aplicación de una sola página que carga una vez y actualiza vistas dinámicamente en el cliente.

SQL: (Structured Query Language) lenguaje estándar para gestionar bases de datos relacionales.

Stateless: la API no almacena estado de sesión entre peticiones; cada solicitud se procesa independientemente.

Swagger: conjunto de herramientas (OpenAPI) para describir, documentar y probar APIs REST.

T

Throttling: limitación de la tasa de peticiones o uso de recursos para proteger un sistema.

Throughput: rendimiento; cantidad de trabajo procesado por unidad de tiempo.

TLS: (Transport Layer Security) protocolo criptográfico que asegura la comunicación en red.

U

UI: (User Interface) interfaz de usuario; elementos con los que interactúa el usuario.

UX: (User Experience) experiencia de usuario; percepción y satisfacción del usuario con el producto.

V

Vendor lock-in: dependencia de un proveedor que dificulta migrar a otra plataforma o servicio.

VM: (Virtual Machine) emulación de un sistema informático que ejecuta un sistema operativo aislado.

W

WIP: (Work in Progress) trabajo en curso; límite que controla la cantidad de tareas simultáneas.

Wireframes: bocetos de baja fidelidad de pantallas para definir estructura y flujo antes del diseño final.

Índice

Índice de tablas

Índice de figuras

Índice de listados

Capítulo 1

Introducción

1.1. Contexto

En un mercado sobresaturado de aplicaciones para el desarrollo personal, es habitual encontrar plataformas repletas de funcionalidades avanzadas, gamificación compleja, infinitas opciones de configuración, integraciones y opciones de configuración que terminan generando complejidad y elevada curva de aprendizaje. Este exceso de características, lejos de facilitar el logro de objetivos, puede distraer al usuario y dificultar la adopción de la herramienta.

Paralelamente, el auge de los servicios en la nube y las plataformas PaaS/IaaS/ SaaS ha simplificado enormemente el despliegue de aplicaciones web escalables y de alta disponibilidad. Servicios como Google Cloud Platform, Microsoft Azure y Amazon Web Services, permiten concentrarse en la lógica de negocio sin preocuparse por la infraestructura subyacente.

GoLife nace en este contexto: aprovechar la potencia de la nube para ofrecer una experiencia de usuario clara, rápida y accesible desde cualquier navegador, manteniendo únicamente las funcionalidades esenciales para el registro y seguimiento de metas.

1.2. Objetivos

Con GoLife, nuestra intencion es presentar un ejercicio práctico de desarrollo web moderno y despliegue en la nube, combinando dos inquietudes principales: por un lado, la curiosidad por experimentar con frameworks actuales y servicios como PaaS/BaaS; por otro, la necesidad de demostrar que una experiencia de usuario clara y directa puede integrarse en una solución robusta y escalable.

En concreto, los objetivos de este proyecto son los siguientes:

- **Ofrecer una plataforma web para el desarrollo personal:** GoLife debe permitir a cualquier usuario definir, monitorizar y gestionar sus metas personales de forma intuitiva y sin barreras de entrada.
- **Centrarse en el seguimiento de metas:** Implementar las funcionalidades esenciales; creación, edición, actualización y eliminación de metas, y un dashboard que muestre el progreso de manera visual y motivadora.
- **Desplegar la solución sobre infraestructura cloud:** Diseñar e implementar una arquitectura que aproveche la flexibilidad, escalabilidad y disponibilidad propias de los entornos en la nube.
- **Hacer el front end accesible públicamente:** Garantizar que la interfaz de usuario esté disponible desde cualquier navegador, sin necesidad de instalaciones locales.

- **Validar la solución con pruebas reales:** Realizar tests de usabilidad y rendimiento, midiendo tiempos de respuesta y recopilando feedback de usuarios para futuras iteraciones sobre el diseño.

A lo largo de este documento, la aplicación web desarrollada será referida como **GoLife**, nombre bajo el cual se agrupan todas las funcionalidades y componentes desarrollados.

1.3. Estructura del documento

El contenido de este documento se distribuye en once capítulos, cada uno con un propósito específico:

- **Capítulo 1. Introducción**

  Presenta el contexto que motiva el desarrollo de GoLife, los objetivos generales del proyecto y la organización del documento.

- **Capítulo 2. Estado del Arte**

  Realiza un estudio sobre la gestión personal, el conocimiento existente sobre aplicaciones de gestión personal y modelos de servicios cloud, terminando con una breve mención de las tecnologías de GoLife.

- **Capítulo 3. Visión General de Producto**

  Despliega un Inception Deck completo para aclarar y presentar la idea y vision de GoLife como un producto.

- **Capítulo 4. Metodología de Desarrollo**

  Explica el proceso ágil elegido, la planificación y las herramientas de seguimiento de tareas.

- **Capítulo 5. Análisis de Requisitos**

  Enumera y describe los requisitos funcionales y no funcionales que GoLife debe cumplir, incluyendo criterios de aceptación.

- **Capítulo 6. Diseño Conceptual y Modelado**

  Muestra los modelados, diagramas y esquemas de base de datos que fundamenta la estructura de GoLife.

- **Capítulo 7. Arquitectura del Sistema**

  Describe la visión de componentes y despliegue, la comunicación entre módulos y los servicios cloud implicados.

- **Capítulo 8. Implementación del Front End**

  Explica la construcción, estructuras y decisiones de diseño en la capa de presentación.

- **Capítulo 9. Implementación de la API**

  Explica la construcción de la API, la lógica de negocio, la persistencia de datos y las pruebas del servidor.

- **Capítulo 10. Evaluación y Resultados**

  Presenta el sistema final, los resultados de las pruebas de usuario y la validación del cumplimiento de requisitos.

- **Capítulo 11. Conclusiones y Trabajos Futuros**

  Resume las lecciones aprendidas, el impacto del proyecto y propone posibles mejoras o extensiones para fases posteriores.

Capítulo 2

Estado del arte

2.1. Definición y ámbito de la gestión Personal

2.1.1. ¿Qué es gestión personal?

En este trabajo entenderemos por **gestión personal** el conjunto de capacidades y prácticas de **autogestión** mediante las cuales una persona orienta su conducta hacia metas significativas, organizando y revisando de forma sistemática su actividad cotidiana. Operativamente, abarca: (a) el seguimiento y la priorización de **tareas** y el uso del tiempo; (b) el diseño y mantenimiento de **hábitos**; (c) el control básico de **finanzas personales**; y (d) la definición, desagregación y revisión de **objetivos**.

En términos psicológicos, la autogestión combina procesos como la autobservación, el establecimiento de metas y el autorrefuerzo para dirigir la conducta hacia resultados deseados (1). En el plano de competencias, se considera una capacidad transversal para aprender y trabajar en entornos cambiantes: favorece la toma de decisiones informadas, el aprendizaje a lo largo de la vida y la adaptación a contextos digitales (2).

2.1.2. Concepto y alcance común

En la práctica, la gestión personal se despliega en cuatro dominios que se retroalimentan:

- **Tareas y uso del tiempo:** Capturar demandas, clarificar próximas acciones, priorizar y planificar. La evidencia sintetizada por el meta-análisis de PLoS One indica que la gestión del tiempo se asocia de forma consistente con mejores resultados (rendimiento académico y laboral) y, especialmente, con mayor bienestar y sensación de control. En la práctica, esto respalda prácticas sencillas como definir próximas acciones claras y proteger bloques de trabajo, porque no solo ayudan a producir más, sino a experimentar menos estrés y más control sobre el día a día (3).
- **Hábitos:** La construcción de hábitos establece automatismos que reducen la fricción ejecutiva y sostienen comportamientos deseados. Las revisiones recientes subrayan que los hábitos funcionan mejor cuando se anclan a señales estables del contexto y se repiten en condiciones parecidas; además, los hábitos no sustituyen el pensamiento deliberado, sino que lo complementan (por ejemplo, combinando un "disparador" claro con una meta y retroalimentación periódica). En otras palabras: diseñar el entorno (señales), repetir con constancia y revisar con realismo es más efectivo que confiar solo en la motivación momentánea (4).
- **Finanzas personales:** La alfabetización y gestión financiera básicas (planificación, seguimiento de gastos/ahorros y toma de decisiones informadas) forman parte del autocuidado instrumental del individuo. De acuerdo con la Recomendación sobre Educación Financiera de la OCDE, la literacia financiera combina conocimientos, habilidades, actitudes y comportamientos que permiten tomar decisiones sólidas y resilientes en el tiempo (5). Traducido al día a día: registrar gastos e ingresos, fijar

objetivos de ahorro y revisar periódicamente el plan no es solo "llevar cuentas"; es ejercer autogestión sobre un recurso crítico.

- **Objetivos:** La fijación y desagregación de metas u objetivos (resultados específicos, medibles y con plazo) da dirección y coherencia a las tareas y hábitos que se planifican en función de estos. La teoría del establecimiento de metas sostiene que objetivos específicos y desafiantes, acompañados de retroalimentación, elevan el desempeño porque orientan la atención, incrementan el esfuerzo y sostienen la persistencia; además, dividir metas en subobjetivos próximos facilita el progreso y la motivación (6). Así, definir metas claras y medir avances periódicos actúa como "esqueleto" que ordena tareas y hábitos.

2.2. Revisión de herramientas representativas

Esta sección ofrece una mirada de referencia al panorama actual de herramientas de gestión personal para identificar patrones de uso, buenas prácticas y fricciones habituales. No busca elaborar un ranking ni fijar requisitos, sino extraer lecciones de diseño útiles para GoLife: qué facilita el arranque, qué hace visible el progreso y qué introduce complejidad innecesaria.

Para ello se analizan cuatro soluciones representativas; **Notion**, **Todoist**, **Habitica** y **Any.do**. Escogidas porque ofrecen propuestas interesantes de cara a la gestión personal. Son herramientas populares (aunque no necesariamente las más masivas) y se han seleccionado precisamente por su diversidad de enfoques y su relevancia práctica para nuestro objetivo.

2.2.1. Notion

Qué es y para quién: Notion (7) es un workspace flexible que combina páginas, notas y bases de datos en un mismo entorno. Está orientado a personas que desean centralizar información y organizar su vida/proyectos con alto grado de personalización, desde uso personal hasta configuraciones más elaboradas.

Dominios que cubre: Tareas ✔ Hábitos ✔ Objetivos ✔ Finanzas simples ✔, el alcance depende de cómo el usuario configure sus plantillas.

Plataformas / Disponibilidad: Web ✔ Móvil (IOS/Android) ✔ Escritorio (Windows/macOS) ✔

Funcionamiento base: El usuario crea una página y añade bloques (texto, listas, tablas, bases de datos). Las bases de datos admiten vistas como tabla, lista, kanban, calendario o galería. El progreso se revisa filtrando/ordenando por propiedades (estado, fecha, etc.). Vease la Figura 2.1.

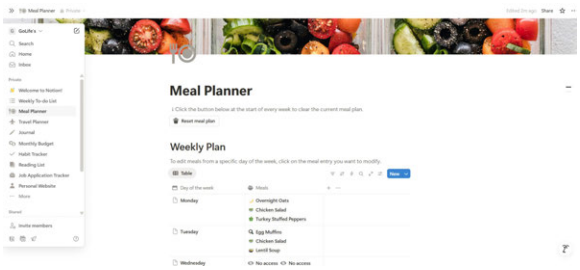
Puntos fuertes:

- **Flexibilidad elevada** con capacidad de creación de páginas y bases de datos con múltiples vistas y propiedades.
- **Plantillas reutilizables** que aceleran la estandarización de rutinas (hábitos, diarios, objetivos).

Puntos débiles:

- **Curva de aprendizaje** y alta fricción inicial por la libertad y capacidad de modelado excesiva.
- **Progreso no estandarizado**, las métricas dependen de cómo se diseñe cada base de datos.

(a) Notion - Página de presentación (7)



(b) Notion - Ejemplo de uso (7)

Figura 2.1: Notion - Vistas

2.2.2. Todoist

Qué es y para quién: Todoist (8) es un gestor de tareas centrado en listas y proyectos, pensado para personas que quieren capturar rápidamente pendientes, planificarlos por fechas y darles seguimiento diario. Apto para uso personal y para colaboración ligera en pequeños equipos.

Dominios que cubre: Tareas ✔ Hábitos ✔ Objetivos ✔ Finanzas simples ✘

Plataformas / Disponibilidad: Web ✔ Móvil (iOS/Android) ✔ Escritorio (Windows/macOS) ✔

Funcionamiento base: El usuario organiza el trabajo en proyectos y secciones, y crea tareas con fecha de vencimiento. Puede asignar prioridades, etiquetas y recordatorios y marcar completadas. Admite tareas periódicas, filtros por propiedades, historial de finalización y colaboración compartiendo proyectos. Vease la Figura 2.2.

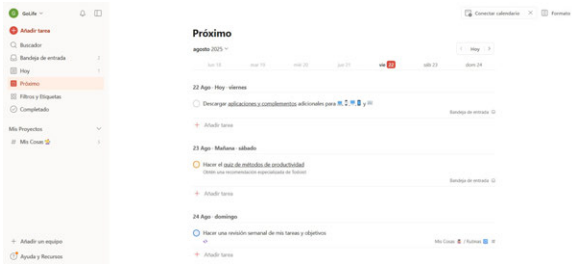
Puntos fuertes:

- **Captura y planificación muy rápidas** con lenguaje natural para fechas y recurrencias, vistas "Hoy/Próximo" claras.
- **Interfaz limpia y jerárquica** con arrastrar/soltar y prioridades P1–P4 bien diferenciadas, lo que facilita reorganizar y enfocar.

Puntos débiles:

- **Modelado limitado** para hábitos y objetivos complejos, se apoya en tareas recurrentes y etiquetas; métricas de progreso poco profundas.
- **Analítica y feedback limitados** dificultan evaluar avances por objetivo/hábito sin montar filtros manuales.

(a) Todoist - Página de presentación(8)



(b) Todoist - Ejemplo de uso(8)

Figura 2.2: Todoist - Vistas

2.2.3. Habitica

Qué es y para quién: Habitica (9) es un gestor de tareas y hábitos con una capa de gamificación estilo RPG. Pensado para quienes se motivan con recompensas, retos y juego social (avatar, equipo, misiones en grupo).

Dominios que cubre: Tareas ✔ Hábitos ✔ Objetivos ✔ Finanzas simples ✘

Plataformas / Disponibilidad: Web ✔ Móvil (iOS/Android) ✔ Escritorio (Windows/macOS) ✘

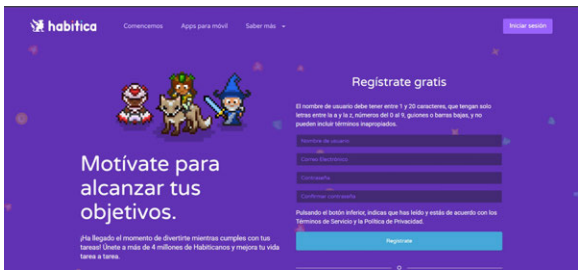
Funcionamiento base: El usuario crea hábitos, tareas recurrentes diarias y pendientes. Completar acciones otorga experiencia y oro; fallar diarias resta salud. Puedes gastar recompensas en equipo y participar en parties/retos para responsabilidad compartida. Vease la Figura 2.3.

Puntos fuertes:

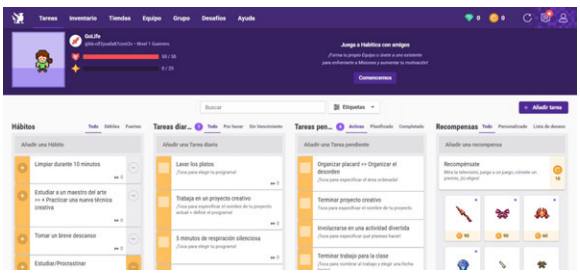
- **Motivación por juego** (XP, oro, rachas, equipo, misiones) que refuerza la constancia y el refuerzo positivo/negativo de hábitos.
- **Tipología clara de tareas** (Hábitos +/−, Diarias, Pendientes) que separa rutina, mantenimiento y objetivos puntuales.

Puntos débiles:

- **Gamificación extrema** puede abrumar; no se entiende bien cómo usar la capa lúdica si solo se busca un seguimiento simple.
- **Analítica limitada**, carece de métricas flexibles para objetivos numéricos.



(a) Habitica - Página de presentación(9)



(b) Habitica - Ejemplo de uso(9)

Figura 2.3: Habitica - Vistas

2.2.4. Any.do

Qué es y para quién: Any.do (10) es un gestor de tareas con enfoque en la planificación diaria ("Plan my day") y vista calendario. Orientado a quienes buscan simplicidad para capturar pendientes, programarlos y revisarlos en un flujo rápido.

Dominios que cubre: Tareas ✔ Hábitos ✔ Objetivos ✔ Finanzas simples ✘

Plataformas / Disponibilidad: Web ✔ Móvil (iOS/Android) ✔ Escritorio (Windows/macOS) ✔

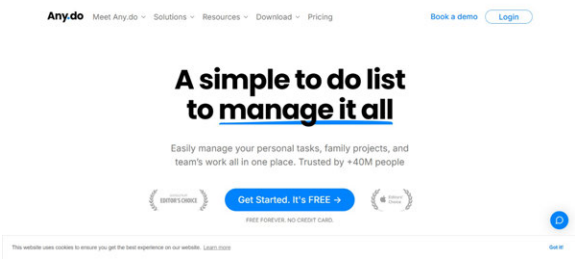
Funcionamiento base: El usuario crea listas y tareas con fecha/hora (soporta lenguaje natural), recordatorios y recurrencias; puedes añadir subtareas y notas, reprogramar por arrastrar/soltar y ver todo en "Hoy" o calendario. Permite colaboración compartiendo listas y asignando tareas. Vease la Figura 2.4.

Puntos fuertes:

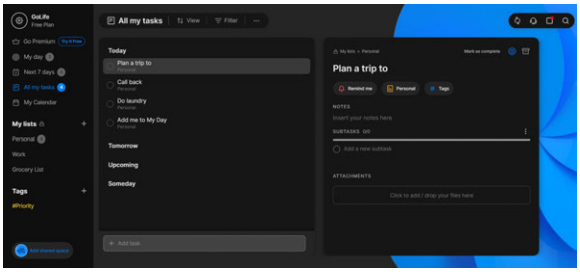
- **Planificación diaria muy clara** con "Plan my day" y vista calendario integrada que facilita priorizar y reprogramar.
- **Interfaz minimalista y directa** (listas, tareas, subtareas) que reduce fricción y favorece el enfoque en lo esencial.

Puntos débiles:

- **Modelado limitado** para objetivos/hábitos avanzados: se apoya en tareas recurrentes y carece de campos personalizados y vistas tipo tablero.
- **Analítica de progreso básica** (sin paneles configurables), lo que dificulta evaluar tendencias sin montar filtros externos.



(a) Any.do - Página de presentación(10)



(b) Any.do - Ejemplo de uso(10)

Figura 2.4: Any.do - Vistas

2.2.5. Comparativa resumida

Herramientas	Dominios	Puntos fuertes	Puntos débiles
Notion	Tareas	Flexibilidad elevada	Alta curva de aprendizaje
	Hábitos	Plantillas reutilizables	Progreso no estandarizado
	Objetivos		
	Finanzas simples		
Todoist	Tareas	Captura y planificación rápidas	Modelado limitado
	Hábitos	Interfaz limpia y jerárquica	Analítica y feedback limitados
	Objetivos		
Habitica	Tareas	Motivación por juego	Gamificación extrema
	Hábitos	Tipología clara de tareas	Analítica limitada
	Objetivos		
Any.do	Tareas	Planificación diaria clara	Modelado limitado
	Hábitos	Interfaz minimalista y directa	Analítica de progreso básica
	Objetivos		

Tabla 2.1: Comparativa resumida de herramientas

2.3. Modelos de Servicios Cloud

Un **modelo de servicio cloud** es la abstracción que define qué capacidades ofrece el proveedor (infraestructura, plataforma o software) y hasta dónde llega su responsabilidad operativa frente a la del equipo que desarrolla y explota la aplicación. En la práctica, delimita el perímetro de gestión (configuración, parches de SO, middleware y runtime,

datos, seguridad y continuidad de negocio) y, con ello, los grados de control, automatización y coste operativo asociados.

Revisaremos primero los **tres modelos principales**: **IaaS** (Infraestructura como Servicio), **PaaS** (Plataforma como Servicio) y **SaaS** (Software como Servicio), situando sus ventajas e inconvenientes en términos de gestión, personalización y costes. Además, por la naturaleza del proyecto, incorporamos dos modelos gestionados de especial interés: **BaaS** (Back end como Servicio), , y **DBaaS** (Base de Datos como Servicio).

Cada servicio materializa un nivel de abstracción distinto: si ofrece máquinas virtuales y redes hablamos de **IaaS**; si proporciona una plataforma gestionada para desplegar código, de **PaaS**; si entrega una aplicación lista para usar, de **SaaS**; si expone funciones típicas de back end (p. ej., autenticación o notificaciones), de **BaaS**; y si administra una base de datos como producto gestionado, de **DBaaS**. Así, los modelos de servicio son categorías que describen dónde se sitúa la frontera de responsabilidades en cada servicio. Estas fronteras pueden verse detalladas en la Figura 2.5. En la práctica, un proyecto real suele combinar varios de ellos (p. ej., una API sobre PaaS, autenticación vía BaaS y persistencia en DBaaS) manteniendo coherente ese reparto.

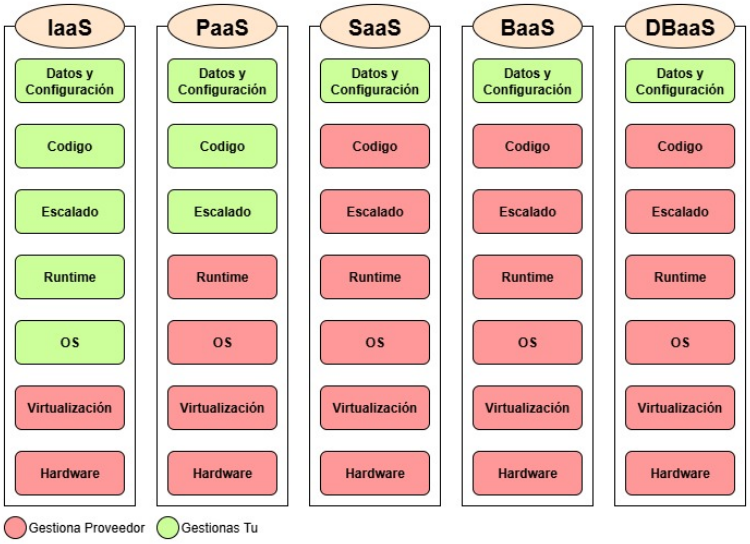


Figura 2.5: Responsabilidades por modelo de servicio en la nube

2.3.1. IaaS (Infraestructura como Servicio)

IaaS es un modelo de servicio cloud que ofrece recursos de infraestructura a demanda (cómputo, almacenamiento, redes y virtualización) operados por el proveedor en la nube. El cliente conserva la responsabilidad sobre el sistema operativo, el middleware/entorno de ejecución, las aplicaciones y los datos, mientras delega el hardware y su operación diaria (centro de datos, física, red, hipervisor). (11; 12). Pueden verse ejemplos en la Figura 2.6.

Qué ofrece:

- Instancias de máquina virtual, redes virtuales (VPC), balanceadores de carga, firewalls y almacenamiento (bloques/objetos).

- Escalado bajo demanda y facturación por uso, con APIs/console para automatizar aprovisionamiento.

Ventajas:

- **Alto control** sobre la pila software (SO, runtime, apps y datos).
- **Escalabilidad a demanda** y reducción de gasto de capital (pago por uso); menos demoras de aprovisionamiento.
- **Fiabilidad** al evitar puntos únicos de fallo mediante recursos gestionados del proveedor.

Inconvenientes:

- Mayor **carga operativa**: configuración, parches y mantenimiento del SO y del stack software corren a cargo del equipo.
- **Seguridad y recuperación** de datos bajo responsabilidad del cliente; protección de aplicaciones heredadas puede ser compleja.
- Posible **complejidad de administración** y dependencia del proveedor si no se diseña con portabilidad en mente.

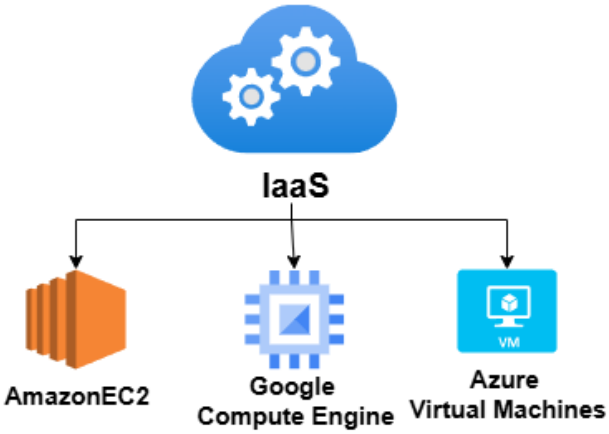


Figura 2.6: Ejemplos de IaaS

2.3.2. PaaS (Plataforma como Servicio)

PaaS es un modelo de servicio cloud que proporciona una plataforma gestionada para desarrollar, ejecutar y desplegar aplicaciones sin administrar la infraestructura subyacente. El proveedor opera el hardware, la red, la virtualización, el sistema operativo y gran parte del runtime/middleware; el equipo se centra en el código y los datos de la aplicación. (11; 12). Pueden verse ejemplos en la Figura 2.7.

Qué ofrece:

- Entornos de ejecución y despliegue gestionados (runtimes, servidores de aplicaciones o contenedores), con integración nativa o por add-ons a servicios gestionados típicos (p. ej., bases de datos, colas, caché).

- Autoescalado, balanceo, monitorización y seguridad integrados, con configuración por consola y/o API, sin ocuparse de SO ni middleware.

Ventajas:

- **Time-to-market** acelerado y mayor productividad al abstraer infraestructura y automatizar despliegues.
- **Menor carga operativa**: parches, alta disponibilidad y escalado gestionados por el proveedor.
- **Estandarización del stack** con buenas prácticas por defecto (registro, métricas, seguridad).

Inconvenientes:

- **Menor control** de SO/red/runtime y límites de personalización e integración.
- Riesgo de **dependencia del proveedor** (servicios propietarios, portabilidad limitada) y costes que crecen con el uso.
- **Restricciones de lenguajes/versiones/servicios** soportados y posibles dificultades con aplicaciones legadas.

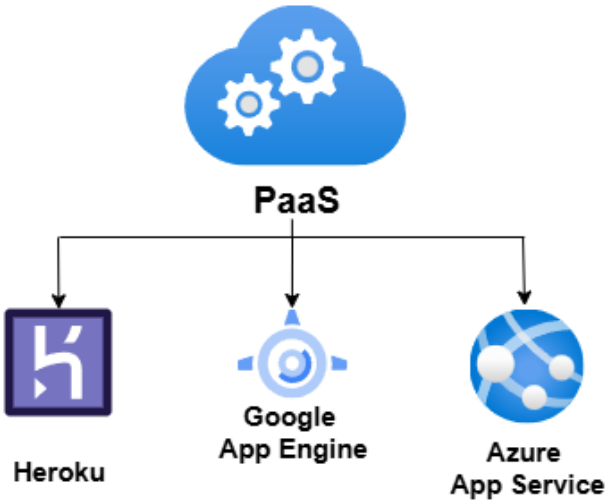


Figura 2.7: Ejemplos de PaaS

2.3.3. SaaS (Software como Servicio)

SaaS es un modelo de servicio cloud en el que el proveedor entrega una aplicación completa accesible por Internet (normalmente vía navegador o app), y se encarga de operar toda la pila: infraestructura, plataforma, runtime y la propia aplicación. El cliente utiliza el software "como servicio", gestionando solo la configuración funcional y sus datos. (11; 12). Pueden verse ejemplos en la Figura 2.8.

Qué ofrece:

- Acceso inmediato a la aplicación (sin instalación local), con actualizaciones y parches automáticos.
- Modelo habitual de suscripción y escalado gestionado por el proveedor (multi-tenant).

Ventajas:

- **Cero mantenimiento técnico** por parte del cliente: operación, seguridad y evolución corren a cargo del proveedor.
- **Puesta en marcha rápida** y coste predecible (suscripción), ideal para adopciones ágiles.
- **Accesibilidad** desde múltiples dispositivos y ubicaciones, con alta disponibilidad gestionada.

Inconvenientes:

- **Menor personalización e integración fina** que en IaaS/PaaS; ajustes limitados al producto.
- **Dependencia del proveedor** y posibles retos de portabilidad de datos (vendor lock-in).
- Consideraciones de **cumplimiento y privacidad** (ubicación y tratamiento de datos) y dependencia de la conectividad.

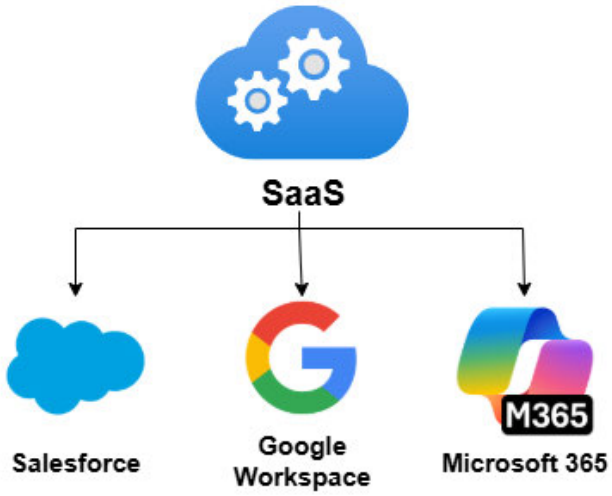


Figura 2.8: Ejemplos de SaaS

2.3.4. BaaS (Back end como Servicio)

BaaS es un modelo de servicio cloud que proporciona funcionalidades típicas de back end como servicios gestionados (p. ej., autenticación, bases de datos, almacenamiento de archivos, notificaciones), accesibles mediante SDKs/APIs. El proveedor opera la infraestructura y el back end del servicio; el equipo se centra en la lógica de su aplicación y en la configuración de datos/reglas, sin administrar servidores propios. (13; 14). Pueden verse ejemplos en la Figura 2.9.

Qué ofrece:

- SDKs/APIs para autenticación de usuarios, persistencia de datos, almacenamiento de archivos y mensajería/push, con consola de gestión y reglas de seguridad.
- Escalado automático, alta disponibilidad y operaciones (backups, parches) gestionadas por el proveedor.

Ventajas:

- **Time-to-market** rápido: reduce la necesidad de programar y operar un back end propio.
- **Menor carga operativa** y costes iniciales: el proveedor asume infraestructura y mantenimiento.
- **Escalabilidad integrada** y servicios listos para integrar en apps web/móviles.

Inconvenientes:

- **Dependencia del proveedor** (vendor lock-in) y portabilidad limitada entre plataformas.
- **Menos control/personalización** del back end (lógica avanzada, rendimiento fino, integraciones no estándar).
- Consideraciones de **privacidad, cumplimiento y localización de datos** según políticas del servicio.

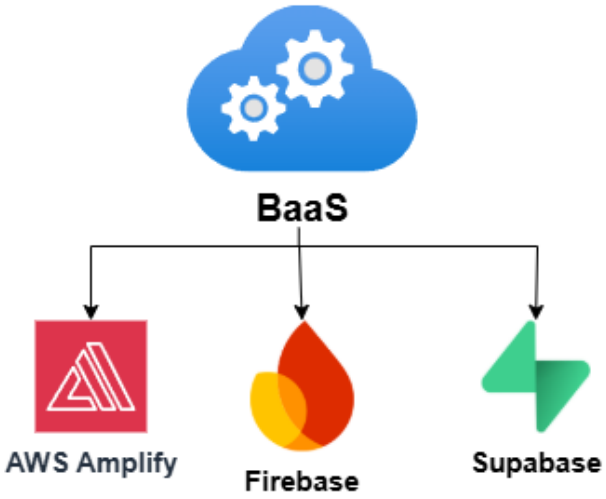


Figura 2.9: Ejemplos de BaaS

2.3.5. DBaaS (Base de Datos como Servicio)

DBaaS es un modelo de servicio cloud en el que el proveedor administra el motor de base de datos como un producto gestionado: aprovisiona instancias/clústeres, aplica parches, gestiona copias de seguridad, alta disponibilidad y escalado; el equipo consumidor se centra en el esquema, las consultas y los datos. (15; 16). Pueden verse ejemplos en la Figura 2.10.

Qué ofrece:

- Aprovisionamiento y operación gestionada del motor (parches, actualizaciones), con consola/API para despliegue y configuración.

- Alta disponibilidad y replicación, copias de seguridad automáticas y opciones de escalado (vertical/horizontal según servicio).
- Funciones de seguridad integrables (cifrado en tránsito/en reposo, control de acceso) y monitorización del rendimiento.

Ventajas:

- **Menor carga operativa**: el proveedor se encarga del motor, la resiliencia y el mantenimiento, acelerando la puesta en marcha.
- **Escalabilidad y disponibilidad** incorporadas (réplicas, particionado/sharding, backups gestionados).
- **Seguridad y cumplimiento** facilitados por capacidades gestionadas y configurables del servicio.

Inconvenientes:

- **Menos control fino** sobre la configuración avanzada del motor y extensiones específicas frente a soluciones autogestionadas.
- **Dependencia del proveedor** (portabilidad y formatos/operaciones), y costes que pueden crecer con almacenamiento, I/O y transferencia.
- Consideraciones de **latencia y localización de datos** (regulatorio/privacidad) según regiones y arquitectura.

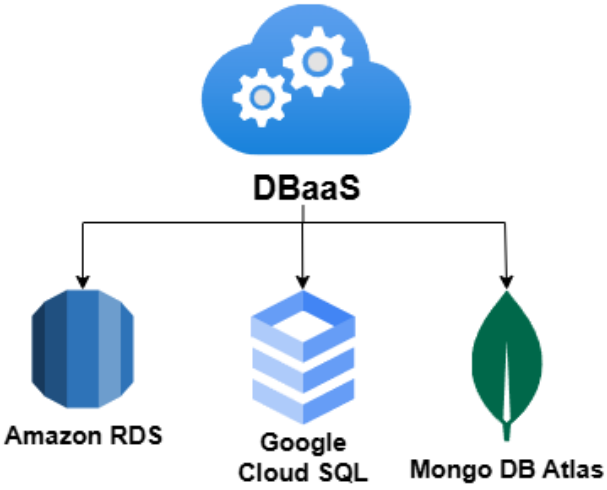


Figura 2.10: Ejemplos de DBaaS

2.4. Tecnologías Utilizadas

A continuacion se mencionan todas las tecnologías utilizadas en este proyecto:

2.4.1. Servicios Cloud

- **Google App Engine(17):** PaaS gestionado para desplegar y escalar aplicaciones web sin administrar servidores, con autoescalado, balanceo y despliegue integrados.
- **Google Cloud Key Management Service (KMS)(18):** Servicio gestionado de gestión de claves que permite crear, rotar y usar claves para cifrado/descifrado y firma, con control de acceso centralizado e integración con GCP.
- **Firebase Hosting (19):** Alojamiento web sobre CDN global con HTTPS automático, dominios personalizados y despliegues versionados; idóneo para SPA y sitios estáticos.
- **Firebase Authentication (20):** Autenticación lista para usar con email/contraseña, teléfono y proveedores federados (p. ej., Google), gestionando tokens y flujos de inicio de sesión.
- **MongoDB Atlas (21):** DBaaS totalmente gestionado para MongoDB que aprovisiona, escala y opera clústeres con alta disponibilidad, copias de seguridad y seguridad integrada.

2.4.2. Desarrollo y Despliegue

- **KanbanFlow (22):** Herramienta Kanban en línea para gestionar tareas por columnas, con soporte de WIP, swimlanes, temporizador Pomodoro y analíticas básicas; facilita la visualización del flujo y la colaboración en tableros compartidos.
- **IntelliJ IDEA (23):** IDE para Java/Kotlin con refactorización inteligente, autocompletado, depuración integrada e integración con Maven/Gradle, Git y frameworks como Spring Boot.
- **GitHub (24):** Plataforma de hospedaje de repositorios Git con control de versiones, issues, pull requests y proyectos; integra CI/CD mediante Actions, revisión de código y wikis para colaboración de equipos.
- **GitHub Actions (25):** Plataforma de automatización que permite crear flujos de trabajo (workflows) para compilar, probar y desplegar código directamente desde los repositorios de GitHub.
- **MongoDB Compass (26):** Cliente gráfico oficial para MongoDB que permite explorar colecciones y esquemas, construir consultas y agregaciones, visualizar rendimiento e índices, y ejecutar operaciones de datos de forma segura.
- **Docker Desktop (27):** Aplicación para macOS/Windows que facilita construir, ejecutar y compartir aplicaciones en contenedores; incluye Docker Engine y Docker Compose, herramientas de redes/volúmenes e integración con IDEs.

2.4.3. Front end

2.4.3.1. Lenguajes

- **JavaScript (ES2015+ y JSX) (28; 29; 30):** Lenguaje estándar ECMAScript (desde ES2015) y sintaxis JSX para describir interfaces de usuario en React.
- **HTML5 y CSS3 (31; 32):** Estándares web para estructura y presentación; HTML como "Living Standard" y CSS definido por el *CSS Snapshot* vigente.

2.4.3.2. Frameworks

- **React 18.2.0 (33; 30; 34):** Biblioteca para construcción de interfaces basada en componentes y JSX.
- **React Router 7.6.2 (35; 36; 37):** Enrutador para SPAs con soporte de datos y estrategias de navegación modernas.
- **Vitest 2.1.9 (38; 39):** Framework de pruebas (runner + aserciones) nativo de Vite, compatible con la API de Jest.
- **Create React App / react-scripts 5.0.1 (40; 41; 42; 43):** *Boilerplate* histórico para SPAs React; `react-scripts` agrupa el toolchain (build, dev, test).

2.4.3.3. SDKs

- **Firebase Web SDK (versión modular) (44):** SDK de JavaScript modular. Permite importaciones granulares (p. ej., `import { getAuth } from "firebase/auth"`), reduciendo el tamaño del bundle y mejorando tiempos de carga.

2.4.3.4. Librerias

Core:

- **Axios 1.10.0 (45; 46):** Cliente HTTP *promise-based* para navegador y Node.js; interceptores y serialización automática de JSON/form.
- **Driver.js 1.3.6 (47; 48):** Recorridos/tutoriales in–app para guiar al usuario (destacados, popovers, pasos).
- **web-vitals 2.1.4 (49; 50):** Medición en campo de Core Web Vitals (LCP, INP, CLS) con API ligera.

Testing:

- **@testing-library/react 16.3.0 (51):** Utilidades para testear componentes desde la perspectiva del usuario.
- **@testing-library/jest-dom 6.6.3 (52):** *Matchers* adicionales para el DOM en Jest/Vitest.
- **@testing-library/user-event 13.5.0 (53):** Simulación de interacciones realistas del usuario.
- **jsdom 24.0.0 (54):** Implementación del DOM/HTML de WHATWG para entorno Node (entorno de pruebas).

2.4.3.5. Plugins (Node.js)

- **react-scripts 5.0.1 (41; 42):** Scripts y configuración de CRA (Webpack 5, dev server, `build`, `test`).
- **@vitest/coverage-v8 2.1.9 (55; 56):** Proveedor de cobertura nativa V8 para Vitest.

2.4.4. Back end

2.4.4.1. Lenguajes

- **Java 17 (LTS) (57; 58):** Lenguaje de programación (Java SE 17) con soporte a largo plazo; documentación oficial y JDK de referencia disponibles en Oracle y OpenJDK.

2.4.4.2. Frameworks

- **Spring Boot 3.5 (59):** Framework Java que acelera el desarrollo de APIs y microservicios mediante autoconfiguración, starters y servidor embebido. Uso de starters: Web, Validation y Security.

2.4.4.3. SDKs

- **Google Cloud KMS (2.74.0) (18):** Cliente oficial de Java para invocar Key Management Service (cifrado/descifrado, firma/verificación) y gestionar claves y key rings con control de acceso e incluso rotación programada.
  Artefacto: `com.google.cloud:google-cloud-kms:2.74.0` (60).
- **Firebase Admin SDK (9.5.0) (61):** SDK del lado servidor para verificar/emitir tokens de Firebase Authentication, administrar usuarios y acceder a servicios de Firebase desde entornos de confianza.
  Artefacto: `com.google.firebase:firebase-admin:9.5.0` (62).

2.4.4.4. Librerias

Core:

- **Bucket4j (8.14.0) (63):** Limitación de tasa en memoria basada en el algoritmo *token bucket*; permite cuotas y protección de endpoints HTTP.
- **Caffeine (3.1.8) (64):** Caché local de alto rendimiento con políticas de expiración/evicción, tamaños máximos y métricas.
- **MongoDB Java Driver Sync (65):** Driver oficial síncrono para acceso nativo a MongoDB; versión gestionada vía BOM.
- **springdoc-openapi (2.8.9) (66):** Genera especificaciones OpenAPI 3 y expone Swagger UI a partir de controladores Spring de forma automática.

Testing:

- **Spring Boot Starter Test (67):** Starter de pruebas que integra JUnit 5, Mockito, AssertJ y utilidades de Spring Test para contextos, MVC y JSON.
- **Spring Security Test (68):** Extensiones para probar seguridad: SecurityMockMvc, anotaciones como `@WithMockUser` y soporte de autenticación/autoridades simuladas.
- **Testcontainers 1.21.3 (69):** Orquestación de contenedores efímeros (Docker) en tests; módulos core, mongodb y junit-jupiter con integración JUnit 5.
- **System Lambda 1.2.1 (70):** Utilidades para tests que capturan `System.out/err`, establecen variables de entorno y propiedades del sistema de forma aislada.

2.4.4.5. Plugins (Maven)

- **Spring Boot Maven Plugin (71):** Empaqueta y ejecuta aplicaciones Spring Boot (repackage, run) e integra propiedades/recursos para despliegue.
- **Maven Surefire Plugin 3.0.0-M9 (72):** Ejecuta tests unitarios (JUnit/Jupiter, TestNG) en la fase `test`.
- **Maven Failsafe Plugin 3.0.0-M9 (73):** Ejecuta pruebas de integración en las fases `integration-test`/`verify`.
- **JaCoCo Maven Plugin 0.8.12 (74):** Genera informes de cobertura (HTML/XML) y permite umbrales de verificación.
- **Maven Resources Plugin 3.1.0 (75):** Copia y filtra recursos (por ejemplo, `src/main/resources`) durante el build.

2.4.5. Herramientas para diagramas y documentación

- **Overleaf (76):** Editor LaTeX colaborativo en la nube con compilación directa, control de versiones y plantillas; facilita la coautoría y el formateo académico sin instalaciones locales.
- **diagrams.net (draw.io) (77):** Herramienta de diagramación en navegador para UML, ER, BPMN, flujos y arquitecturas; exporta a PNG/SVG/PDF.
- **StarUML (78):** Editor de modelado UML multiplataforma orientado a crear diagramas como casos de uso, clases, secuencia, actividad y más. Ofrece exportación (PNG/SVG/PDF), plantillas y un sistema de extensiones para ampliar funcionalidades.
- **Online Gantt (79):** Editor de diagramas de Gantt online para planificar cronogramas con tareas, dependencias e hitos; útil para visualizar fases y estimaciones.
- **Figma (80):** plataforma colaborativa de diseño UI/UX basada en la web; permite edición simultánea, prototipado interactivo y componentes reutilizables.
- **Swagger UI (81):** Genera documentación interactiva a partir de especificaciones OpenAPI; permite explorar endpoints y probar peticiones directamente en el navegador.
- **OpenAPI Initiative (82):** Especificación estándar, independiente del lenguaje, para describir APIs HTTP; base para generación de clientes/servidores y documentación consistente.
- **GitHub Pages (83):** Alojamiento estático integrado en GitHub para publicar sitios desde repositorios; admite HTTPS, dominios personalizados y Jekyll, ideal para documentación y portafolios.

Capítulo 3

Visión general de producto

En esta sección presentamos la visión general de GoLife, siguiendo la estructura de un **Inception deck**. Cada apartado define un aspecto clave de nuestro producto. desde la motivación inicial, hasta los costes estimados para alinear objetivos y clarificar el alcance.

3.1. ¿Por qué estamos aquí?

No se puede crear un gran producto si, en primer lugar, no sabemos con claridad por qué lo estamos desarrollando. Plantearnos esta pregunta nos da, como equipo, el contexto necesario para tomar decisiones coherentes y alineadas con los objetivos reales.

En nuestro caso, queremos ofrecer una herramienta genérica que cualquier persona pueda usar sin necesidad de formación previa. Al entender que el valor principal de GoLife es su sencillez de uso y la claridad en la visualización de metas, podemos priorizar una interfaz intuitiva, minimizar la curva de aprendizaje y garantizar que el usuario comience a beneficiarse desde el primer acceso.

Al mismo tiempo, pretendemos implementar y experimentar con plataformas y sistemas en la nube, para aprovechar su escalabilidad, alta disponibilidad y despliegue continuo. Conociendo que uno de los beneficios esperados es reducir costes operativos y acelerar el tiempo de lanzamiento, podemos negociar y diseñar la arquitectura adecuada sin perder de vista el propósito: facilitar el registro y seguimiento de metas personales de forma inmediata y sin complicaciones.

3.2. Elevator Pitch

Un **elevator pitch** sintetiza en menos de dos minutos la propuesta de valor de nuestro proyecto, despertando el interés suficiente para querer saber más. A continuación presentamos la esencia de GoLife, definida en términos claros y directos para destacar sus ventajas frente a otras soluciones.

Para: Cualquier usuario que quiera cuantificar y dar seguimiento a sus metas y objetivos personales.

Quién(es): Personas interesadas en monitorizar su progreso de manera sencilla y motivadora.

El producto: GoLife.

Es una: plataforma web.

Que: Permite a los usuarios introducir sus metas y objetivos, registrar actualizaciones periódicas y visualizarlas en un panel de control.

A diferencia de: Otras apps más complejas, GoLife ofrece configuración mínima y un flujo de uso inmediato.

Nuestra propuesta de valor: Facilitar el registro, la visualización y el seguimiento de las metas personales sin abrumar al usuario con opciones avanzadas.

3.3. Caja del Producto

Diseñar una "caja de producto" es un ejercicio de marketing y creatividad que nos sitúa en la mente del cliente. Imagina GoLife presentado en un estante: el diseño del envase debe comunicar de un vistazo los beneficios clave, captar la atención y generar confianza.

A continuación se muestran los puntos por los que GoLife destaca, acompañados de sus mensajes principales:

- **Sencillez instantánea:** Arranca con un clic y, si lo deseas, sigue un breve tour opcional que puedes omitir en cualquier momento.
- **Panel de progreso motivador.** Visualiza tu avance con gráficos claros y dinámicos que impulsan la constancia.
- **Configuración mínima.** Olvídate de opciones complejas; sólo define tus metas y empieza a trabajar.
- **Acceso en todas partes:** Disponible en navegador, sincronizado en la nube al instante.

3.4. Lista de NOes

Antes de arrancar con el desarrollo, en la Tabla 3.1 definimos claramente qué queda fuera de nuestro alcance, qué puntos deben decidirse más adelante y qué aspectos sí abordaremos en esta fase inicial. Esto nos ayuda a focalizar esfuerzos y a gestionar expectativas.

DENTRO DEL ALCANCE	FUERA DEL ALCANCE
▪ Registro y gestión de metas personalizadas ▪ Panel de control (dashboard) básico ▪ Tutorial interactivo opcional (saltable) ▪ Autenticación y gestión de usuarios ▪ Despliegue íntegro en la nube	▪ Integración / Interacción con redes sociales ▪ Aplicación móvil ▪ Informe de datos exportable (CSV/PDF) ▪ Módulos de cooperación entre usuarios
SIN DECIDIR	
▪ Funcionalidades de gamificación simple ▪ Notificaciones push en navegador	

Tabla 3.1: Lista de NOes

3.4.1. Dentro del Alcance

- **Registro y gestión de metas personalizadas:** Los usuarios podrán crear, editar, eliminar metas. Así como crear y eliminar registros de estas.
- **Panel de control (dashboard) básico:** Vista central con gráficas sencillas e indicadores claros del progreso de las metas.
- **Tutorial interactivo opcional (saltable):** Tour guiado al primer acceso que el usuario puede omitir en cualquier momento.
- **Autenticación y gestión de usuarios:** Registro, login con Google y recuperación de contraseña.
- **Despliegue íntegro en la nube:** La aplicación se aloja completamente en plataformas cloud.

3.4.2. Fuera del Alcance

- **Integración/interacción con redes sociales:** No se incluirá conexión con APIs de Facebook, X (Twitter), Instagram u otras.
- **Aplicación móvil:** GoLife funcionará únicamente en navegadores web modernos; no habrá app nativa.
- **Informe de datos exportable (CSV/PDF):** La descarga de informes queda para fases posteriores.

- **Módulos de cooperación entre usuarios:** Se descartan retos grupales, compartir metas o foros internos en esta versión.

3.4.3. Sin Decidir

- **Funcionalidades de gamificación simple:** Puntos, niveles, recompensas o mensajes básicos que impulsen la motivación sin sobrecargar al usuario.
- **Notificaciones push en navegador:** Mensajes push cuando el usuario no está activo en la pestaña.

3.5. Conoce a tus Vecinos

Antes de desarrollar GoLife es fundamental identificar a todas las personas y equipos que, directa o indirectamente, conocen, afectan y/o muestran interés por el proyecto. Esto facilita la comunicación, la recogida de requisitos y la adopción final.

Para representar e identificar a estos **stakeholders**, vamos a utilizar un diagrama de cebolla (Figura 3.1) con las siguientes capas:

- **Internos:** Grupo directamente involucrado en el desarrollo y ejecución diaria del proyecto; toman decisiones operativas y reciben rápidamente el valor generado.
- **Institucionales:** Entidades que avalan y supervisan el proyecto desde un punto de vista académico y normativo; no intervienen en los detalles técnicos, pero su aprobación y cumplimiento de sus procesos son imprescindibles.
- **Externos:** Las personas que utilizarán la aplicación en su día a día; su satisfacción, feedback y patrones de uso guían las mejoras y la evolución del producto, aunque no formen parte del equipo de desarrollo.
- **Secundarios:** Servicios y plataformas externas que no interaccionan mano a mano con el equipo, pero cuyas características, coste, rendimiento, escalabilidad y disponibilidad, condicionan la viabilidad técnica y económica del despliegue.

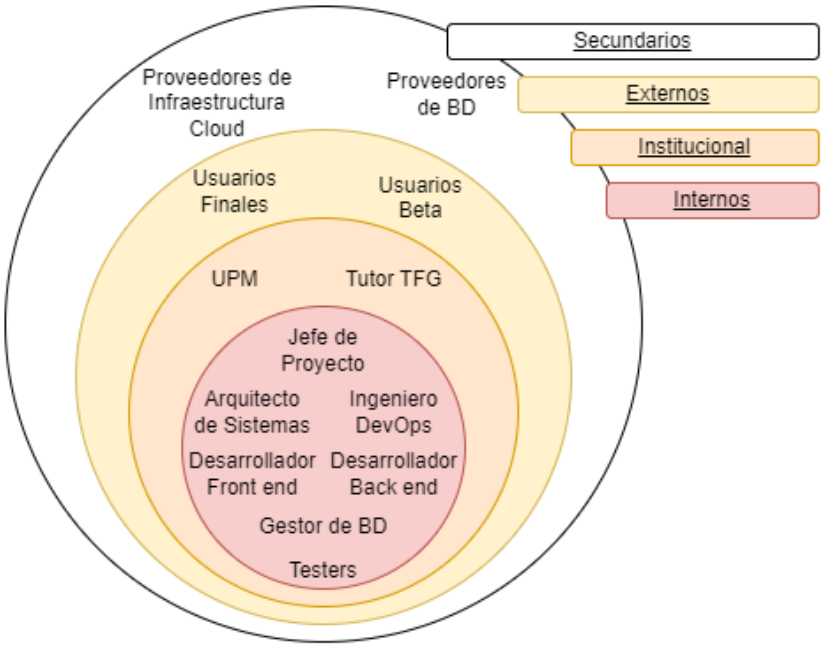


Figura 3.1: Diagrama de Cebolla

3.6. Haz Ver la Solución

En este apartado mostramos de manera clara la arquitectura y las herramientas principales con las que abordaremos GoLife, para asegurar que todo el equipo y los stakeholders conocen el enfoque técnico desde el inicio.

Presentamos en la Figura 3.2 un diagrama de alto nivel que ilustra los componentes principales del sistema y sus relaciones.

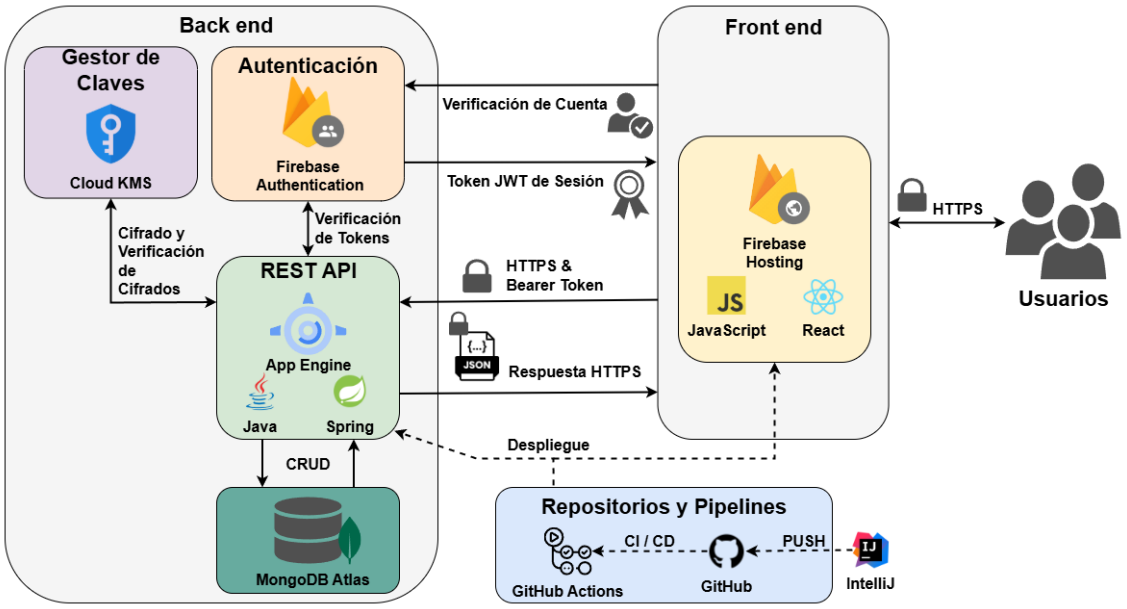


Figura 3.2: Solución de GoLife

A continuación se detallan brevemente los bloques principales de la solución:

- **Front end (React en Firebase Hosting)** Interfaz de usuario accesible desde cualquier navegador que consume la API REST y muestra el dashboard de progreso.
- **Autenticación (Firebase Authentication)** Servicio gestionado que verifica credenciales, emite y valida tokens JWT para el inicio de sesión y acceso a la API.
- **REST API (Spring en App Engine)** Punto de entrada HTTPS que recibe peticiones con Bearer Token, implementa la lógica de negocio y realiza operaciones CRUD sobre la base de datos.
- **Gestor de Claves (Cloud KMS)** Servicio REST API que cifra y verifica los campos identificativos antes de guardar en la base de datos y al leerlos; KMS custodia las claves y gestiona su rotación fuera de la aplicación.
- **Base de Datos (MongoDB Atlas)** Almacena de forma escalable documentos con la información de usuarios y metas personales.
- **Repositorios y Pipelines (GitHub & GitHub Actions)** Control de versiones en GitHub e integración continua con GitHub Actions para ejecutar tests y desplegar automáticamente en el entorno cloud.

3.7. ¿Qué nos quita el sueño?

Identificar los riesgos sobre los que podemos influir y aquellos que escapan a nuestro control nos permite focalizar esfuerzos y ser conscientes de las amenazas reales al éxito de GoLife. Vease la Tabla 3.2:

Riesgos controlables	Riesgos no controlables
▪ Faltas de sincronización en el equipo de desarrollo. ▪ Curva de aprendizaje de React y Spring superior a la prevista. ▪ Escasez de casos de prueba o entornos de test automatizados. ▪ Problemas de integración CI/CD en GitHub Actions. ▪ Problemas de comunicación entre componentes. ▪ No adopción de la aplicación por parte de los usuarios.	▪ Caídas o latencia e incidencias en servicios cloud. ▪ Cambios en coste o facturación de los servicios cloud (planes gratuitos o tarifas). ▪ Modificaciones regulatorias sobre protección de datos que afecten al tratamiento de información personal. ▪ Problemas de conectividad de usuarios finales (ancho de banda, bloqueos de red).

Tabla 3.2: Riesgos controlables vs no controlables

3.8. Tomar las Medidas

Antes de comenzar el desarrollo profundo, presentamos una estimación a alto nivel de la duración y el esfuerzo de cada fase. Estas cifras no son compromisos, sino orientaciones para validar la viabilidad con los recursos disponibles y ajustar exceptativas.

- **Planificación y Diseño (1,5 meses)** Cierre de requisitos, diagramas de arquitectura, esquemas de base de datos y wireframes.
- **Configuración de CI/CD y Servicios Cloud (0,5 meses)** Despliegue y configuracion inicial de entornos en la nube y definición de pipelines en GitHub Actions para integración y tests automáticos.
- **Desarrollo del Back end y Base de Datos (1 mes)** Implementación de la API REST y los modelos de datos en MongoDB Atlas, con cobertura mediante pruebas unitarias y de integracion.
- **Desarrollo del Front end (1 mes)** Construcción de la interfaz en React e integración con la API, validado en cada ejecución del pipeline.
- **Pruebas e Integración Final (0,5 meses)** Pruebas de usabilidad con usuarios y pruebas generales para validar el cumplimiento de requisitos.
- **Margen de Contingencia (0,5 meses)** Reserva para imprevistos, revisión de feedback del tutor y corrección de bugs críticos.

Total estimado: 5 meses (± 0,5 meses).

Esta planificación se ha elaborado partiendo de la experiencia en proyectos académicos de alcance similar, ajustando los tiempos para reflejar la dedicación parcial propia de la vida de estudiante y considerando el esfuerzo adicional que supone aprender nuevas herramientas y componentes durante el desarrollo.

3.9. Que vamos a dar

Para visualizar mejor el grado de flexibilidad de cada palanca del proyecto, utilizamos la siguiente escala (Tabla 3.3):

Nivel	**Definición**
Muy flexible	Puede modificarse o posponerse con impacto mínimo en el proyecto.
Flexible	Admite ajustes importantes, pero requiere planificación para no desviar el alcance.
Ajustable	Se puede recortar o aplazar con revisión de prioridades y posible desplazamiento de fechas.
No negociable	Umbral mínimo que debe cumplirse; no es posible bajar de este nivel sin comprometer el objetivo.
Nulo	Elemento cuya ausencia detendría por completo el avance del proyecto.

Tabla 3.3: Niveles de flexibilidad de las palancas del proyecto

A continuación, cada factor se sitúa en esta escala según su nivel de rigidez (Tabla 3.4):

Factor	Nivel de flexibilidad
Plazo de entrega	Flexible (se cierra antes de fin de año)
Presupuesto	Muy flexible (créditos académicos y free tier)
Alcance	Ajustable (funcionalidades secundarias recortables)
Calidad mínima	No Negociable (estable, cumple requisitos básicos)
Facilidad y simplicidad	Nulo (experiencia intuitiva desde el inicio)
Seguridad	Nulo (autenticación robusta y cifrado)
Rendimiento	Ajustable a objetivo (<500 ms)

Tabla 3.4: Palancas del proyecto y su nivel de flexibilidad

Con esta visión clara de qué factores no podemos cambiar y cuáles podemos flexibilizar, el equipo sabe dónde concentrar esfuerzos cuando surjan imprevistos.

3.10. ¿Cuál va a ser el coste?

Para ofrecer una estimación aproximada del coste del proyecto, hemos considerado lo siguiente:

- Dos desarrolladores a media jornada (50 %): salario bruto anual medio de 31 000 €(de acuerdo a Glassdoor (84)) para ingenieros de software en España → 1 292 €/mes cada uno, total 2 583 €entre ambos.
- Google Cloud Platform (Crédito) (85): GCP nos proporciona 265 €de crédito durante 3 meses, validos mientras quede crédito o no se haya llegado a los 3 meses.
- Google Cloud Platform (App Engine) (86): Con una única instancia de App Engine, tipo F1 (384 MB de memoria y limite de cpu de 600 MHz), la más pequeña, y en la zona europe-west6, conllevaría un coste de $\approx 0,059$ €/h → ≈ 43 €/mes. Asumiendo un tráfico nulo o despreciable durante el desarrollo.
- Firebase (Spark Plan) (87): Plan gratuito que nos proporciona 10 GB de almacenamiento, 360 MB de transferencia diarios y hasta 50.000 usuarios activos mensuales. Incluye Firebase Hosting y Firebase Authentication.
- MongoDB Atlas (Tier M0) (88): Plan gratuito que nos proporciona 512 MB de almacenamiento, 10 GB de transferencia y un máximo de 100 operaciones/s.

En la siguiente Tabla 3.5, calculamos el coste total estimado del proyecto:

Concepto	**Coste (€/mes)**
Desarrolladores (2 × 50 % jornada)	2 583
App Engine (meses 1–3, crédito GCP)	0
App Engine (meses 4–5, F1 europe-west6)	43
Firebase (Spark Plan)	0
MongoDB Atlas (Tier M0)	0
Coste mensual (meses 1–3)	**2 583**
Coste mensual (meses 4–5)	**2 626**
Coste total en 5 meses	**3×2 583 + 2×2 626 = 13 001**

Tabla 3.5: Estimación de costes mensuales y total

Capítulo 4

Metodología de desarrollo

4.1. Planificación Temporal

Esta planificación se apoya en las estimaciones definidas en la Sección 3.8, complementándola con un backlog estructurado y un diagrama de Gantt. El objetivo es disponer de una visión a gran escala (cronograma y dependencias).

4.1.1. Backlog inicial

El backlog es la lista viva y priorizada del trabajo pendiente del proyecto. Centraliza requisitos, tareas y mejoras, y sirve como referencia única para la planificación operativa y la proyección temporal mediante el diagrama de Gantt. La siguiente Tabla 4.1 actúa como plantilla base, organizada en cinco bloques generales; puede actualizarse y detallarse conforme avanza el proyecto.

ID	Nombre
	Sección 1: Planficación y Diseño
S1-1	Obtener Requisitos
S1-2	Modelar Arquitectura
S1-3	Obtener Casos de Uso
S1-4	Diseñar Casos de Uso Extendidos
S1-5	Diseñar Base de Datos
S1-6	Diseñar UI Wireframes
S1-7	Diseñar API endpoints
	Sección 2: Configuración CI/CD y Servicios Cloud
S2-1	Configurar Firebase Hosting
S2-2	Configurar Pipeline CI/CD de GitHub a Firebase Hosting
S2-3	Configurar App Engine (GCP)
S2-4	Configurar Pipeline CI/CD de GitHub a App Engine
S2-5	Configurar Firebase Authentication
S2-6	Configurar Mongo DB Atlas

(Continúa en la página siguiente)

(Continúa de la página anterior)

ID	Nombre
	Sección 3: Desarrollo Back end y BD
S3-1	Conectar Back end con Mongo DB Atlas
S3-2	Implementar Persistencia (CRUD)
S3-3	Implementar Endpoints
S3-4	Implementar Reglas de Negocio
S3-5	Conectar Back end con Firebase Auth.
S3-6	Tests
	Sección 4: Desarrollo Front end
S4-1	Conectar Front end con Firebase Auth.
S4-2	Implementar Login
S4-3	Conectar Front end con Back end
S4-4	Implementar UI Base
S4-5	Implementar Gráficos y Estadísticas
S4-6	Tests
	Sección 5: Pruebas e Integración FInal
S5-1	Diseño de Cuestionario Usabilidad
S5-2	Tests de Usabilidad con Usuarios
S5-3	Tests de verificación de requisitos

Tabla 4.1: Backlog base

4.1.2. Diagrama de Gantt

El diagrama de Gantt es una representación temporal de las tareas del backlog: cada tarea (o grupo de tareas) se muestra como una barra cuya longitud indica su duración y cuyas relaciones reflejan dependencias. En nuestro caso, el Gantt se alimenta de la tabla anterior y sirve para visualizar etapas, así como hitos y ventanas de pruebas.

El cronograma que se muestra a continuación (Figura 4.1) no es una guía estricta, sino la duración y flujo ideal del trabajo: puede ajustarse según prioridad, disponibilidad y resultados de las revisiones semanales.

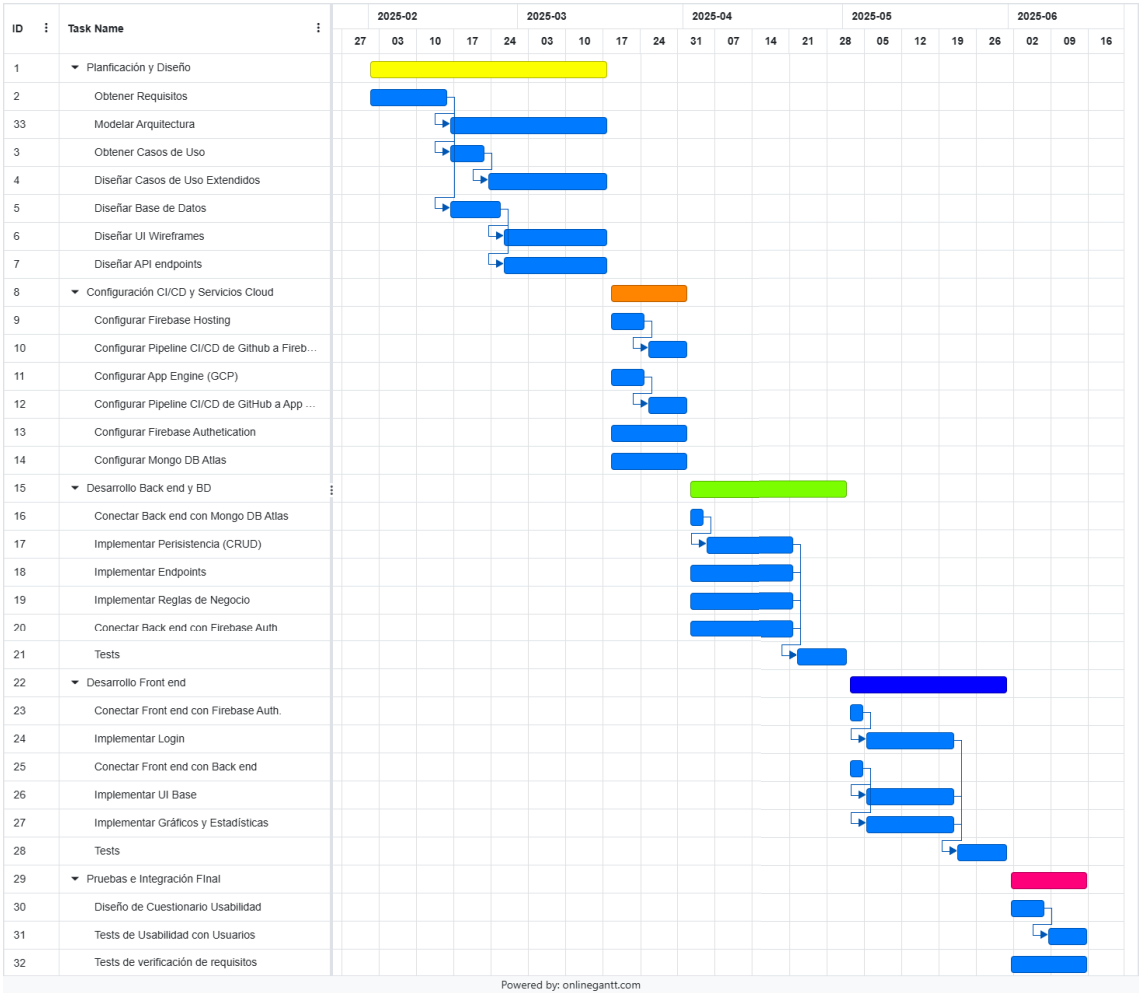


Figura 4.1: Diagrama de Gantt

4.2. Proceso Ágil

4.2.1. Método Kanban

Kanban es un método para gestionar trabajo y servicios que se aplica sobre el proceso existente, sin imponer roles ni ceremonias nuevas de forma prescriptiva. Su propósito es hacer visible el flujo real del trabajo, limitar cuánto llevamos en paralelo y mejorar el sistema de manera evolutiva. La guía oficial (89) resume dos grupos de principios: de gestión del cambio (comenzar con lo que se hace ahora, mejorar mediante cambios pequeños y fomentar liderazgo en todos los niveles) y de gestión del servicio (entender necesidades del cliente, gestionar el trabajo y revisar periódicamente el sistema y sus políticas).

Para que un sistema funcione realmente como Kanban (y no solo como un tablero visual), se aplican prácticas básicas mantenidas en el tiempo: visualizar el trabajo y su flujo; limitar el trabajo en curso (*WIP*) para reducir multitarea y acelerar finalizaciones; gestionar el flujo para que las tarjetas avancen con regularidad; hacer explícitas las políticas (criterios de entrada/salida por columna, definición de "bloqueado", prioridades); establecer bucles de retroalimentación con una cadencia adecuada; y mejorar de forma colaborativa mediante

pequeños experimentos. Con estas prácticas el sistema opera en *pull*: el trabajo se "tira" cuando hay capacidad disponible y se respeta el WIP acordado.

Es clave definir un punto de compromiso (desde dónde una tarea cuenta como "dentro" del sistema) y un punto de entrega (dónde se considera terminada). Entre ambos se miden métricas de flujo como *cycle time*, *lead time* y *throughput*, que permiten detectar cuellos de botella, ajustar límites WIP y negociar expectativas con datos. Conviene diferenciar un "tablero kanban" simple (solo columnas) de aplicar el *Método Kanban* completo: sin límites WIP, políticas explícitas ni puntos de compromiso/entrega, se trata únicamente de una visualización y no de un sistema Kanban real.

4.2.2. ¿Por qué Kanban?

Elegimos Kanban porque encaja con las necesidades reales del proyecto: queremos entregar de forma continua, mantener visibilidad del estado y adaptarnos a cambios sin rehacer toda la planificación.

Otro motivo clave es la flexibilidad académica: no siempre podemos dedicar el 100 % del tiempo, ni el alcance está totalmente cerrado desde el día uno. Kanban no exige sprints o ceremonias fijas; se superpone a nuestro proceso actual. Nos permite visualizar el flujo y promover mejoras evolutivas y sostenibles, algo especialmente adecuado cuando hay aprendizaje en marcha (nuevas herramientas, servicios cloud).

Por último, lo combinamos con un diagrama de Gantt para cubrir dos vistas complementarias: Gantt nos da la foto temporal a gran escala (hitos, dependencias y ventanas de trabajo) y Kanban nos da la gestión operativa del flujo en el día a día (qué está por hacer, en curso, bloqueado o terminado). Es una combinación que nos proporciona planificación de alto nivel sin renunciar al control del trabajo en curso y la adaptación continua.

4.2.3. Nuestro tablero y políticas

Nuestro tablero gobierna cada etapa/sección del desarrollo: no se avanza al tablero de la etapa siguiente hasta completar todas las tareas del tablero actual (cierre por etapas). Las columnas son: **Backlog → Ready → In Progress → Testing → Done**. Vease Figura 4.2. Un ítem solo pasa a Ready cuando cumple condiciones de preparación y no tiene dependencias abiertas; si existe alguna dependencia, permanece en Backlog hasta que se cierre según el diagrama de Gantt.

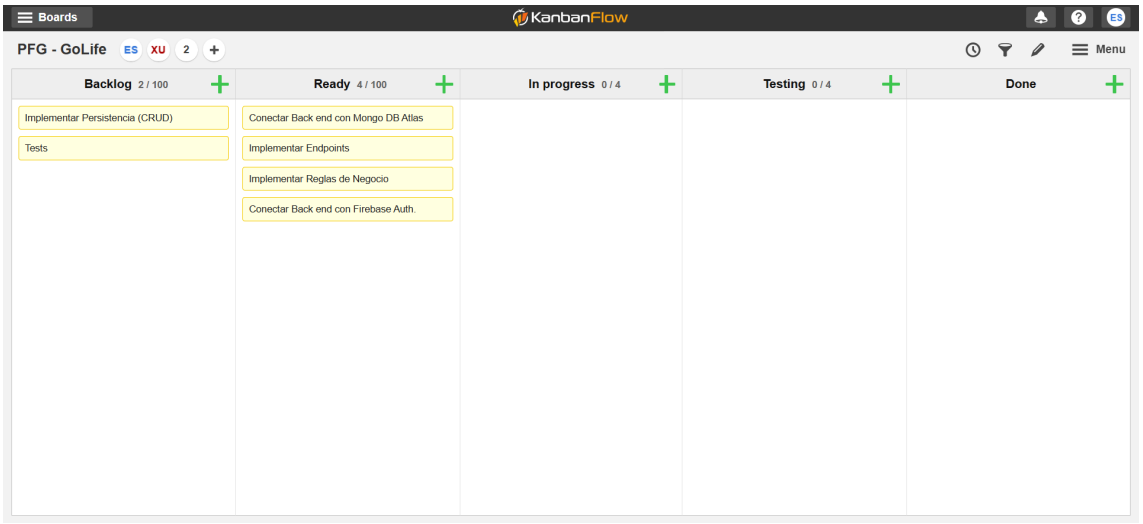


Figura 4.2: Tablero Kanban inicial de etapa de Desarrollo de Back end y BD

Compromiso y entrega

El punto de compromiso del sistema es el paso a Ready (seleccionado para ejecutar); el punto de entrega es Done, que implica push válido en Github. Entre ambos puntos se gestionan límites WIP y políticas.

Límites WIP y política de *pull*

Cada desarrollador puede tener como máximo 2 tareas abiertas a la vez; con 2 desarrolladores, el WIP global en In Progress es 4. Las tarjetas se mueven con pull de derecha a izquierda: cuando hay hueco, se termina lo que está más a la derecha antes de empezar algo nuevo. Las dependencias con otras tareas deben respetar el diagrama de Gantt.

Criterios de salida de Testing

Un ítem puede salir de Testing solo si tiene 100 % de tests pasados, $\geq$80 % de cobertura de instrucciones y $\geq$70 % de cobertura de ramas. Entonces pasa a Done y se registra el *push* correspondiente.

Reapertura de tareas

Las tareas en Done pueden reabrirse (volver a Ready) si aparecen errores/bugs o cambian requisitos. No se pueden reabrir tareas de tableros de etapas ya cerradas.

Clases de servicio

- **Estándar**: coste de retraso moderado; se gestionan con las políticas normales. Por defecto, toda tarea de la que dependen otras (salvo que la dependiente sea el Testing final de etapa) es Estándar.
- **Intangible**: bajo coste de retraso a corto plazo pero impacto a largo. Por defecto, toda tarea no Estándar.

Bloqueos, señalización y cadencias

Todas las tareas son inicialmente amarillas. Si surge una incidencia o bloqueo (Figura 4.3), se marcan en rojo para revisión; hay reunión semanal para revisar trabajo completado, trabajo en curso y estado del proyecto.

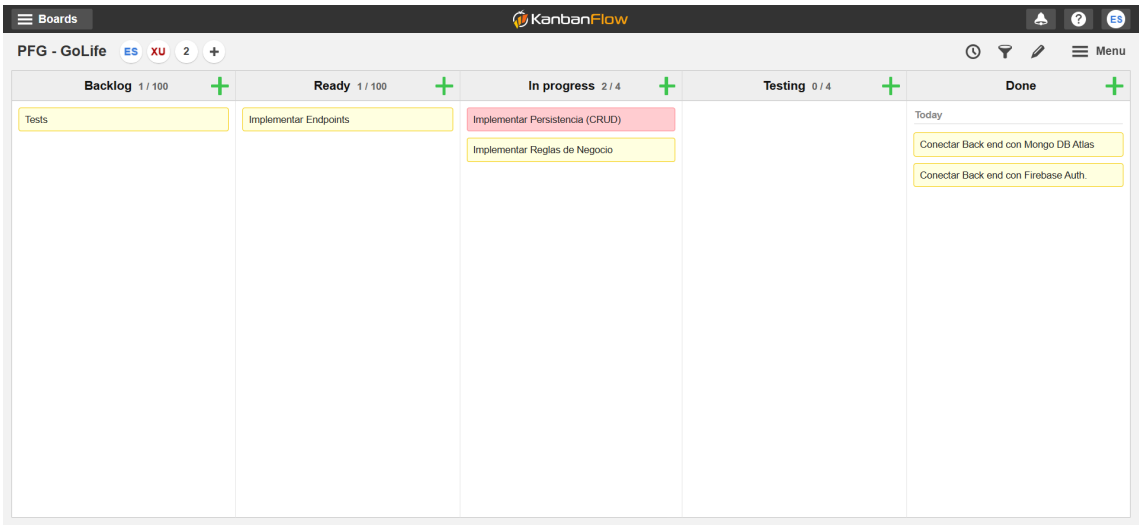


Figura 4.3: Tablero Kanban con incidencia

Políticas por columna (resumen)

Columna	Entrada (DoR)	Salida (DoD)
Backlog	Idea/tarea registrada; dependencias identificadas (pueden estar abiertas).	Pasa a **Ready** cuando está preparada y sin dependencias abiertas.
Ready	Preparada (criterios claros, datos/entorno disponibles).	Pasa a **In Progress** cuando hay capacidad libre (WIP) y se asigna.
In Progress	Asignada y con WIP disponible (2 por persona; 4 global).	Pasa a **Testing** cuando hay build y cambios listos para verificar.
Testing	Recibido desde In Progress; listo para verificación. Tests locales.	Pasa a **Done** con 100 % tests pasados, $\geq$80 % instrucciones y $\geq$70 % ramas, más *push* válido en Github.
Done	Cumple salida de Testing.	Puede **reabrirse a Ready** si aparece bug/cambio; no si pertenece a una etapa/tabla cerrada.

Tabla 4.2: Resumen Políticas de Kanban por columna

4.3. Gestión De Versiones

El proyecto se gestiona en GitHub con dos repositorios separados, uno para front end y otro para back end, de modo que cada componente pueda gestionarse, evolucionar y mantenerse de forma independiente. A continuación se presentan los enlaces clicables a los repositorios de código del proyecto:

(a) Repositorio Front end

(b) Repositorio Back end

Figura 4.4: Repositorios del Proyecto (Clicables)

Capítulo 5

Análisis de requisitos

Los siguientes requisitos han sido extraídos de la siguiente descripción del proyecto, y de múltiples reuniones con nuestro tutor Bordel Sanchez, Borja:

> En este PFG se deberá diseñar y programar una aplicación web, en la que un usuario podrá introducir sus metas y objetivos personales y monitorizar su cumplimiento de forma periódica.
>
> La plataforma web deberá ser genérica, de tal forma que cada usuario pueda introducir objetivos o metas personales. Cada meta deberá tener un valor cuantitativo, que después de monitorizarla, mediante actualizaciones periódicas y un panel de control (dashboard) de visualización.
>
> La aplicación web podrá desarrollarse completamente con una arquitectura cliente-servidor clásica, pero si se desea profundizar podrá generarse una arquitectura cloud con varios componentes de servicios (microservicios) que se decida trabajar en equipo.

5.1. Requisitos Funcionales

- **RF1-Creación de Metas:** El sistema permitirá que el usuario introduzca/cree sus metas en el sistema
- **RF2-Edición de Metas:** El sistema permitirá que el usuario edite sus metas previamente creadas
- **RF3-Actualización de Metas:** El sistema permitirá que el usuario actualice sus metas para progresar en ellas
- **RF4-Eliminación de Metas:** El sistema permitirá que el usuario elimine sus objetivos previamente creados
- **RF5-Cuantificación de Metas:** El sistema cuantificara numéricamente las metas y objetivos personales
- **RF6-Dashboard:** El sistema presentará al usuario un panel de control (dashboard), en el cual visualizar el progreso de sus metas
- **RF7-Creación de Usuario:** El sistema permitirá que el usuario cree su cuenta en mismo sistema, con sus credenciales (email, nombre de usuario y contraseña)
- **RF8-Login:** El sistema permitirá que el usuario acceda a su cuenta con sus credenciales de usuario (email, contraseña)
- **RF9-Eliminación de Usuario:** El sistema permitirá que el usuario elimine su cuenta con sus credenciales de usuario (email, contraseña)
- **RF10-Recuperación de Contraseña:** El sistema permitirá que el usuario pueda modificar su contraseña en caso de haberse olvidado de ella

5.2. Requisitos No Funcionales

5.2.1. Requisitos de Usabilidad

RNF1–Diseño Genérico: El sistema debe presentar un diseño genérico, de tal forma que pueda servir a una multitud de usuarios con metas variadas.

Criterios de Aceptación

1. La interfaz y funcionalidades no deben estar limitadas a un único caso de uso, permitiendo la adaptación a diferentes perfiles de usuarios.
2. Las pruebas de usuario con un grupo diverso (mínimo 5 perfiles diferentes) deben demostrar que al menos el 80 % de los usuarios pueden utilizar las funciones principales sin asistencia.

RNF2–Interfaz Intuitiva El sistema debe presentar una interfaz intuitiva y fácil de usar para una amplia gama de usuarios. O sea, ser user-friendly.

Criterios de Aceptación

1. Se debe realizar al menos una prueba de usabilidad con usuarios reales y obtener una calificación mínima de 4/5 en facilidad de uso.
2. La cantidad de pasos necesarios para completar acciones clave no debe superar los estándares establecidos en aplicaciones similares del mercado.

5.2.2. Requisitos de Escalabilidad

RNF3–Escalabilidad El diseño debe contemplar la posibilidad de crecimiento en la base de usuarios y en la cantidad de datos manejados sin que ello afecte negativamente el rendimiento.

Criterios de Aceptación

1. La arquitectura debe permitir la expansión de almacenamiento y cómputo sin necesidad de una reestructuración completa.

5.2.3. Requisitos de Disponibilidad

RNF4–Disponibilidad El sistema debe estar activo y disponible al menos el 99 % del tiempo durante su operación normal.

Criterios de Aceptación

1. La disponibilidad del sistema debe ser $\geq 99\,\%$ mensual, medido a través de un sistema de monitoreo automatizado.
2. Las interrupciones no planificadas no deben superar las 87,6 horas al año o 7,2 horas al mes.

5.2.4. Requisitos de Infraestructura

RNF5–Arquitectura Cloud El sistema debe ser diseñado con el propósito de presentar una arquitectura cloud. Arquitectura Cloud: modelo de diseño y despliegue de sistemas que utiliza recursos de computación, almacenamiento y redes proporcionados a través de servicios en la nube.

Criterios de Aceptación

1. Todos los servicios del backend deben ejecutarse en una infraestructura en la nube.
2. Los datos deben almacenarse en servicios de bases de datos en la nube, sin dependencias de almacenamiento local.

5.2.5. Requisitos de Seguridad

RNF6–Securización El sistema debe estar securizado y los datos encriptados.

Criterios de Aceptación

1. El sistema debe implementar un esquema de autenticación basado en tokens JWT con expiración configurable.
2. La API debe validar la autenticidad de los tokens en cada solicitud protegida y rechazar aquellas sin un token válido.

Capítulo 6

Diseño conceptual y modelado

Este capítulo presenta los resultados de la fase de diseño del proyecto y la evolución de los distintos modelos elaborados hasta alcanzar la solución actual. El objetivo es traducir los requisitos (funcionales y no funcionales) a artefactos de diseño que guíen la implementación y permitan razonar sobre los datos y los flujos principales del sistema.

6.1. Casos de Uso

Un **diagrama de casos de uso** ofrece una vista de alto nivel sobre qué hace el sistema desde la perspectiva de sus usuarios y de servicios externos. Permite identificar actores, delimitar el sistema y agrupar las capacidades que este ofrece (casos de uso), así como las dependencias más relevantes entre ellas. Es una herramienta para acordar el alcance funcional con las partes interesadas y para facilitar la trazabilidad con los requisitos, sirviendo como puente hacia la especificación detallada (casos de uso extendidos), sin entrar aún en la implementación.

El siguiente diagrama de casos de uso (Figura 6.1) sintetiza los actores y las interacciones identificadas tras analizar los requisitos y elaborar el modelo de diseño. A continuación se describen los actores implicados:

Usuario: Persona que interactúa con la aplicación para crear, actualizar y consultar sus metas.

Firebase Authentication: Servicio externo que actúa como proveedor de autenticación y registro de usuarios.

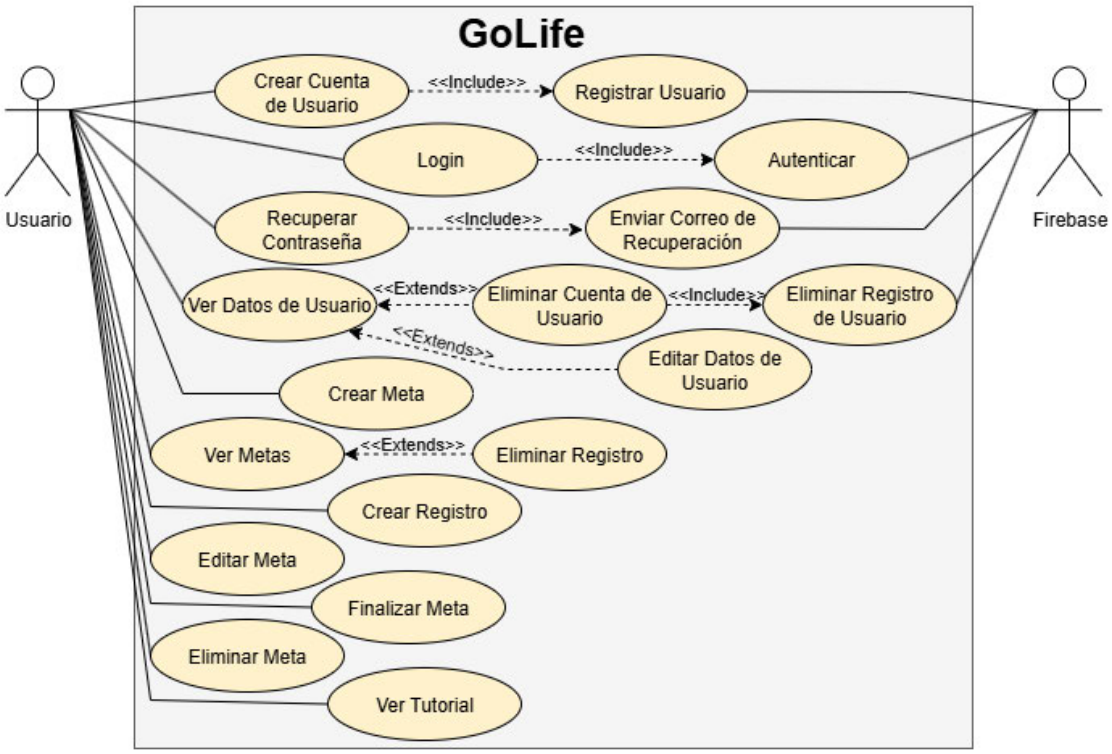


Figura 6.1: Diagrama de Casos de Uso de GoLife

6.2. Casos de Uso Extendidos

Los **Casos de Uso Extendidos** desarrollan el *Diagrama de Casos de Uso* previo y se documentan en una **ficha tabular** por caso con la siguiente estructura: ID, Nombre, Actor primario, Actores secundarios, Descripción, Evento activador, Precondiciones, Flujo normal (secuencia numerada de pasos), Flujos alternativos y de error, Postcondiciones, Prioridad y, cuando aplica, Otra información y Suposiciones. Se anotan además relaciones `«include»` / `«extend»` entre casos. Este nivel de detalle permite razonar sobre variantes, errores y procesos del sistema.

A continuación se incluyen **ejemplos representativos** de los casos de uso extendidos diseñados mediante la Figura 6.2 y Figura 6.3. El inventario completo, con todos los flujos y alternativas, puede consultarse en el Anexo: Casos de Uso Extendidos de GoLife.

ID:	CU-1
Nombre:	Crear Cuenta de usuario
Actor primario:	Usuario
Actores secundarios:	Firebase
Descripción:	Los usuarios pueden crear su cuenta en el sistema
Evento activador:	El usuario pulsa el botón de “Crear Cuenta” en la pantalla de Login
Precondiciones:	El usuario no debe tener una cuenta existente con el mismo correo en el sistema
Postcondiciones:	• La cuenta ha sido creada exitosamente • El usuario puede loguearse con sus credenciales
Flujo Normal:	1. El usuario accede a la pantalla de Login 2. El usuario introduce sus credenciales en la pantalla de Login (correo y contraseña) 3. El usuario pulsa el botón de “Crear Cuenta” 4. El sistema envía los datos ingresados a Firebase 5. <<include>> CU-2 Registrar Usuario 6. El sistema recibe la respuesta de Firebase 7. El sistema crea el perfil del usuario en su base de datos 8. El sistema carga los datos del usuario 9. El sistema carga la pantalla de Dashboard
Flujo Alternativo:	6.a.1. El sistema recibe una respuesta de error de Firebase 6.a.2. El sistema muestra el error en la pantalla de Login
Prioridad:	Alta
Otra información:	
Suposiciones:	

Figura 6.2: Caso de Uso Extendido: Crear Cuenta de Usuario

ID:	CU-8
Nombre:	Eliminar Cuenta de Usuario
Actor primario:	Usuario
Actores secundarios:	Firebase
Descripción:	Los usuarios pueden eliminar su cuenta y con ella todos sus datos correspondientes
Evento activador:	El usuario pulsa el botón de "Eliminar cuenta"
Precondiciones:	El usuario se encuentra en la pantalla de Perfil
Postcondiciones:	La cuenta del usuario y todos sus datos han sido eliminados y el usuario se encuentra en la pantalla de Login
Flujo Normal:	1. El usuario pulsa el botón de "Eliminar Cuenta" 2. El sistema una pantalla de confirmación 3. El usuario pulsa el botón de "Eliminar cuenta" 4. El sistema solicita a Firebase eliminar la cuenta del correo del usuario 5. <<Include>> CU-9 Eliminar Registro de Usuario 6. El sistema recibe la respuesta de Firebase 7. El sistema borra los datos del usuario 8. El sistema desloguea al usuario 9. El sistema carga la pantalla de Login
Flujo Alternativo:	3.a.1. El usuario pulsa el botón "Cancelar" 3.a.2. El sistema cierra la pantalla de confirmación 5.a.1. El sistema recibe un error de Firebase 5.a.2. El sistema muestra el error en la pantalla de Perfil
Prioridad:	Media
Otra información:	
Suposiciones:	

Figura 6.3: Caso de Uso Extendido: Eliminar Cuenta de Usuario

6.3. Modelado de Base de Datos

6.3.1. Modelado conceptual de Metas

GoLife aborda la gestión personal desde un enfoque centrado en metas u objetivos medibles, con dos únicos tipos de meta que simplifican el uso y, aun así, cubren la mayoría de casos en los dominios descritos:

Modelo de metas empleado

- **Meta numérica**: registra un valor diario con unidad personalizable (horas, km, páginas, €, kg, . . .) y se evalúa frente a un número objetivo X. El sistema no realiza acumulaciones automáticas ni sumas por periodo; el usuario interpreta si busca subir o bajar hacia X. La evaluación diaria puede basarse en estar en objetivo (cumple hoy) o en la distancia al objetivo para esa meta.
- **Meta booleana**: un registro sí/no por día (¿se cumplió hoy?). Permite medir frecuencia en el tiempo y rachas (días consecutivos en los que se cumple).

Cómo este modelo cubre los dominios

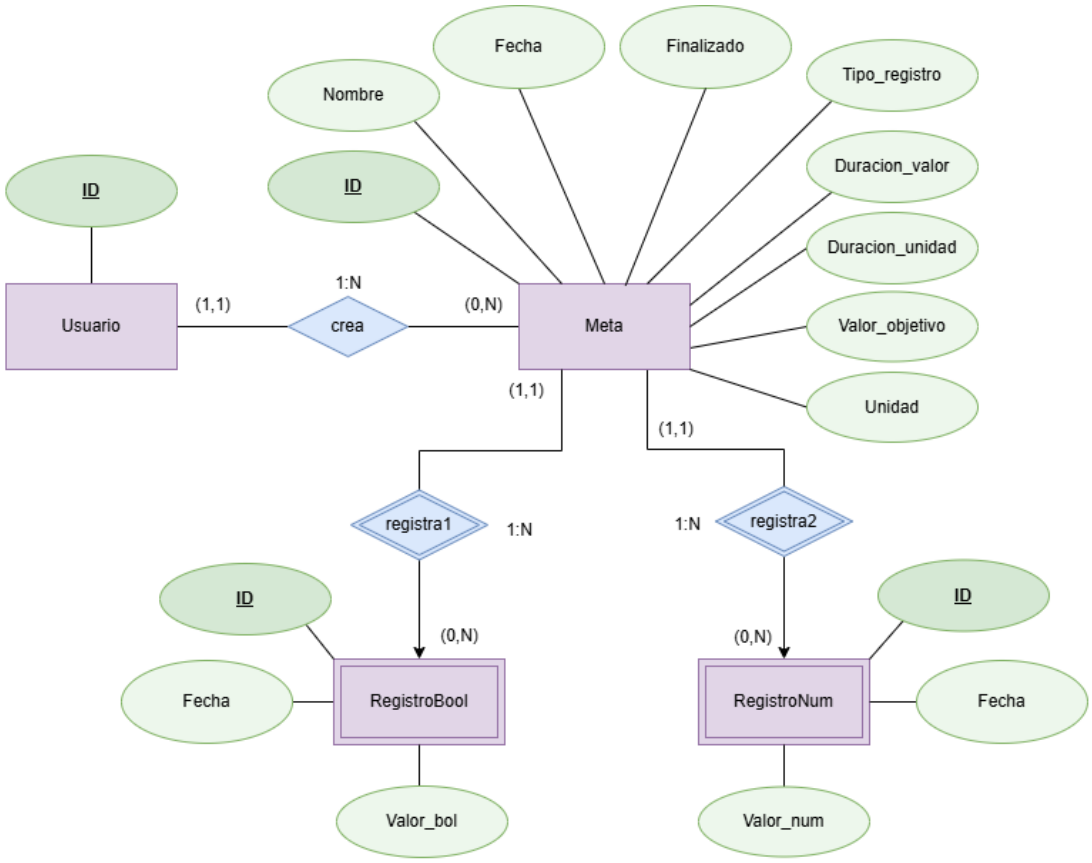
Estos dominios son los descritos en la Subsección 2.1.2.

- **Tareas y uso del tiempo:** sin calendarios ni dependencias complejas, una "próxima acción" puede modelarse como meta booleana ("¿La realicé hoy?"). Si se desea controlar tiempo de foco diario, se usa meta numérica con objetivo fijo por día (p. ej., 90 min), evaluando día a día si se alcanzó X.
- **Hábitos:** se representan directamente con booleanas ("¿Leí hoy?", "¿Salí a correr?") y sus rachas. Cuando el hábito requiere cantidad diaria (vasos de agua, pasos), se usa meta numérica con objetivo fijo por día (p. ej., ≥ 8 vasos).
- **Finanzas personales (básicas):** metas numéricas donde el usuario introduce un valor diario significativo (p. ej., "gasto de hoy $\leq X$ €" o "saldo actual" frente a una cifra objetivo). No hay sumas mensuales automáticas: la verificación es diaria y por meta.
- **Objetivos:** metas numéricas con número objetivo X (el usuario decide si pretende acercarse por arriba o por abajo) y metas booleanas para avances binarios ("¿Avancé hoy?"). Cada meta muestra su cumplimiento diario y, en su caso, racha.

6.3.2. Modelo inicial

Diagrama Entidad–Relación (versión inicial)

El siguiente Diagrama Entidad–Relación (Figura 6.4) ilustra la estructura principal de datos de **GoLife**. Cada **Usuario** puede crear múltiples **Metas**. Una **Meta** es de tipo *numérica* o *booleana* y, en función de ello, sus avances diarios se registran en **RegistroNum** o **RegistroBool**, respectivamente.



Semántica no contemplada

Meta: Una misma *Meta* no podra tener relación con *RegistroBool* y *RegistroNum*, solo podrá tener relación con el que ponga en *Tipo_registro*

Meta: Si *Duración_unidad* = 'indefinido' entonces *Duracion_valor* = 'null'

Meta: Al crearse *Meta* el valor de *Finalizado* sera '0'

DOMINIOS

Usuario
ID: Varchar(20)

RegistroBool
Id: Integer
Fecha: datetime
Valor_bol: TINYINT(1) // 0=false 1=true

RegistroNum
Id: Integer
Fecha: datetime
Valor_num: float

Meta
Id: Integer
Nombre: Varchar(50)
Fecha: datetime
Finalizado: TINYINT(1) // 0=false 1=true
Tipo_registro: 'RegistroBool', 'RegistroNum'
Duración_Valor: Integer
Duración_unidad: 'dias', 'semanas', 'meses', 'años', 'indefinido'
Valor_objetivo: float
Unidad: Varchar(20)

Figura 6.4: Diagrama Entidad-Relación de GoLife

Diagrama de clases (versión inicial)

El siguiente Diagrama de Clases (Figura 6.5) muestra la estructura básica del modelo de dominio de la aplicación. Cada **Usuario** puede tener múltiples **Metas**, las cuales incluyen información como el tipo de registro (*numérico* o *booleano*) y metadatos de identificación y estado. Para llevar el progreso de cada meta, se definen dos clases de **Registro** diferenciadas: **RegistroNum** para valores numéricos diarios y **RegistroBool** para confirmaciones diarias sí/no.

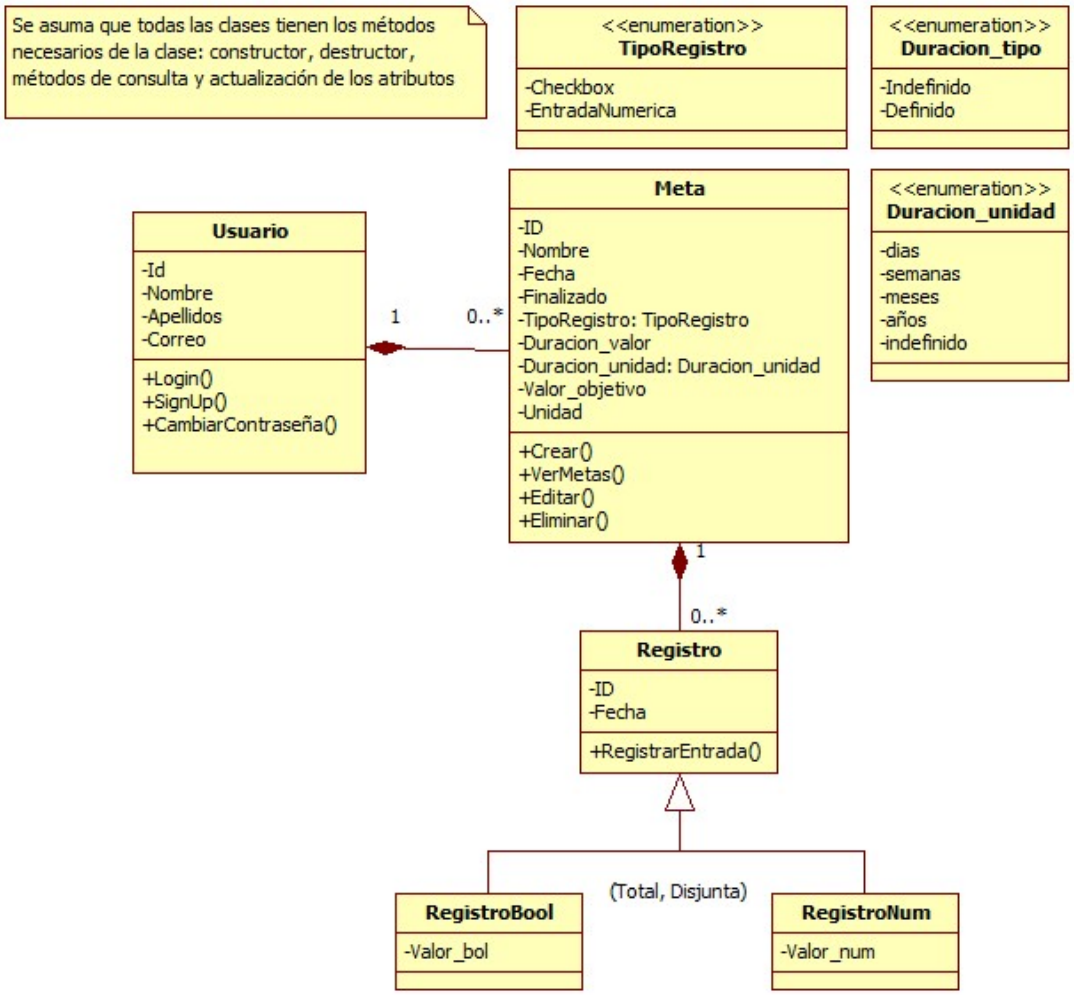


Figura 6.5: Diagrama de Clases de GoLife

6.3.3. Decisión de cambio a base de datos no relacional

La evolución del diseño nos llevó a migrar desde un esquema relacional a un *documento BSON* (JSON binario) de MongoDB por razones de modelo y de operación.

En primer lugar, la naturaleza **polimórfica** de las metas (*numéricas* y *booleanas*) y la necesidad de exponer **respuestas JSON uniformes** en los endpoints implicaban múltiples

joins y consultas complejas para componer cada payload en SQL, con el consiguiente coste en latencia y mantenibilidad.

En segundo lugar, al tratarse de una aplicación web , priorizamos **esquema flexible** y **escalabilidad horizontal** (sharding) junto con alto rendimiento en operaciones de lectura/escritura típicas del patrón de documentos embebidos.

Motivaciones principales

- **Ajuste natural al dominio y al payload:** el documento de *Meta* puede embedir (o vincular) sus registros diarios sin *joins* costosos, alineando almacenamiento y forma del JSON servido.
- **Consultas más sencillas:** agregaciones sobre arrays y filtrados por fecha/estado se expresan de forma directa, reduciendo complejidad de SQL y sobrecarga de composición de respuestas.
- **Rendimiento y escalabilidad:** distribución horizontal y particionado para manejar volúmenes grandes de escritura/lectura con patrones temporales.

El cambio a NoSQL no implica renunciar a la seguridad transaccional donde importa. En MongoDB, las operaciones a **nivel de documento** son **ACID**, y existen **transacciones multi–documento** cuando una operación debe abarcar varias colecciones (90).

Para nuestro patron de acceso, la atomicidad por documento es suficiente en la gran mayoría de casos. Cuando puntualmente se requiera coherencia cruzada, se puede optar por transacciones o por diseños de consistencia explícita. En paralelo, la plataforma ofrece propiedades **BASE** (alta disponibilidad y consistencia eventual) que permiten equilibrar rendimiento y durabilidad.

6.3.4. Modelo final en MongoDB

En la base de datos se han definido dos colecciones principales:

- **Usuarios**: almacena los documentos de usuario y sus metadatos.
- **Metas**: almacena los objetivos definidos por cada usuario y sus registros.

La siguiente Figura 6.6 expone la estructura de documentos de la base de datos:

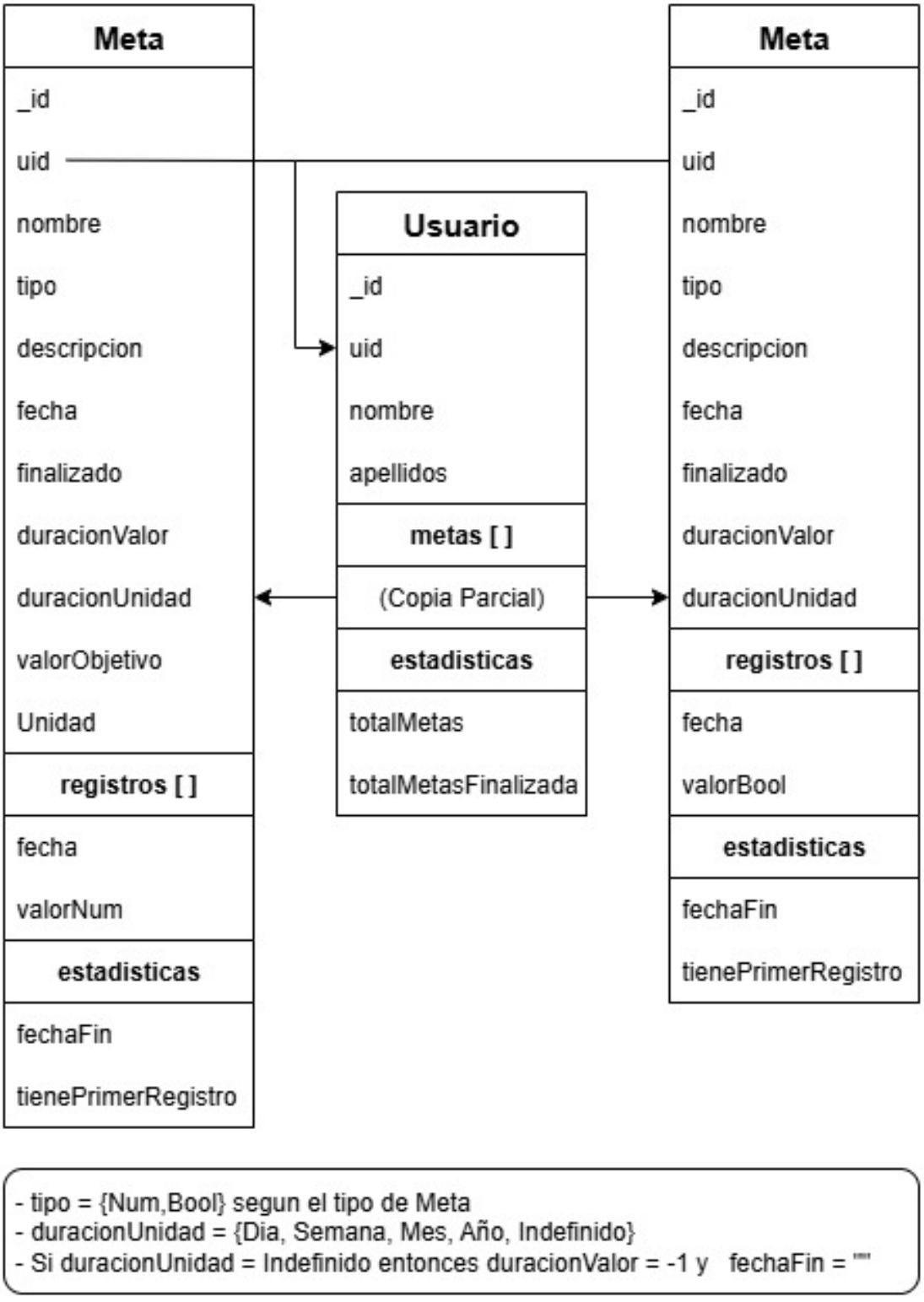


Figura 6.6: Esquema de Documentos de la BD de GoLife

Ejemplos JSON

Como complemento al esquema lógico de la Figura 6.6, se incluyen a continuación tres ejemplos JSON que ilustran la estructura de los documentos persistidos.

Ejemplo 1 - Documento de Usuario

```
{
  "_id": {
    "$oid": "689b6faa28307420129bead0"
  },
  "uid": "cneaMWRxiaXPXTZHKefz3An5zxEsX0j93MFi54vXdZs=",
  "nombre": "Jaime",
  "apellidos": "Perez",
  "metas": [
    {
      "_id": {
        "$oid": "689cee000a7b9b94e61ce4c1"
      },
      "nombre": "Leer mas libros",
      "tipo": "Num",
      "fecha": "2025-02-15",
      "finalizado": false,
      "duracionValor": -1,
      "duracionUnidad": "Indefinido",
      "valorObjetivo": 10,
      "unidad": "Libros"
    },
    {
      "_id": {
        "$oid": "68a3c79034ee23aa560a9e13"
      },
      "nombre": "Correr",
      "tipo": "Bool",
      "fecha": "2025-02-28",
      "finalizado": false,
      "duracionValor": 3,
      "duracionUnidad": "Semanas"
    }
  ],
  "estadisticas": {
    "totalMetas": 2,
    "totalMetasFinalizadas": 0
  }
}
```

Listado 6.1: Documento de Usuario

Ejemplo 2 - Documento de Meta Numérica

```
{
  "_id": {
    "$oid": "689cee000a7b9b94e61ce4c1"
  },
  "uid": "cneaMWRxiaXPXTZHKefz3An5zxEsX0j93MFi54vXdZs=",
  "nombre": "Leer mas libros",
  "tipo": "Num",
  "descripcion": "Leer 10 libros",
  "fecha": "2025-02-15",
  "finalizado": false,
  "duracionValor": -1,
  "duracionUnidad": "Indefinido",
  "valorObjetivo": 10,
  "unidad": "Libros",
  "estadisticas": {
    "fechaFin": "",
    "tienePrimerRegistro": true
  },
  "registros": [
    {
      "fecha": "2025-02-15",
      "valorNum": 4
    },
    {
      "fecha": "2025-02-17",
      "valorNum": 1
    }
  ]
}
```

Listado 6.2: Documento de Meta Numérica

Ejemplo 3 - Documento de Meta Booleana

```
{
  "_id": {
    "$oid": "68a3c79034ee23aa560a9e13"
  },
  "uid": "cneaMWRxiaXPXTZHKefz3An5zxEsX0j93MFi54vXdZs=",
  "nombre": "Correr",
  "tipo": "Bool",
  "descripcion": "Correr a Diario",
  "fecha": "2025-02-28",
  "finalizado": false,
  "duracionValor": 3,
  "duracionUnidad": "Semanas",
  "estadisticas": {
    "tienePrimerRegistro": true,
    "fechaFin": "2025-03-21"
  },
  "registros": [
    {
      "fecha": "2025-02-28",
      "valorBool": false
    },
    {
      "fecha": "2025-03-01",
      "valorBool": true
    }
  ]
}
```

Listado 6.3: Documento de Meta Booleana

Patrones de modelado aplicados

A continuación se describen los patrones de diseño empleados y su manifestación en el esquema.

Referencia extendida (Extended Reference):

- **Definición breve:** duplicar en el documento principal un **subconjunto de campos de alta demanda** del documento referenciado, junto al identificador, para evitar *joins/lookups* repetidos y acelerar lecturas. (91)
- **Dónde se refleja:** en el documento de usuario se guarda, para cada meta, un resumen con campos clave (p. ej., `nombre`, `tipo`, estado, fechas) y el `_id` de la meta para saltar al detalle cuando se necesita; véase el Listado 6.1 frente a los Listado 6.2 y Listado 6.3.

Calculado (Computed):

- **Definición breve:** precalcular y materializar valores derivados (contadores, agregados, flags) para reducir coste en lecturas frecuentes, actualizándolos en los ciclos de escritura o en tareas diferidas. (92)
- **Dónde se refleja:** en estadisticas del usuario (`totalMetas` y `totalMetasFinalizadas`) y en estadísticas de metas (`tienePrimerRegistro`, `fechaFin`). Vease estadísticas en los Listado 6.1, Listado 6.2 y Listado 6.3.

Esquema polimórfico (Polymorphic Schema):

- **Definición breve:** una misma "entidad" con **formas distintas** diferenciadas por un discriminador de tipo (p. ej., `tipo`). (93)
- **Dónde se refleja:** metas `Num` vs `Bool`, con estructuras de `registros[]` diferentes (p. ej., `valorNum` frente a `valorBool`) y campos específicos como `valorObjetivo`/`unidad`; véase el Listado 6.2 y Listado 6.3

Además, se ha decidido **embeber** la lista `registros[]` dentro del documento de `metas` porque el acceso típico a la meta completa implica leer sus registros; así se evitan *joins* y se reduce la latencia de lectura/escritura al operar sobre un único agregado. En cuento a los registros individuales, ahora se identifican a partir de su valor `fecha`; con la restricción de únicamente poder haber un registro por día.

Es notable, que dada la situación, la lista de `registros[]` puede potencialmente crecer de manera indefinida. Por eso ahora vamos a tratar las diferentes preocupaciones al respecto y mostrar porque es una decisión deliberada:

Almacenamiento:

- Solo se permite **un registro por día** y el **tamaño por registro es despreciable** (véase la Tabla 6.1).
- Aun con un registro diario durante un año, el total no alcanzaría $\sim 0{,}01$ MB.
- El tamaño máximo de un documento BSON es **16 MB**, por lo que el almacenamiento no es un problema en este contexto. (94)

Registro	**Variable**	**Tamaño (bytes)**	**Total 1 año (MB)**	**Total 27 años (MB)**
Numérico	`fecha`	14	0.00803	0.21681
	`* String - 10 chars`			
	`valorNum`	8		
	`* Double`			
Booleano	`fecha`	14	0.005475	0.147826
	`* String - 10 chars`			
	`valorBool`	1		
	`* Boolean`			

Tabla 6.1: Estimación de tamaño de `registros[]`

Rendimiento:

- MongoDB maneja de forma eficiente lecturas/escrituras sobre arrays embebidos; y, dado el **alcance acotado** del proyecto y la **restricción de un registro/día**, priorizamos la **simplicidad operativa**.
- Como referencia, el **Performance Advisor** puede señalar arrays "desacotados" y, en su API de **schema advice**, incluye descriptores de *Arrays con más de 10 000 entradas* como candidatos a revisión de esquema (*triggerType* `DOCS_CONTAIN_UNBOUNDED_ARRAY`). (95)
- Eso supone $\approx$ 10 000 días, es decir, $\sim$ 27,4 años de registros diaros continuados. Incluso acercarse a la mitad del umbral (5 000) implica $\sim$ 13,7 años.
- En nuestro caso, **no** se requieren agregaciones, operaciones o procesados complejos sobre `registros[]`; por lo tanto, mantenemos la **lista embebida completa**.

Índices:

- **Índice por `_id` (automático).** MongoDB crea de forma automática un índice único por `_id` en cada colección. Este índice cubre búsquedas puntuales por identificador y es suficiente para operaciones de lectura/escritura sobre un documento concreto (p. ej., una meta).

- **Índice por `uid` (acceso/filtrado por propietario).** Para acelerar las consultas y filtros por usuario se añade un índice sobre `uid`. Todas las consultas y actualizaciones se realizarán filtrando por `{ uid: CURRENT_UID, ...}`, lo que mejora el rendimiento y refuerza el aislamiento lógico por propietario.

```
1 /* El indice por _id es automtico; no requiere creacion manual */
2
3 db.users.createIndex({ uid: 1 });
4 db.goals.createIndex({ uid: 1 });
```

Listado 6.4: Creación de Índices en MongoDB

6.4. Modelado de la Arquitectura

6.4.1. Arquitectura inicial

En su ideación inicial, GoLife tenía cuatro componentes principales para asegurar su correcto funcionamiento y despliegue completo en la nube. Estos son Firebase Hosting, Firebase Authentication, App Engine de Google Cloud Platform y CloudSQL de Google Cloud Platform. Dicha arquitectura viene mostrada por la siguiente Figura 6.7.

Los servicios de base de datos y alojamiento de API back end son proporcionados por Google Cloud Platform (GCP). Mientras que el alojamiento de front end y gestión y autenticación de cuentas de usuario por Firebase, también parte de Google.

- **Firebase Hosting**: alojamiento del front end.
- **Firebase Authentication**: gestión y autenticación de cuentas de usuario.
- **App Engine (GCP)**: ejecución de la API back end.
- **Cloud SQL (GCP)**: base de datos relacional.

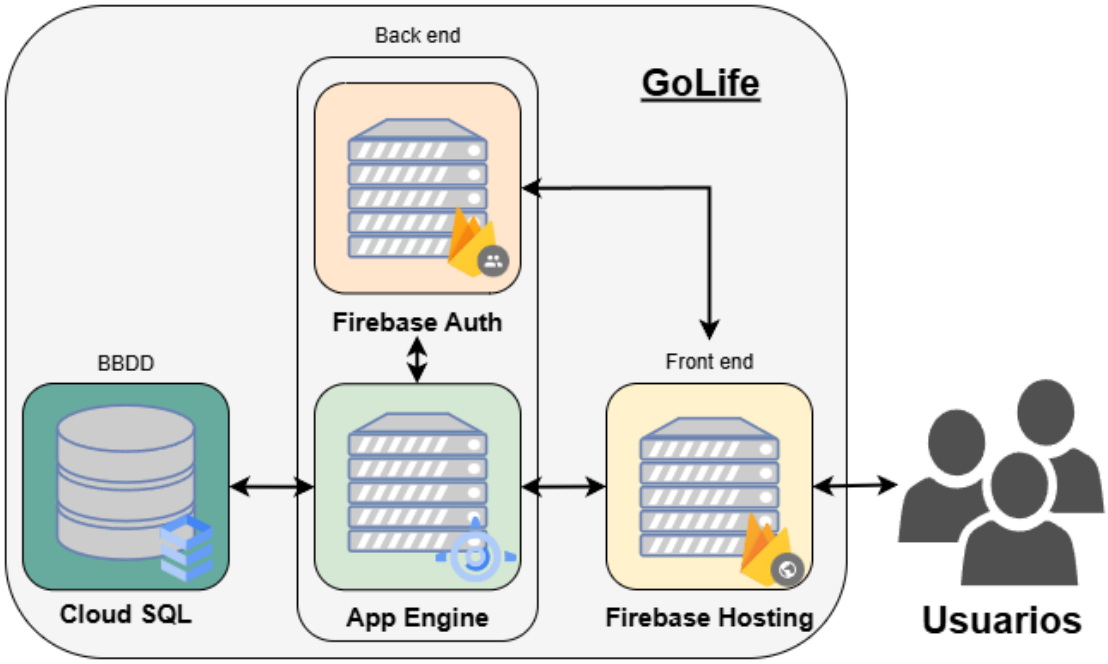


Figura 6.7: Diagrama de Arquitectura Inicial de GoLife

6.4.2. Evolución y motivos del cambio

El primer cambio arquitectónico vino motivado por la decisión de migrar de una base de datos relacional a una no relacional, tal y como se explica en la Subsección 6.3.3. Aprovechando ese punto de inflexión, se decidió además diversificar la arquitectura y no depender exclusivamente de un único proveedor: por ello, el servicio **Cloud SQL** de Google Cloud Platform se sustituyó por **MongoDB Atlas** como servicio cloud de base de datos de documentos.

Este cambio se adoptó por los siguientes motivos principales:

- **Ajuste al dominio:** las metas y sus registros se modelan de forma natural como documentos (embedding), reduciendo composición de respuestas.
- **Simplicidad operativa:** menos lógica de *joins* y transformaciones para servir JSON.
- **Diversificación:** reducción del *vendor lock-in* al introducir un servicio gestionado fuera de GCP.

Como resultado, se configura la arquitectura intermedia mostrada en la Figura 6.8.

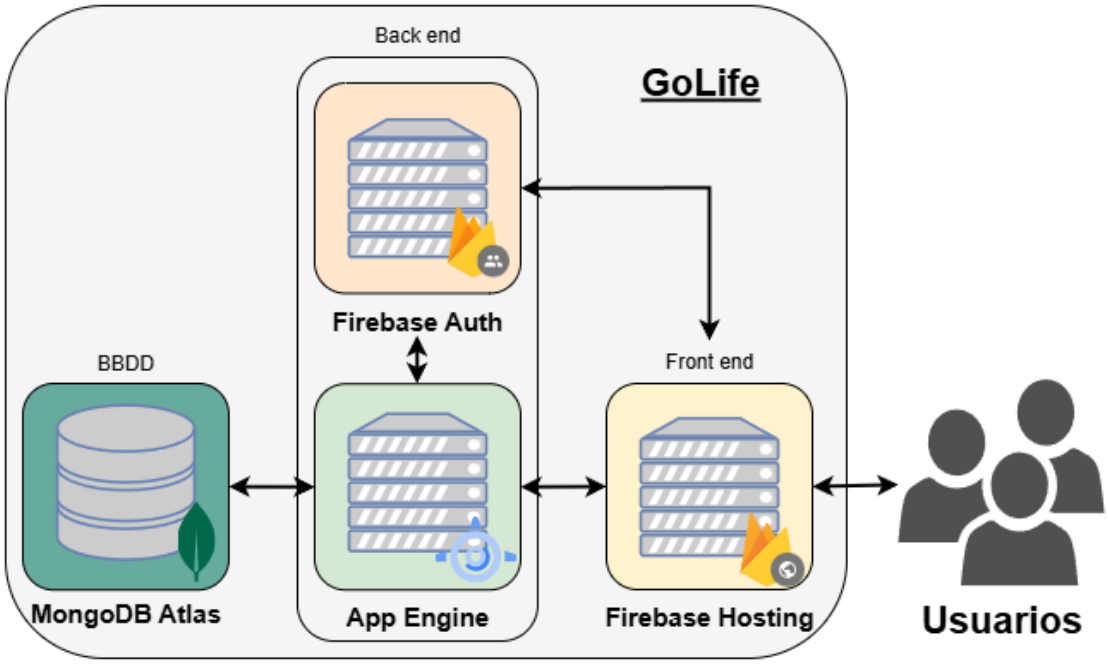


Figura 6.8: Diagrama de Arquitectura Intermedia de GoLife

El segundo cambio vino tras revisar el cumplimiento de la **RGPD**. Se incorporó el **cifrado de elementos identificativos**. Y por ende, se dejó de almacenar el `uid` en claro y se pasó a guardar únicamente un **seudónimo cifrado** en la base de datos. Con ello se rompe el vínculo directo entre el correo de la cuenta y sus metas/registros, cumpliendo el objetivo de **seudonimización** del Art. 4 (96). Operativamente, la API sigue filtrando por el seudonimo, por lo que la funcionalidad se mantiene sin exponer identificadores en claro.

Para realizar el cifrado/verificación sin repartir claves por el sistema, se introdujo un **gestor de claves (Cloud KMS de Google Cloud Platform)**: sólo un componente con acceso a KMS calcula o verifica el seudonimo cifrado y el resto de servicios nunca manipula información de identificación personal (PII) en claro. Esta separación de funciones refuerza el principio de mínimo privilegio, facilita la rotación y custodio de claves fuera de la aplicación y alinea la seguridad del tratamiento con el Art. 32 (97).

6.4.3. Arquitectura final

La presente sección consolida la solución resultante tras la evolución descrita. El diseño final incorpora las dos decisiones clave ya motivadas: (i) almacenamiento documental en MongoDB Atlas y (ii) seudonimización del identificador de usuario mediante un gestor de claves (Cloud KMS), manteniendo la validación de identidad con Firebase Authentication. La figura Figura 6.9 muestra esta visión a alto nivel.

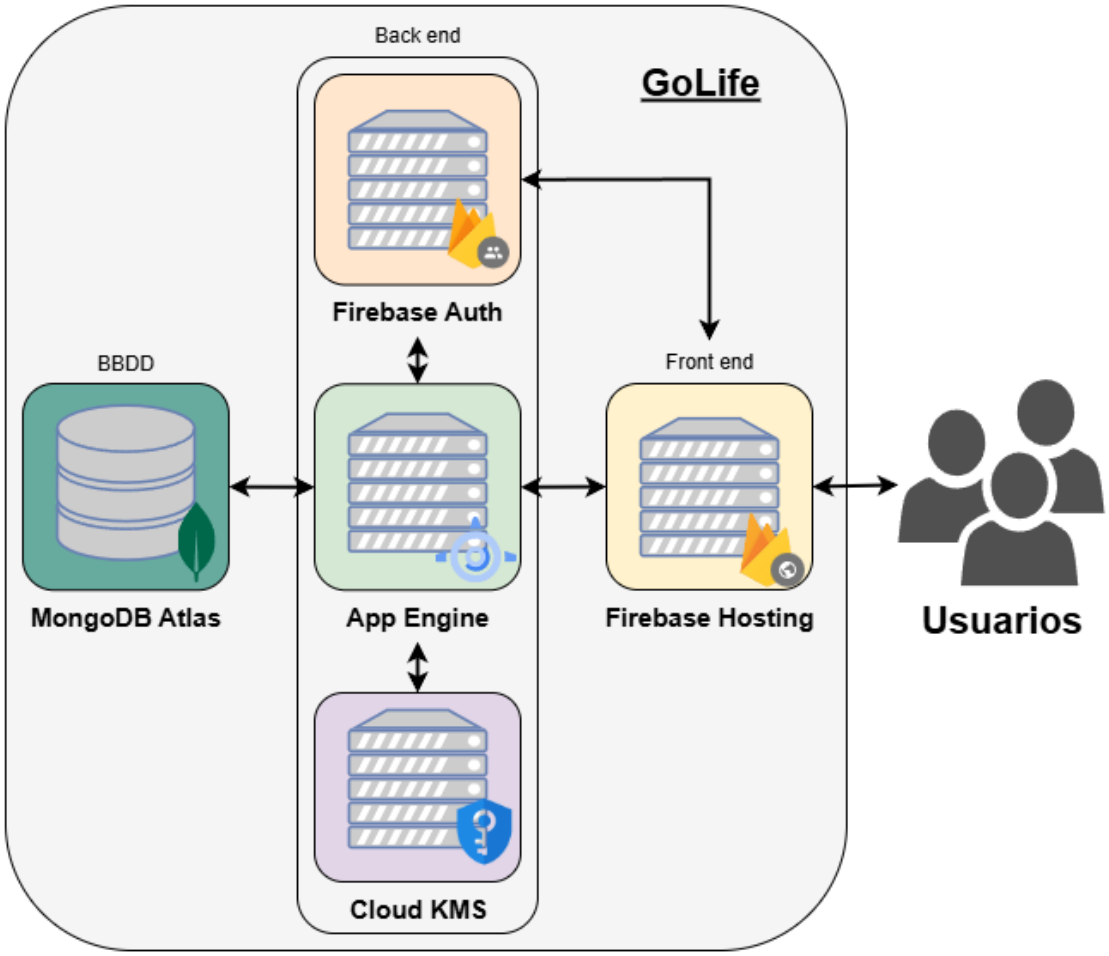


Figura 6.9: Diagrama de Arquitectura Final de GoLife

Componentes y papel de cada uno

- **Firebase Hosting:** servicio PaaS que aloja el cliente web que invoca la API y presenta el dashboard.
- **App Engine:** servicio PaaS que aloja la API que expone endpoints REST y coordina la lógica.
- **Firebase Authentication:** servicio BaaS que gestiona cuentas y emite/valida JWT por petición.
- **MongoDB Atlas:** servicio DBaaS que proporciona la persistencia documental de usuarios y metas.
- **Cloud KMS:** servicio PaaS (más especificamente SECaaS o Seguridad como Servicio) que gestiona claves y genera/verifica el seudónimo cifrado del `uid` para no almacenar identificadores en claro.

A continuación se presentan dos diagramas de secuencia conceptuales que ilustran cómo colaboran los componentes de la solución (Front end servido por Firebase Hosting, API

en App Engine, Firebase Authentication, Cloud KMS y MongoDB Atlas) ante dos flujos comunes: **login** y **consulta de meta**. Su objetivo es mostrar la coreografía entre componentes, sin entrar demasiado en detalles.

Login: El cliente autentica con el SDK de Firebase (requiere red) y obtiene un *ID Token (JWT)*. La API valida ese token localmente (mediante claves JWKS cacheadas) y, cuando procede, deriva un cifrado del `uid` mediante Cloud KMS para operar sobre la base de datos.

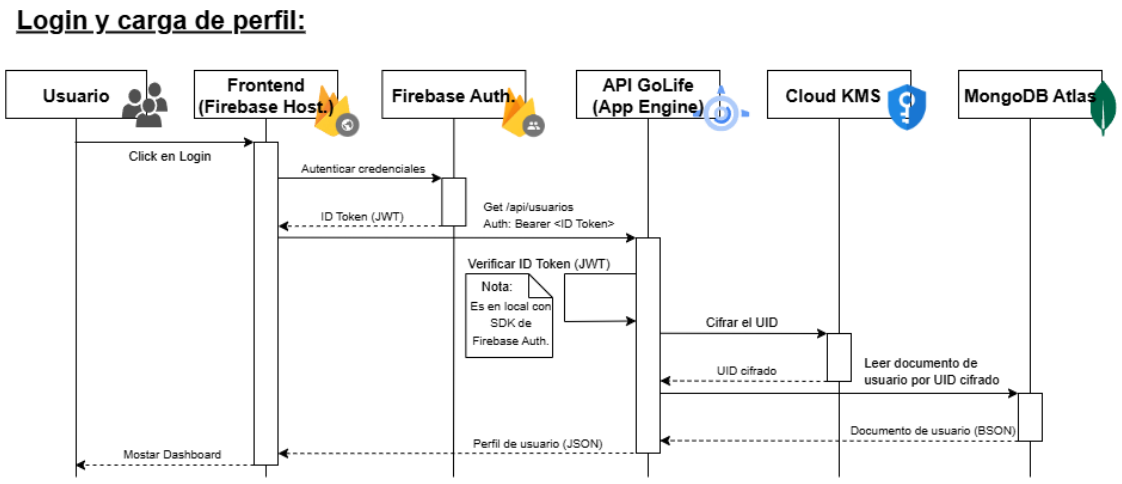


Figura 6.10: Diagrama de Secuencia de Login de GoLife

Consulta de meta: El Front end solicita la meta autenticado con Bearer JWT. La API verifica el token en local y consulta en MongoDB Atlas la meta mediante su id único. Luego valida la pertenencia mediante una verificación del cifrado del `uid` existente en la meta mediante Cloud KMS.

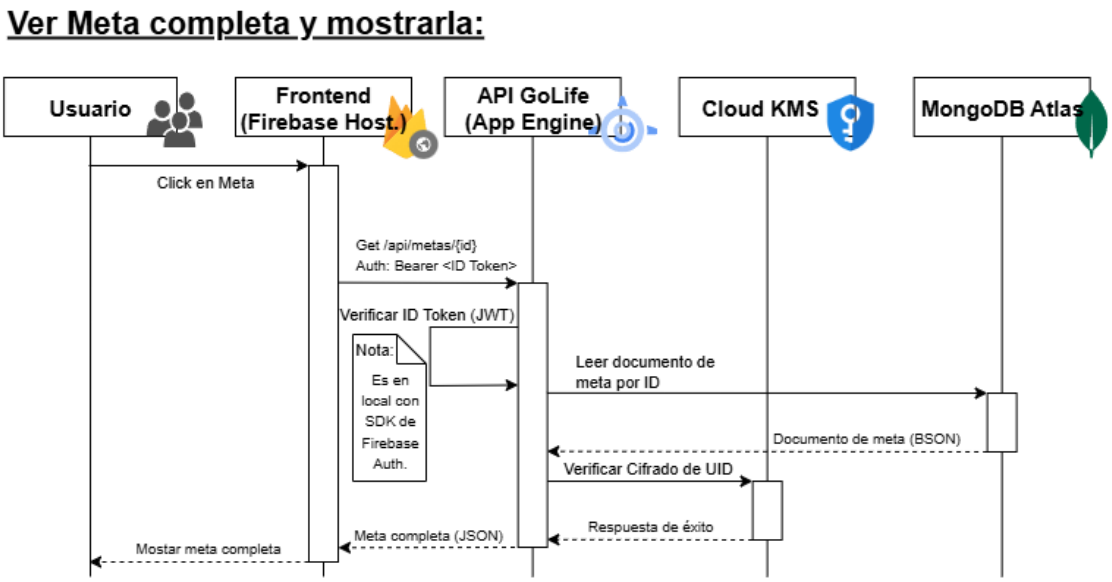


Figura 6.11: Diagrama de Secuencia de ver Meta completa de GoLife

6.5. Modelado de la API

6.5.1. Diseño de endpoints inicial

El diseño inicial de la API sigue un enfoque **RESTful** para la comunicación entre el front end y el back end. La **autenticación** se modeló mediante `Bearer ID Token` en la cabecera HTTP de cada petición, es decir, `Authorization: Bearer <ID Token>`. Si el token no estaba presente o era inválido, la operación no se ejecutaba. Dicho ID Token es el token JWT de sesión proporcionado por Firebase Authentication tras loguearte o crear cuenta por primera vez.

En cuanto al **modelado de recursos**, el acceso era mayoritariamente **individual** y separado por tipos de recurso, con **endpoints específicos para metas** y **endpoints específicos para sus registros**. Esta separación respondía al modelo de base de datos **relacional** de la etapa inicial, evitando agregaciones de gran tamaño y favoreciendo consultas dirigidas a cada recurso (por ejemplo, rutas para metas y rutas para registros asociadas a una meta concreta).

Para errores, se definieron las siguientes respuestas estándar:

- **401 Unauthorized**: cuando falla la autenticación (token ausente o inválido).
- **400 Bad Request**: ante peticiones malformadas (parámetros o cuerpo inválidos).
- **500 Internal Server Error**: para errores internos inesperados.

Para no sobrecargar esta sección, no se detalla aquí el diseño inicial de endpoints. A diferencia de otros modelos preliminares, su inventario es extenso y requeriría una atención que desviaría el foco del diseño final. El detalle completo (rutas y ejemplos) puede consultarse en el Anexo: Diseño de Endpoints Inicial de GoLife.

6.5.2. Impacto del cambio de base de datos en el diseño de la API

El paso de una base de datos relacional a una **no relacional basada en documentos** permitió **unir endpoints** que antes estaban separados. Al poder entregar al front end la misma o mayor información en un único documento, la aplicación realiza **menos llamadas** a la API del back end y la propia API necesita **menos tratamiento de datos**, sin composiciones ni agregaciones adicionales. El resultado fue una **mejora sustancial** a la hora de servir correctamente al front end.

Los **documentos diseñados** (expuestos en la Subsección 6.3.4) ya tienen una forma orientada a las necesidades del front end, por lo que sólo requieren **ajustes o filtros mínimos** para transmitir su información.

En resumen, el nuevo modelo nos permitió **centralizar funciones de endpoints** que antes estaban dispersas en varios, simplificando tanto el consumo desde el front end como la implementación en el back end.

La siguiente Tabla 6.2 presenta una vista resumida de los cambios sufridos por el diseño inicial de enpoints. Mientras que la Tabla 6.3 muestra un resumen de endpoints nuevos.

Inicial	Estado	Final / Observaciones
POST api/usuarios	Modificado	Cuerpos y respuestas adaptados
GET api/usuarios	Modificado	Cuerpos y respuestas adaptados
DELETE api/usuarios	Modificado	Respuestas adaptadas
POST api/metas	Eliminado	Endpoint dividido en 2
GET api/metas	Eliminado	Ya no es necesario
GET api/metas/full	Eliminado	Ya no es necesario
GET api/metas/activas	Eliminado	Ya no es necesario
GET api/metas/activas/full	Eliminado	Ya no es necesario
GET api/metas/inactivas	Eliminado	Ya no es necesario
PUT api/metas/{metaId}	Eliminado	Sustituido por PATCH
GET api/metas/{metaId}	Modificado	GET api/metas/{mid}, cuerpos y respuestas adaptados
DELETE api/metas/{metaId}	Modificado	DELETE api/metas/{mid}, cuerpos y respuestas adaptados
POST api/metas/{metaId}/finalizar	Modificado	Respuestas adaptadas
POST api/metas/{metaId}/registros	Modificado	POST api/metas/{mid}/registros?tipo=[Num,Bool], cuerpos y respuestas adaptados
GET api/metas/{metaId}/registros	Eliminado	Ya no es necesario

Tabla 6.2: Migración de endpoints (inicial → actual)

Actual	Motivo / Observaciones
PATCH api/usuarios	Modificación de datos de usuario
POST api/metas/bool	Separación de responsabilidades - meta booleana
POST api/metas/num	Separación de responsabilidades - meta numérica
PATCH api/metas/{mid}?tipo=[Num,Bool]	Edición de datos de meta (no sustitución completa)
DELETE api/metas/{mid}/registros/{fecha}	Eliminación de registros por fecha como su identificador (son únicos por día)

Tabla 6.3: Endpoints nuevos exclusivos del diseño final

6.5.3. Especificación final

La especificación **final** de la API del *back end* se ha elaborado con **OpenAPI** y se presenta mediante **Swagger UI**. En despliegue, la documentación es servida desde el propio *deployment* de la API y resulta accesible públicamente.
La siguiente Figura 6.12, muestra un resumen de los endpoints finales a través de la documentación de Swagger:

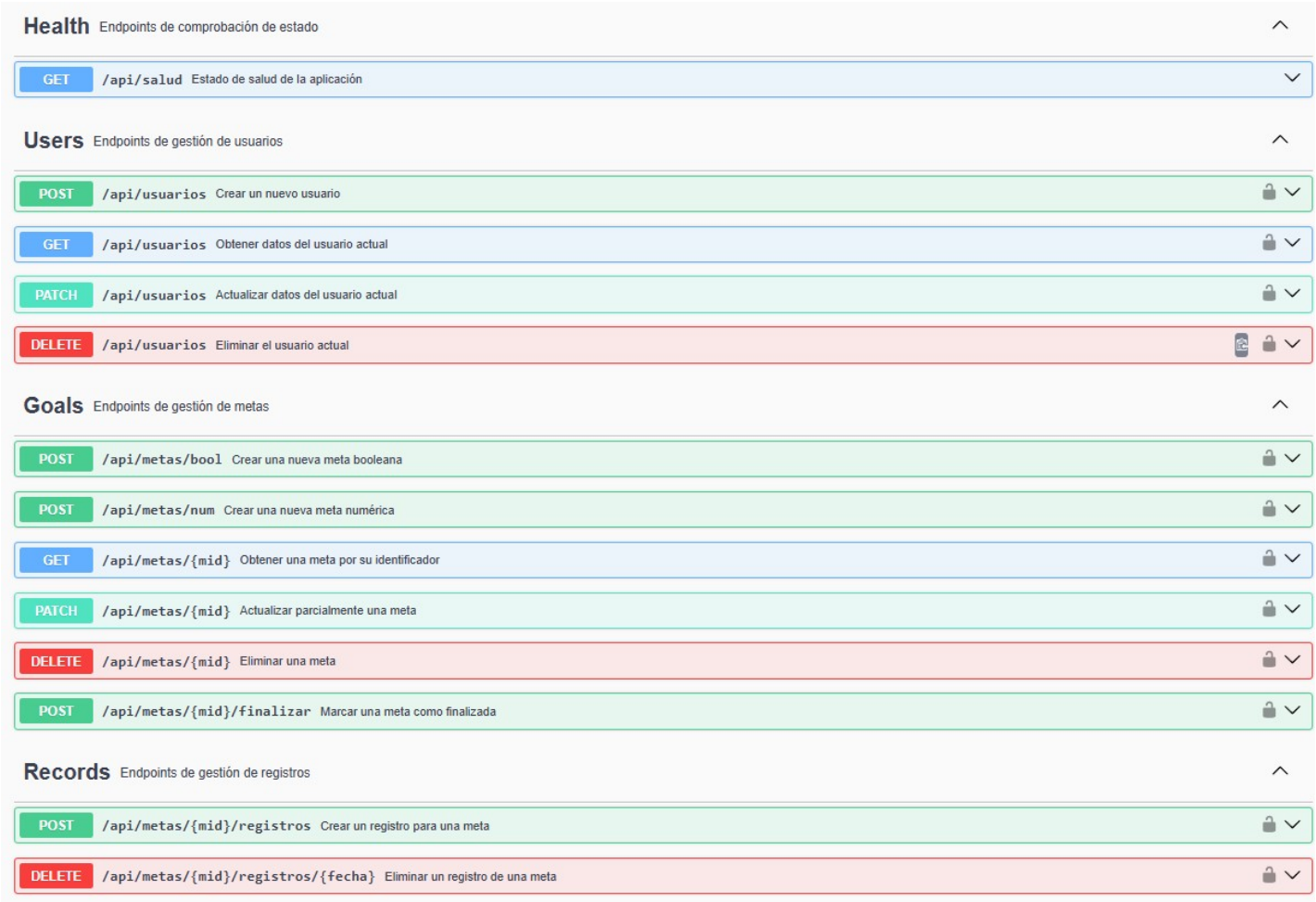


Figura 6.12: Resumen de endpoints de la documentación Swagger

Por limitaciones de continuidad del servicio tras la finalización del proyecto, se ofrece aquí una alternativa funcional aunque más limitada: un **HTML estático** de Swagger UI. Al ser estático, se pierden algunas capacidades interactivas, pero consideramos que sigue proporcionando un entendimiento suficiente de la API.

Figura 6.13: Documentacion de la API (Clickable)

La documentación recoge cada **endpoint** con sus métodos y descripciones, los **payloads** requeridos, los **mensajes de error** y las **respuestas** con ejemplos. Asimismo, incluye los **DTOs (Data Transfer Objects)** utilizados por la API, indicando sus atributos y restricciones. Las siguientes figuras presentan ejemplos de endpoints detallados en la documentación.

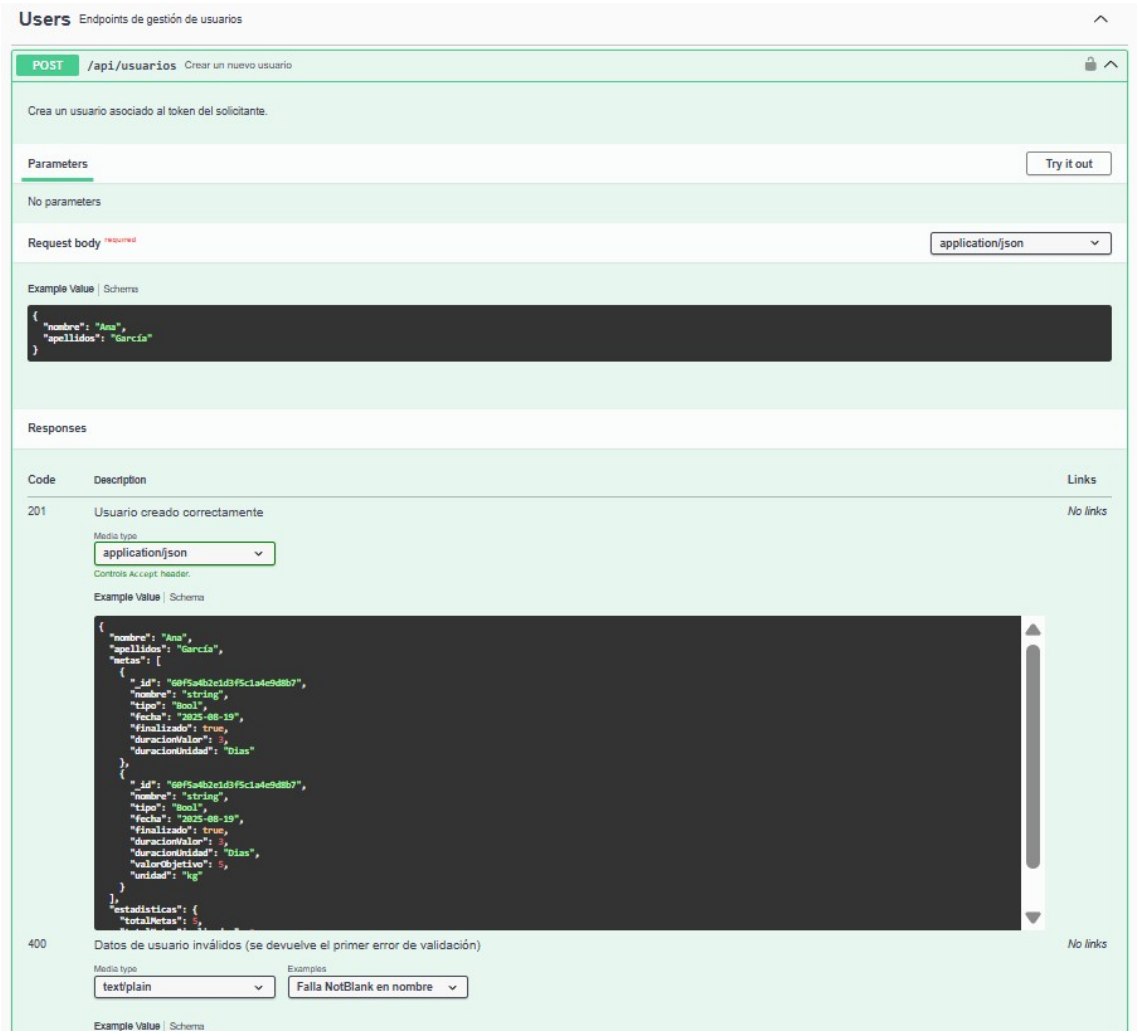


Figura 6.14: Endpoint de Usuarios de la documentación Swagger

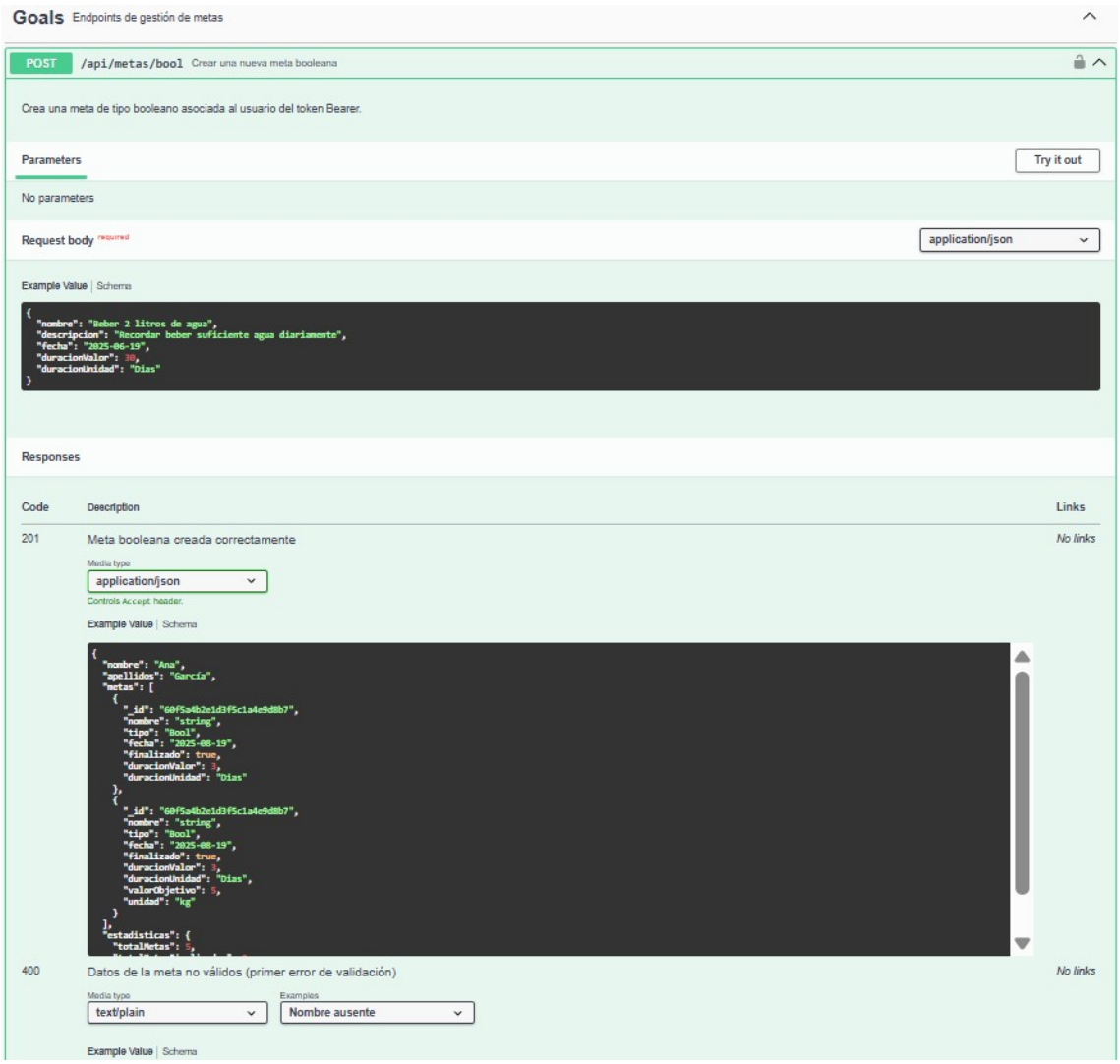


Figura 6.15: Endpoint de Metas de la documentación Swagger

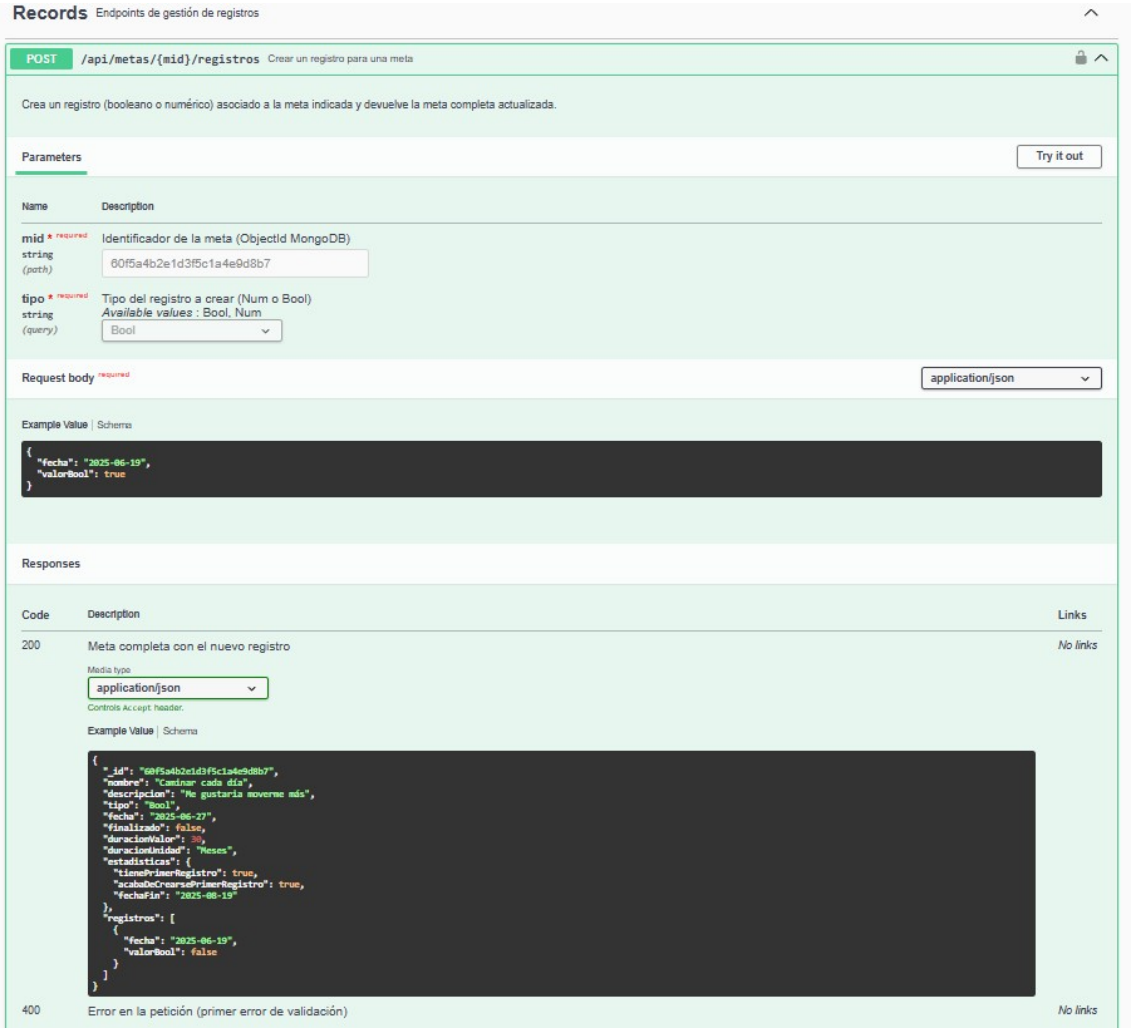


Figura 6.16: Endpoint de Registros de la documentación Swagger

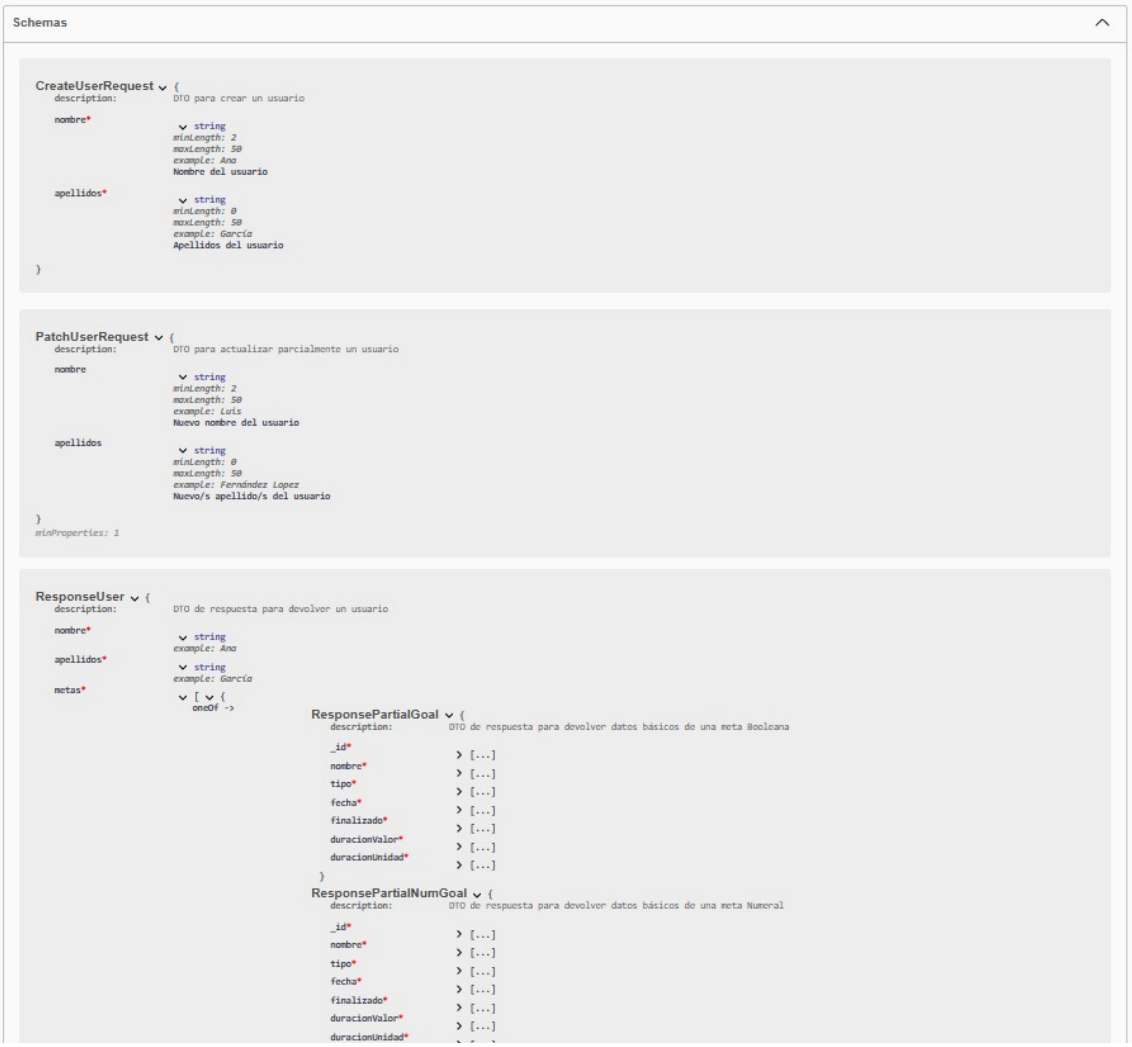


Figura 6.17: Esquemas (DTOs) en la documentación Swagger

Además de los endpoints definidos para la gestión de usuarios, metas y registros; la API incorpora un **endpoint de comprobación de salud** para verificar la conectividad con los servicios que utiliza (MongoDB Atlas, Firebase Authentication y Cloud KMS). Su objetivo es confirmar, de forma sencilla, que las dependencias externas siguen operativas.

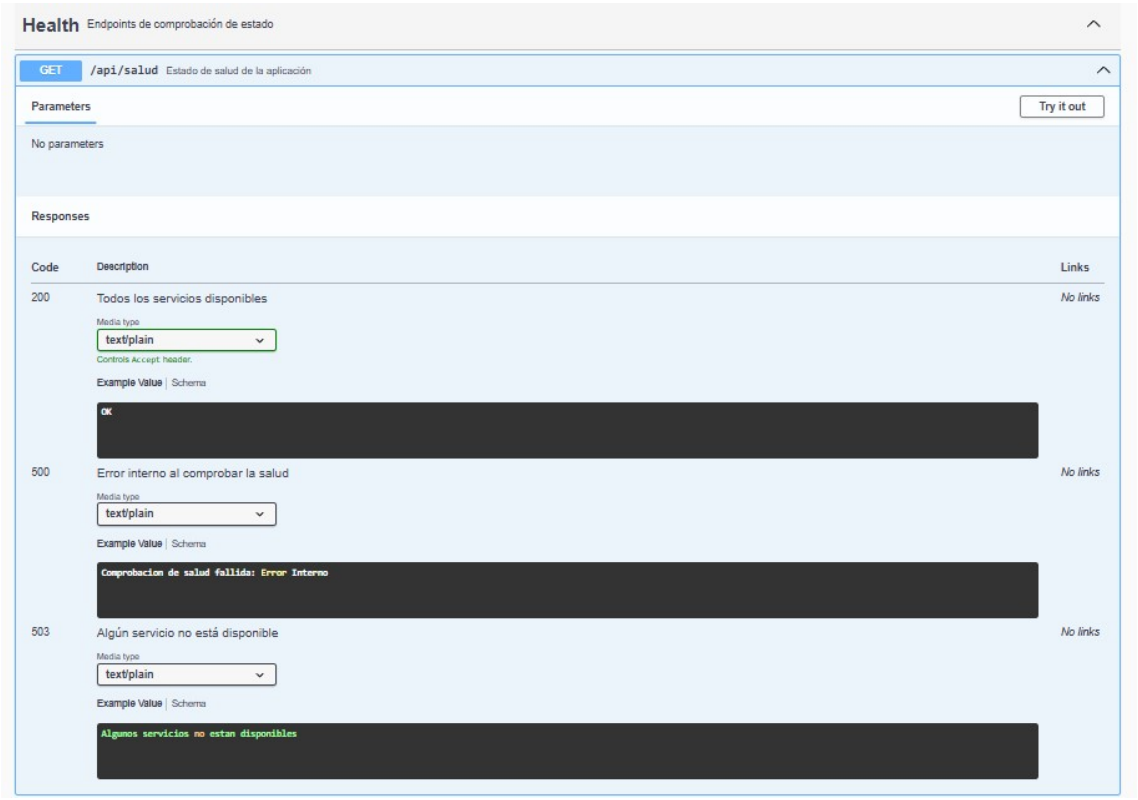


Figura 6.18: Endpoint de salud de la documentación Swagger

6.6. Diseño de Interfaz de Usuario (UI)

Un buen diseño de interfaz es clave para reducir fricción, guiar al usuario y sostener los flujos principales del sistema. Para ello elaboramos **wireframes en Figma** que sirvieron como guía del diseño conceptual de pantallas y navegación. El foco fue un **dashboard centralizado** desde el que gestionar y analizar las metas: desde ahí el usuario puede crear, ver, actualizar, finalizar o eliminar metas y registrar su progreso, obteniendo siempre información relevante de un vistazo.

6.6.1. Flujos de navegación

Siguiendo la **regla de los tres clics** (el usuario alcanza la información/acción crítica en ≤ 3 clicks), los wireframes definen estos recorridos:

1. **Acceso.** Figura 6.19 →
   - Si es la primera vez y hay que crear cuenta: Figura 6.20 (recogida de datos de usuario) → Figura 6.21.
   - Si ya está registrado: acceso directo a Figura 6.21.
2. **Dashboard como eje.** Desde Figura 6.21:
   - *Crear meta*: Figura 6.22 → regreso al Figura 6.21.

- *Acciones rápidas sobre una meta existente*: **editar** (Figura 6.24), **finalizar** (Figura 6.25), **crear registro** (Figura 6.27) o **eliminar** (Figura 6.26) *directamente* desde el Figura 6.21. Opcionalmente, *ver meta* (Figura 6.23) para detalle e histórico.

3. **Perfil.** Figura 6.21 → Figura 6.29 → vuelta al Figura 6.21. Desde Figura 6.29 es posible **eliminar la cuenta**.

Estos recorridos priorizan accesos directos desde el dashboard para que las acciones frecuentes y la información relevante estén disponibles en el mínimo número de interacciones.

6.6.2. Wireframes (visión estructural)

A continuación se incluyen los wireframes principales, centrados en la página, que ilustran la disposición y jerarquía de la información. No representan el estilo definitivo, sino el **modelo base** que orientó el desarrollo.

Comparten la estructura y los flujos con la interfaz final, pero ésta incorpora mayor claridad, elementos adicionales y opciones avanzadas. La UI definitiva se presenta en su sección correspondiente.

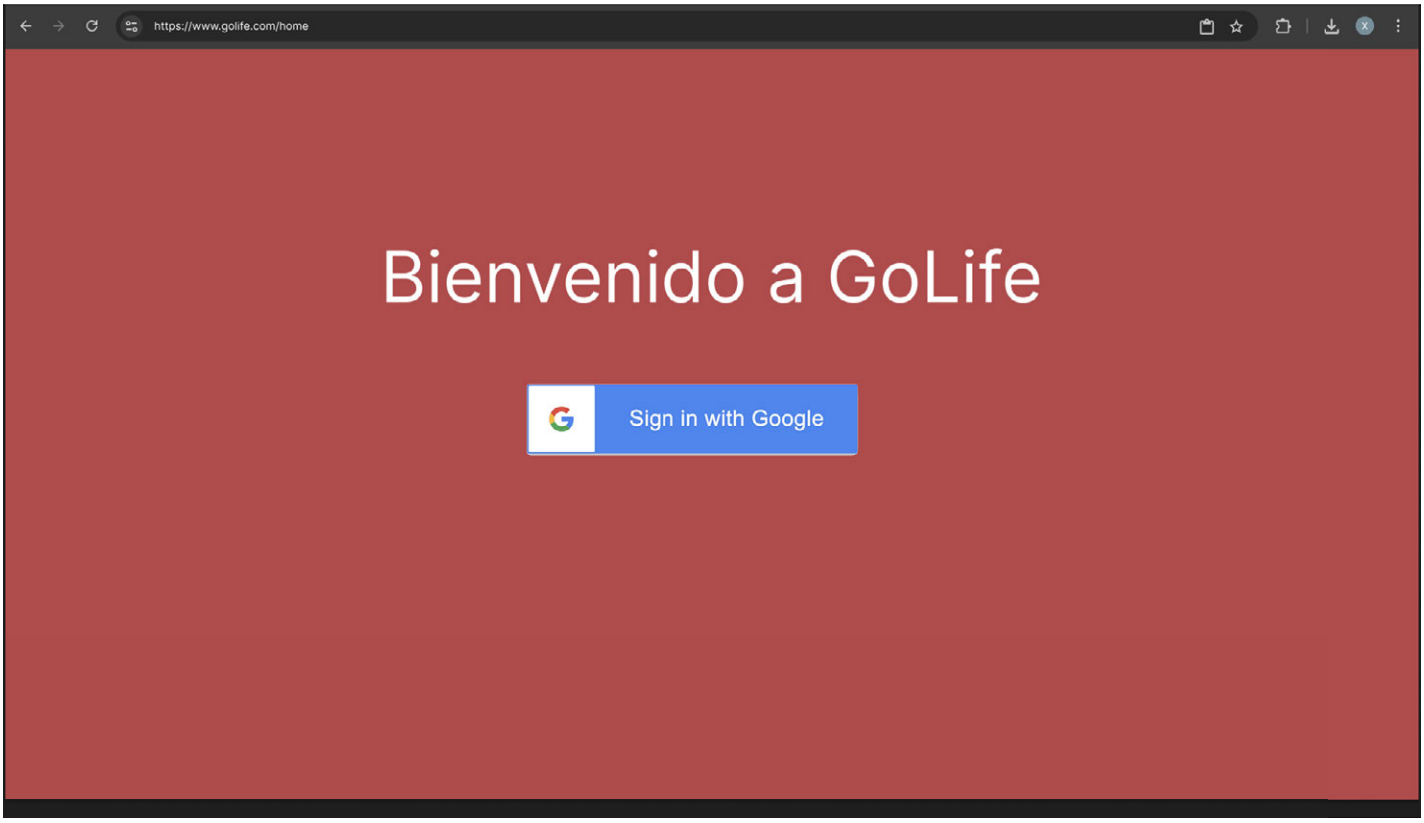


Figura 6.19: Wireframe - Login

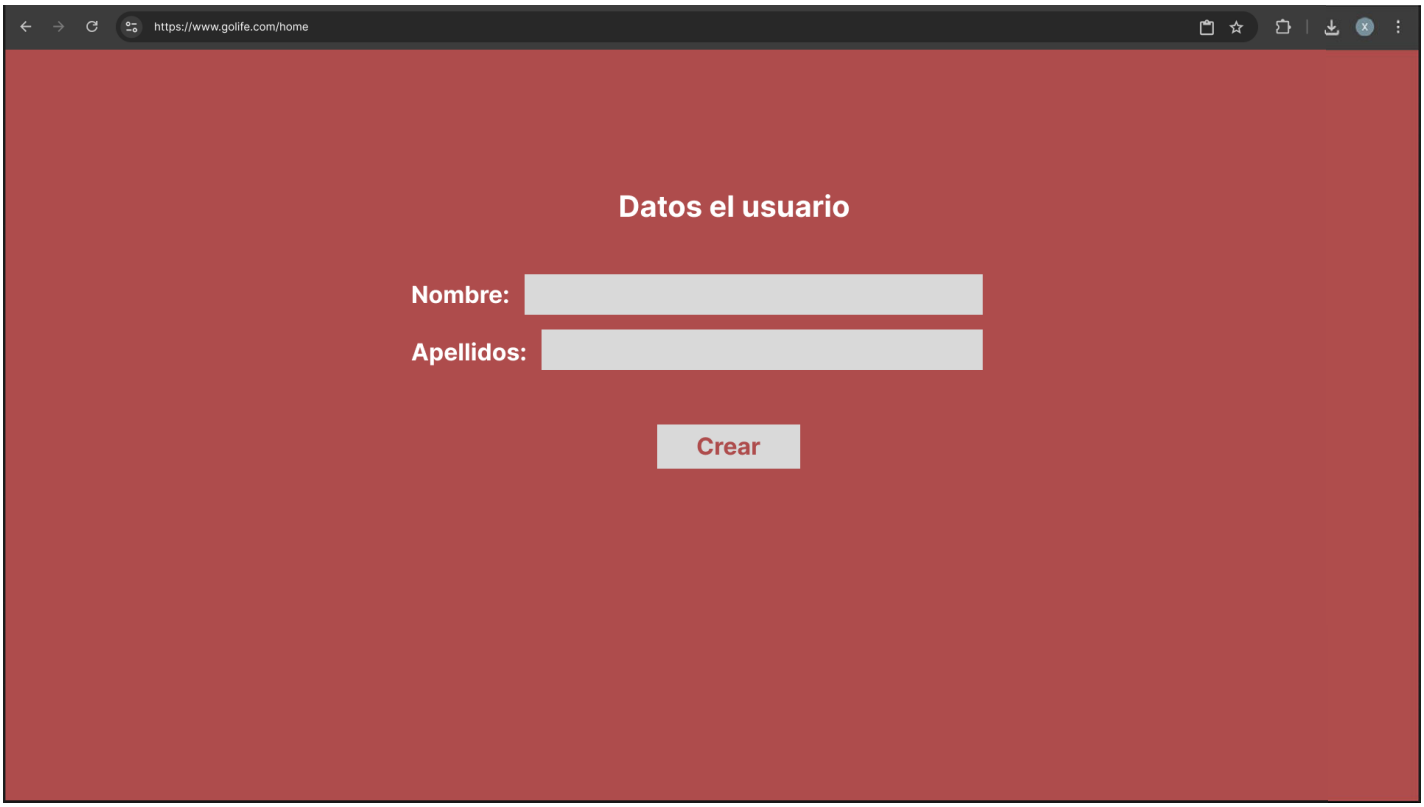


Figura 6.20: Wireframe - Primer login: recogida de datos de usuario

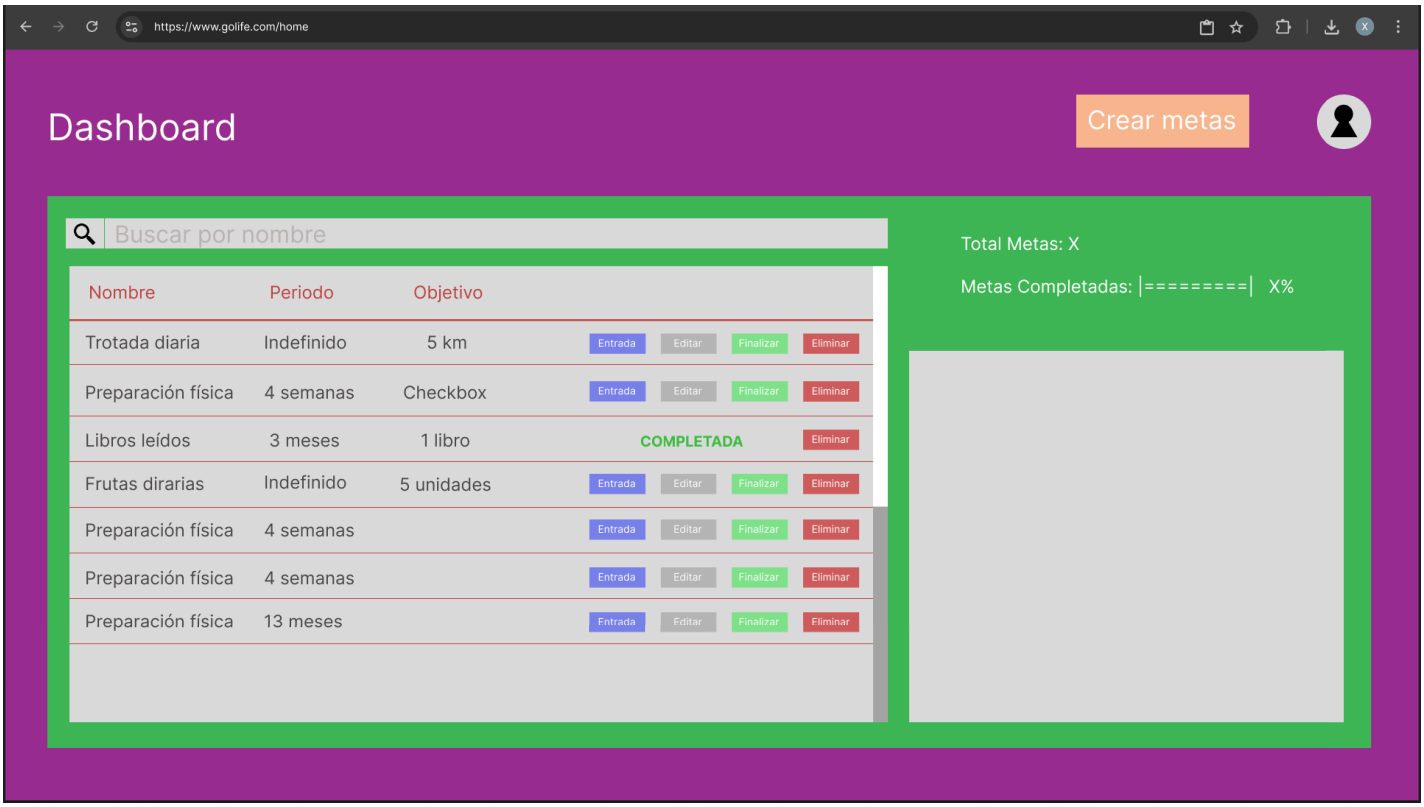


Figura 6.21: Wireframe - Dashboard con listado de metas y accesos a acciones

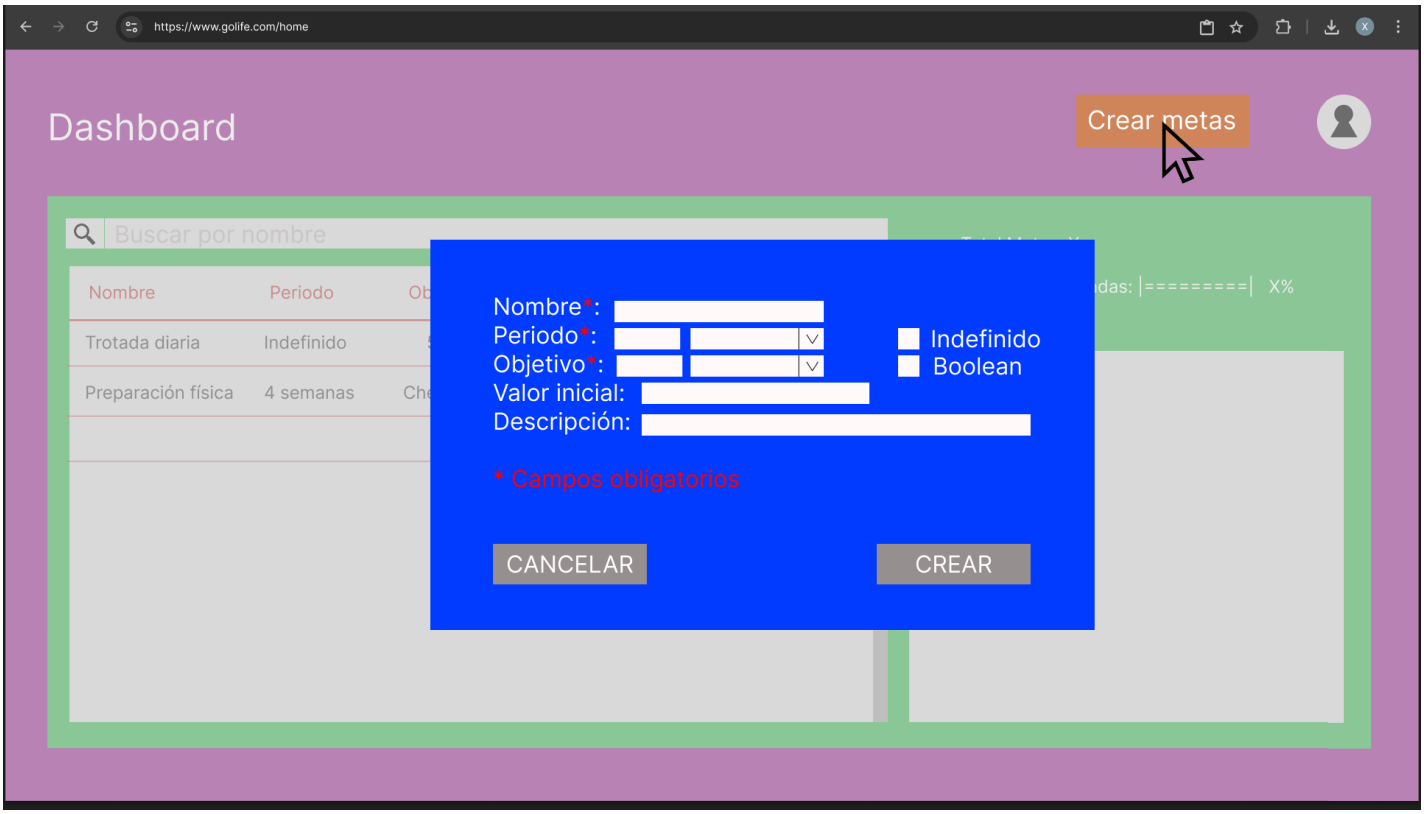


Figura 6.22: Wireframe - Crear meta: formulario básico

Figura 6.23: Wireframe - Ver meta: detalle y registros embebidos

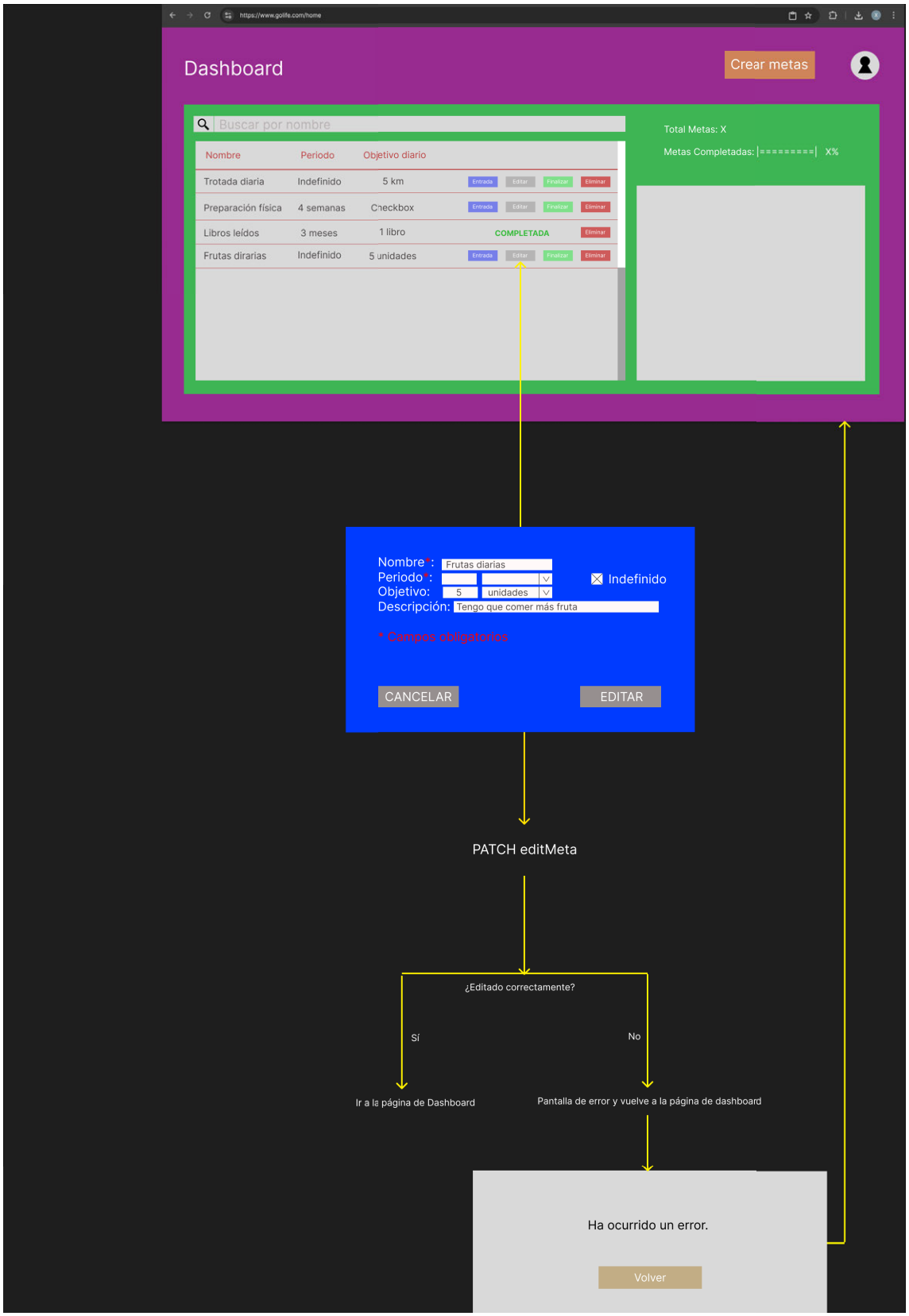


Figura 6.24: Wireframe - Editar meta: actualización de atributos

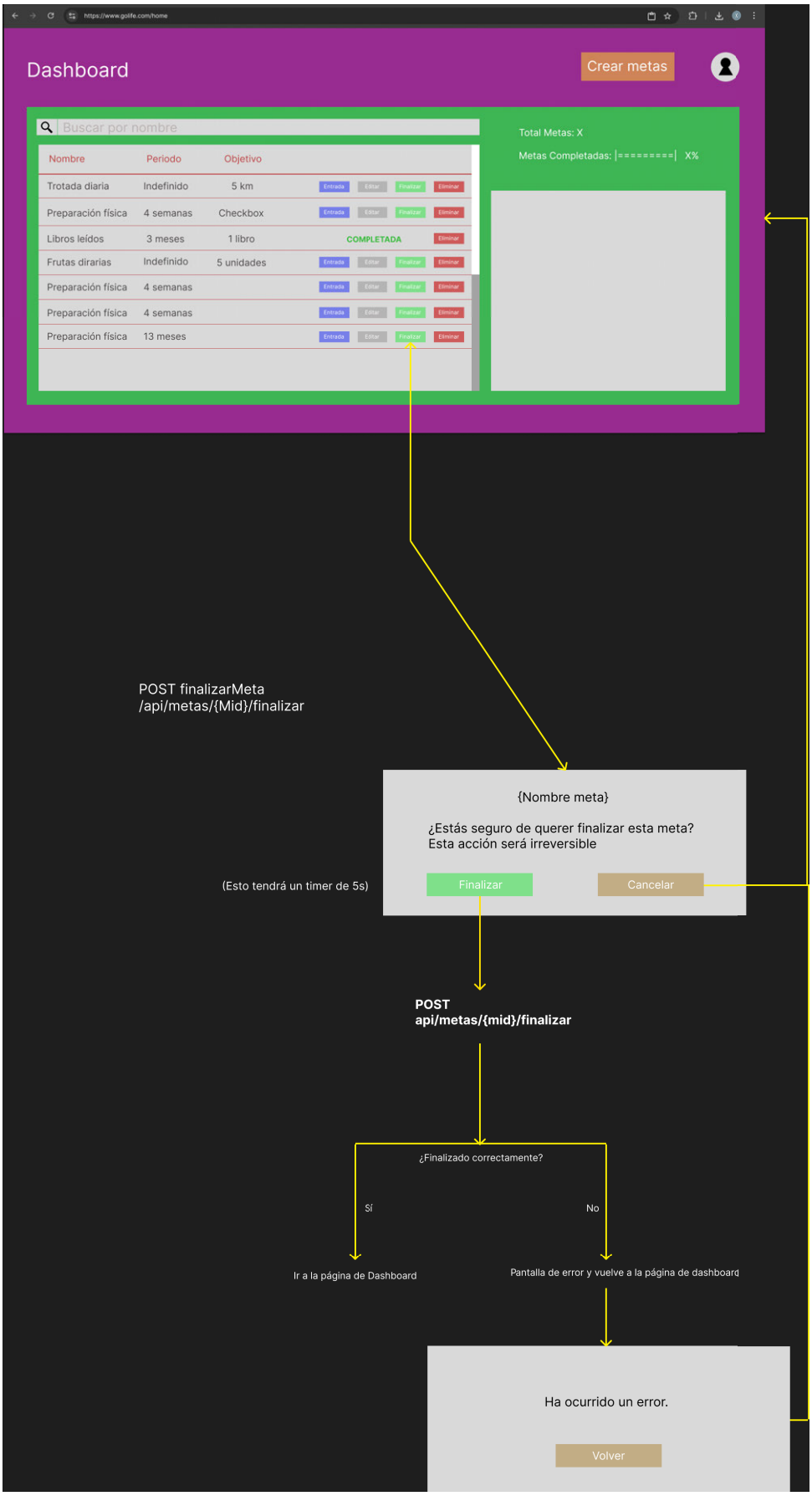


Figura 6.25: Wireframe - Finalizar meta: confirmación y estado

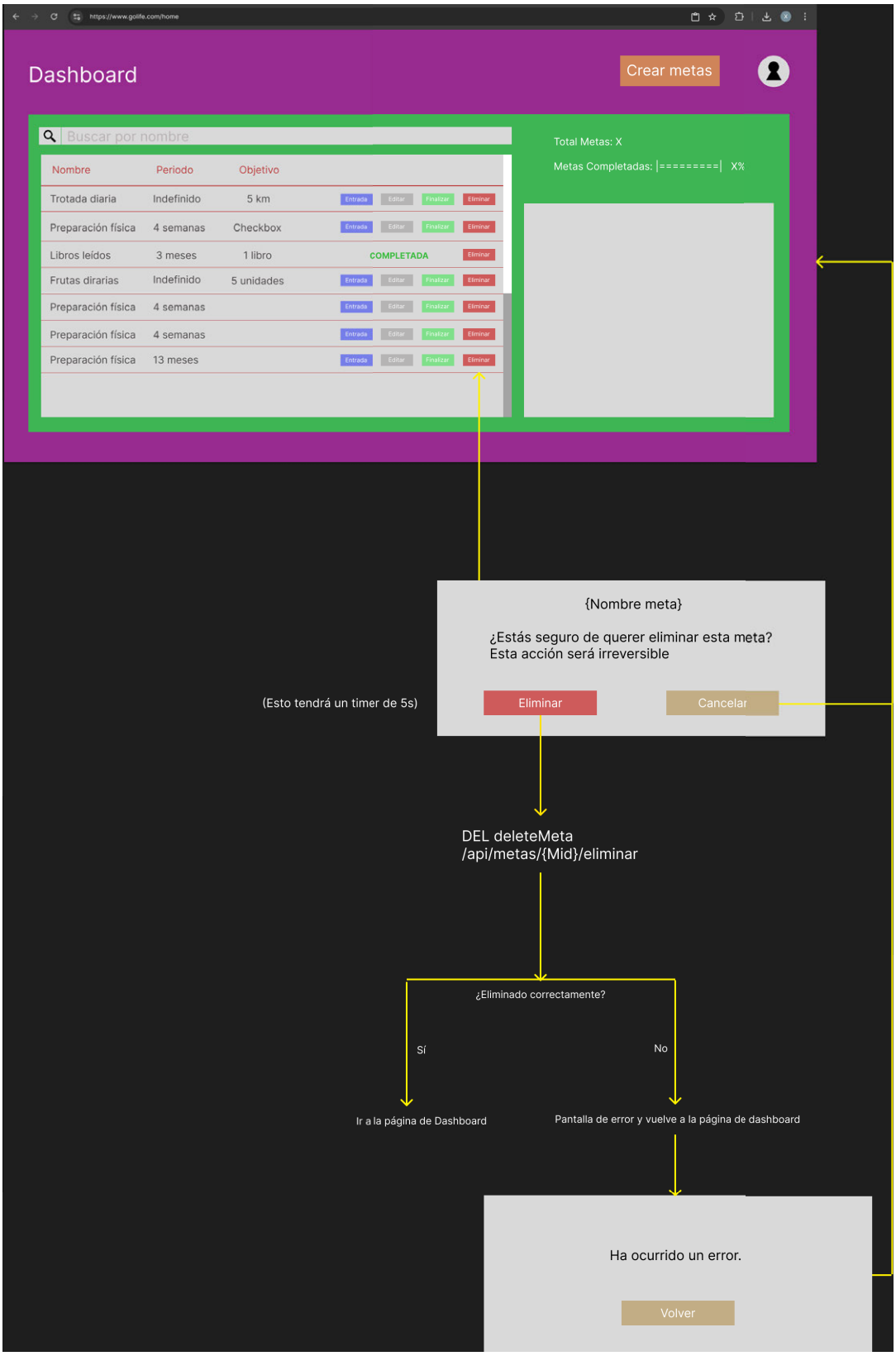


Figura 6.26: Wireframe - Eliminar meta: confirmación de borrado

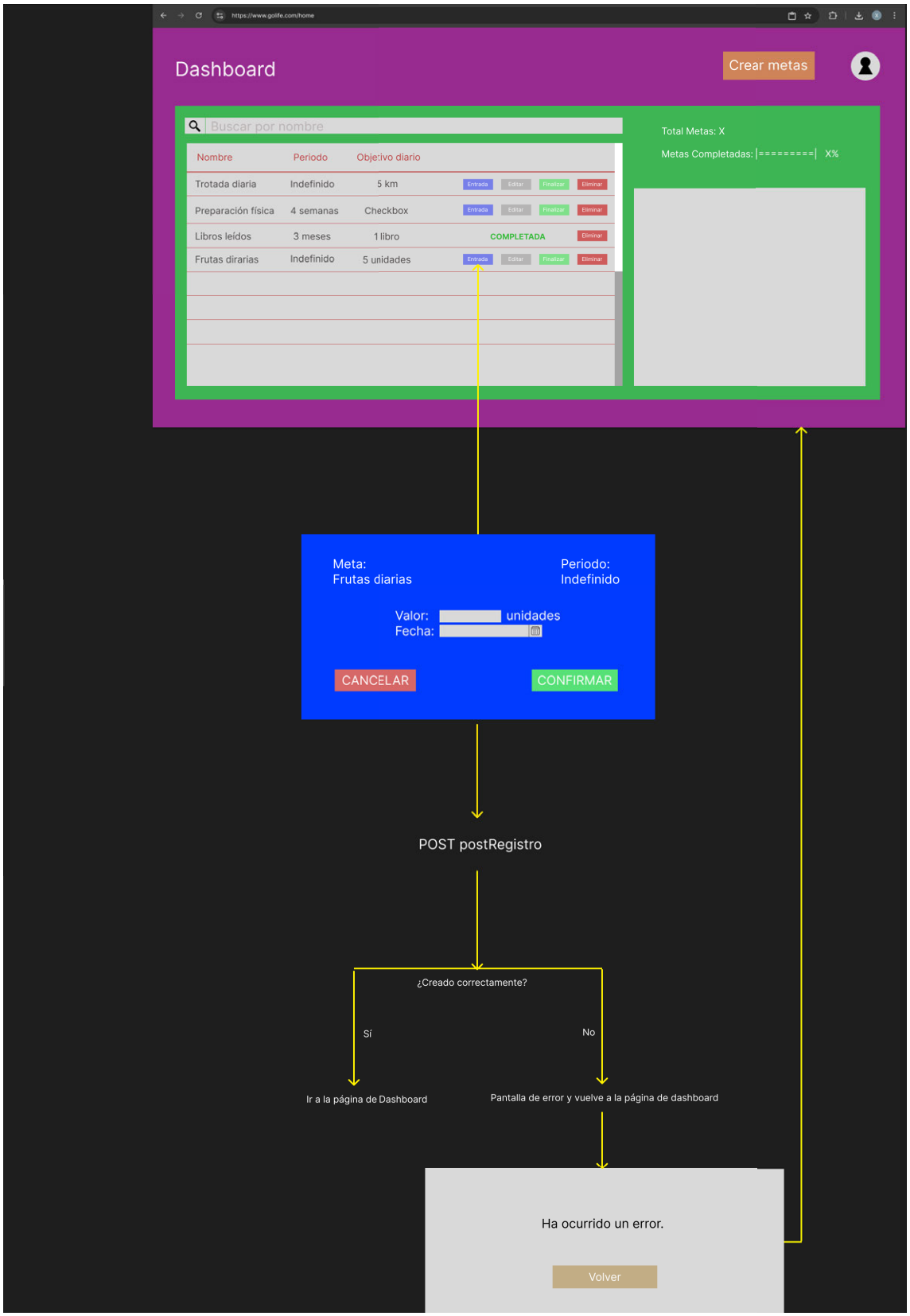


Figura 6.27: Wireframe - Crear registro: alta de progreso diario

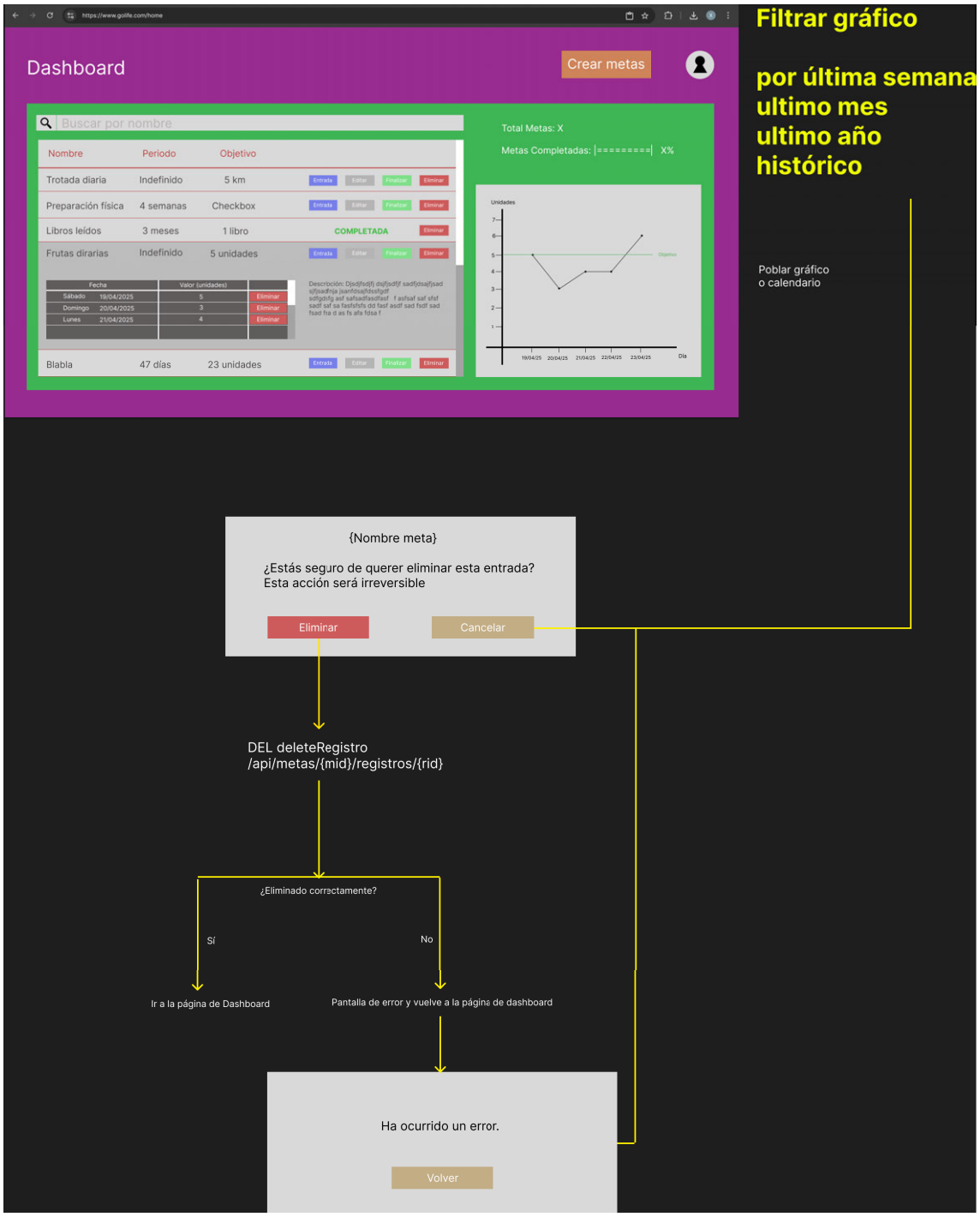


Figura 6.28: Wireframe - Eliminar registro: confirmación de borrado

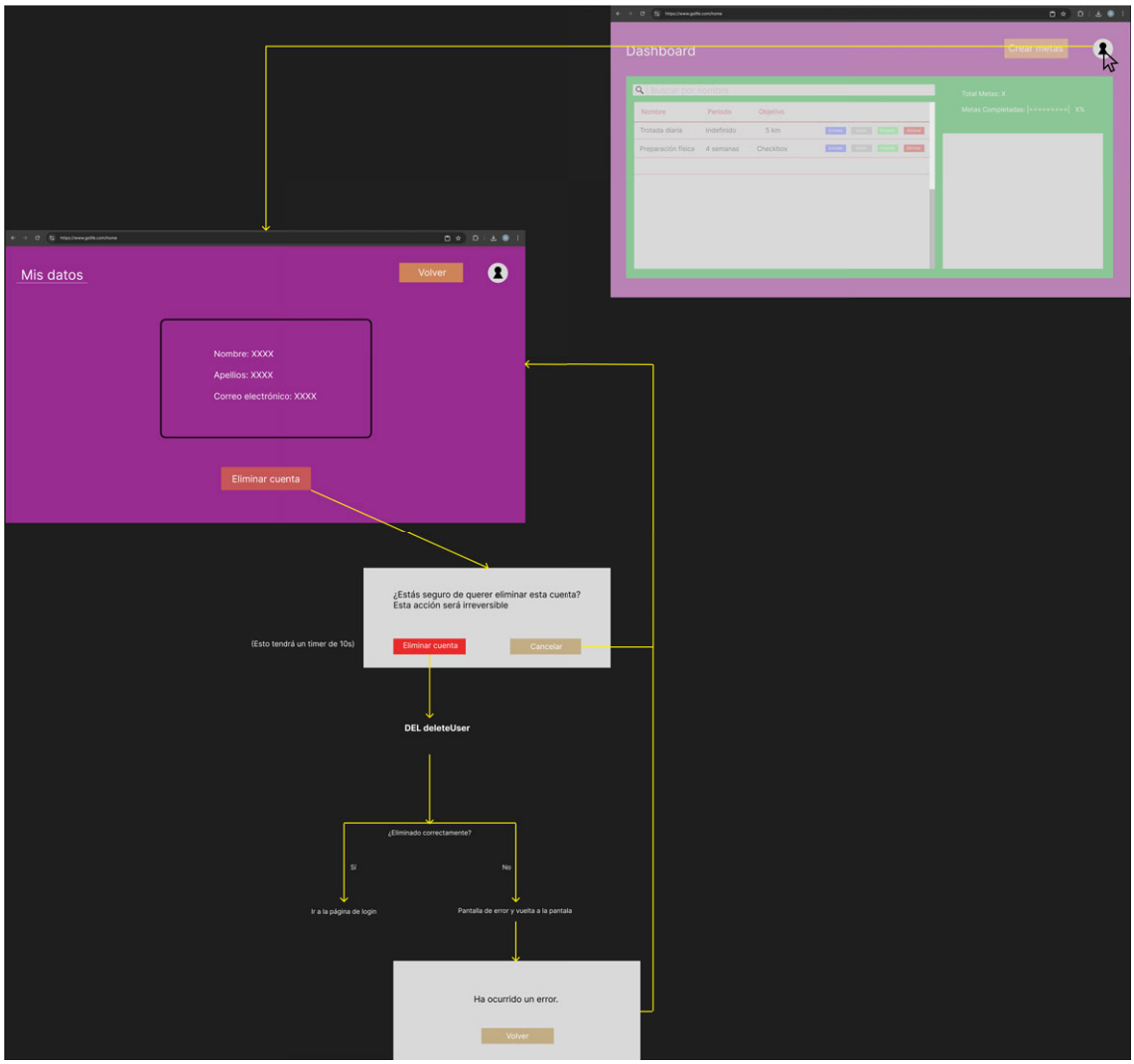


Figura 6.29: Wireframe - Perfil de usuario

Capítulo 7

Arquitectura del Sistema

7.1. Vista general de despliegue

En esta sección se presenta la **estructura de despliegue** de GoLife. Puede entenderse como una versión más detallada y operativa de la Figura 6.9, donde se mostraba la arquitectura final del sistema a nivel conceptual.

El diagrama se ha construido siguiendo la notación **C4** porque ofrece un equilibrio entre simplicidad y precisión, es agnóstica de herramientas y centra la conversación en responsabilidades y relaciones reales del sistema, facilitando mostrar la visión mismo (98).

C4 es un **modelo para visualizar arquitectura software** mediante un conjunto de diagramas jerárquicos que describen el contexto del sistema, sus contenedores ejecutables, los componentes internos y, finalmente, cómo todo ello se despliega en infraestructura (98). Para interpretar la vista de despliegue, conviene aclarar dos conceptos (99):

- **Deployment Node** (Nodo de despliegue): un entorno de ejecución **donde se aloja software** (por ejemplo, una máquina física/virtual, un servicio gestionado en la nube, un clúster o una plataforma de contenedores). Los nodos pueden anidarse para reflejar niveles de infraestructura.
- **Container (C4)**: una **unidad ejecutable/desplegable** de software (p. ej., una API web, una SPA, una base de datos). En C4 "container" no implica necesariamente Docker; describe la unidad lógica que se ejecuta dentro de uno o varios nodos.

La siguiente Figura 7.1 muestra el diagrama de despliegue a nivel de componentes y servicios, incluyendo dependencias y los principales canales de comunicación.

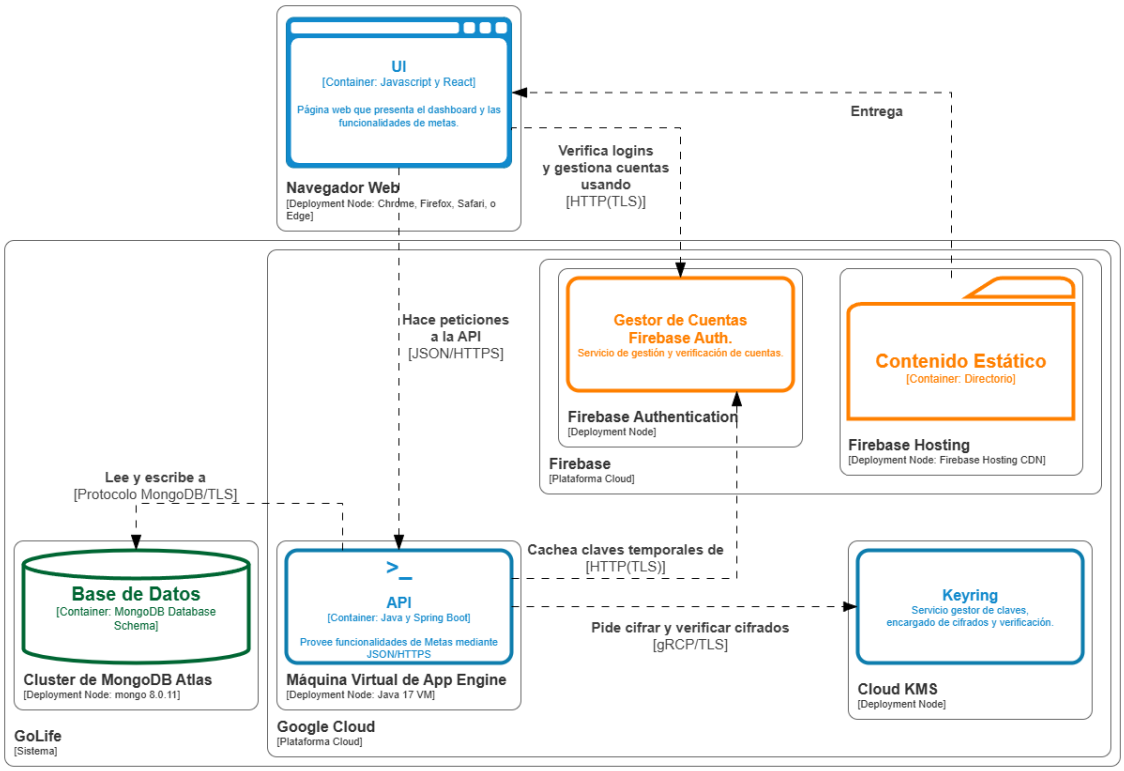


Figura 7.1: Diagrama de Despliegue de GoLife

Descripción de Nodos de Despliegue

- **Navegador Web:** Punto de acceso del usuario y entorno de ejecución de la interfaz del front end.
- **Firebase Hosting:** Servicio gestionado que entrga la interfaz de usuario desplegada a través de una **CDN** (*Content Delivery Network*). Una CDN distribuye copias de los archivos en *puntos de presencia* cercanos a los usuarios; las solicitudes se sirven desde el nodo más próximo, reduciendo latencia y descarga del origen.
- **Firebase Authentication:** Servicio gestionado (no desplegamos software propio).
- **Máquina Virtual de App Engine:** La API se **ejecuta en instancias gestionadas** por App Engine (VMs que Google provisiona/escala). Los despliegues crean versiones, y App Engine enruta tráfico a la versión activa.
- **Cloud KMS:** Servicio gestionado (no desplegamos software propio).
- **Cluster de MongoDB Atlas:** Base de datos **gestionada** desplegada como *réplica*. **No hay un "servidor único"** al que nos unamos, la conexión redirige las operaciones al nodo adecuado.

7.2. Topología en GCP

En **Google Cloud Platform (GCP)** se han definido **tres proyectos principales**, con una separación clara de responsabilidades:

- **GoLife** - alberga el *front end* servido mediante **Firebase Hosting**.
- **GoLifeAPI** - proyecto *back end* que ejecuta la **API** en **App Engine**.
- **GoLifeAuth** - proyecto *back end* orientado a **servicios de autenticación y claves**: **Firebase Authentication** y **Cloud KMS**.

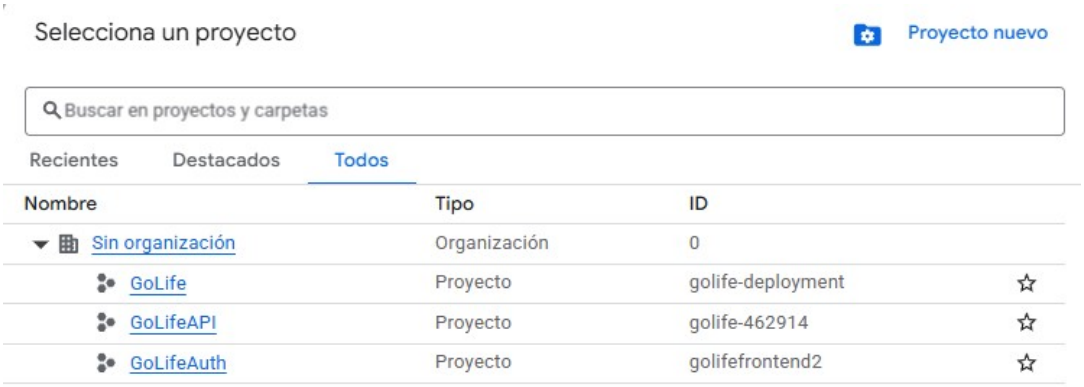


Figura 7.2: Proyectos de GCP del entorno GoLife

Esta división de componentes y servicios entre proyectos es deliberada, y es debido a los siguientes tres motivos principales:

- **Gestión:** Cada ámbito (front end, API y autenticación/claves) se administra de forma independiente, con configuración, cotas y ciclos de despliegue propios. Esto simplifica el gobierno del ciclo de vida y la trazabilidad de cambios por dominio.
- **Mantenibilidad:** La división reduce el acoplamiento operativo y facilita diagnósticos: incidencias del front end no afectan al pipeline de la API, y la rotación/custodia de claves se trata sin interferir con el código de negocio.
- **Seguridad:** La segmentación por proyecto limita el radio de impacto y permite aplicar principio de mínimo privilegio: la cuenta de servicio de la API sólo obtienen permisos explícitos para usar claves en GoLifeAuth cuando es necesario (acceso cross-project), mientras que el proyecto de hosting no tiene acceso a datos ni a KMS.

En cuanto a **regiones/ubicaciones**, los recursos de Google Cloud con selección de región se han desplegado en `europe-west6` (App Engine y Cloud KMS) y la base de datos en `europe-west1`. La elección responde a **criterios de coste** y a mantener cercanos los componentes principales para **minimizar la latencia** entre API y Cloud KMS, manteniendo la base de datos dentro de la UE con latencia intraeuropea asumible. La región `europe-west6` se ubica en **Zúrich (Suiza)** y `europe-west1` en **St. Ghislain (Bélgica)** (100).

El servicio de **Firebase Authentication**, por **limitaciones operativas y de gestión** de Google, opera y almacena sus datos **exclusivamente en Estados Unidos**; es decir, el procesamiento vinculado a alta/login y a la emisión/renovación de ID Tokens se realiza en centros de datos ubicados en EE. UU. (101).

7.3. Identidad y Acceso (IAM)

IAM (Identity and Access Management) de Google Cloud es el sistema que permite definir **quién** (identidad) puede **hacer qué** (permisos) **sobre qué** recurso, mediante la asignación de **roles** a **miembros** en políticas de acceso (102).

Las **cuentas de servicio** son identidades no humanas que representan a aplicaciones y cargas de trabajo (p. ej., una API en App Engine). Estas cuentas obtienen **credenciales** para autenticarse frente a otros servicios y se les otorgan **roles IAM** que determinan exactamente qué operaciones pueden realizar (por ejemplo, usar una clave en Cloud KMS) (103).

En esta sección se presentan las **cuentas de servicio** y **permisos IAM** configurados para que GoLife funcione como un sistema integrado: qué identidad usa cada componente, qué roles tiene y en qué proyecto, y cómo se orquesta el acceso cross-project cuando procede.

7.3.1. GoLife

El proyecto **GoLife** aloja el front end (Firebase Hosting). A nivel de IAM cuenta con las siguientes identidades y permisos, orientados a servir estáticos y automatizar despliegues, manteniendo la gestión humana bajo una única cuenta propietaria (misma para todos los proyectos):

- **Cuenta de servicio:** `firebase-adminsdk-fbsvc@...` (gestionada por Firebase)
  - **Servicio:** integración de Firebase en el proyecto.
  - **Roles:**
    - Administrador de Firebase Authentication
    - Agente de servicios del administrador del SDK de Firebase Admin
    - Creador de tokens de cuenta de servicio
  - **Propósito:** permitir a servicios de Firebase ejecutar operaciones administrativas vinculadas al proyecto.
- **Cuenta de servicio:** `github-action-10020263332@...` (pipeline de GitHub Actions para front end)
  - **Servicio:** CI/CD del sitio estático.
  - **Roles:**
    - Administrador de Firebase Hosting
    - Administrador de Firebase Authentication
    - Consumidor de Service Usage
    - Desarrollador de Cloud Functions
    - Lector de Cloud Run
    - Visualizador de claves de API
  - **Propósito:** desplegar en Hosting y consultar/usar servicios necesarios durante el pipeline.
- **Cuenta humana:** `golifepfg@gmail.com`

- **Rol:** Propietario (gestión global del proyecto).

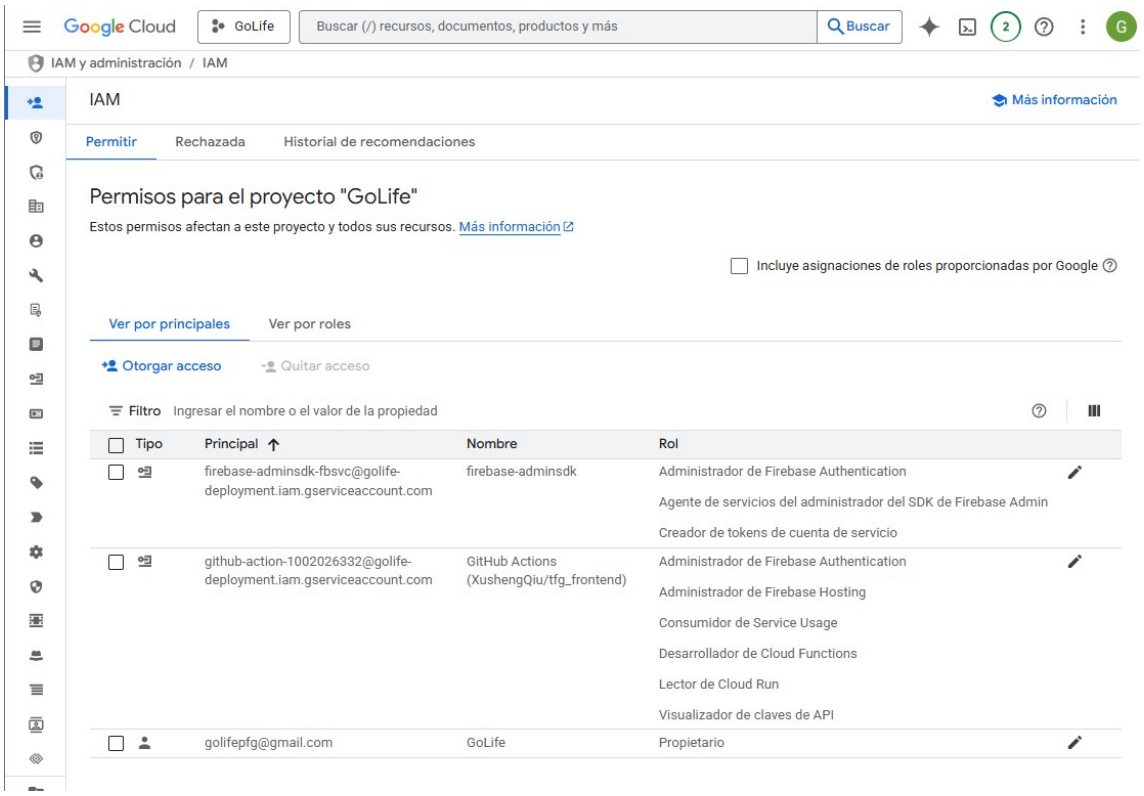


Figura 7.3: IAM de GoLife: Cuentas de servicio y permisos

7.3.2. GoLifeAPI

El proyecto **GoLifeAPI** aloja la API back end desplegada en **App Engine**. Las identidades y permisos de IAM visibles son:

- **Cuenta de servicio:** `github-deployer@...` (pipeline de GitHub Actions para desplegar en App Engine)
  - **Servicio:** CI/CD de la API.
  - **Roles:**
    - Administrador de App Engine
    - Implementador de App Engine
    - Administrador de almacenamiento
    - Editor
    - Usuario de cuenta de servicio
  - **Propósito:** compilar y desplegar de versiones de App Engine y publicar artefactos en el *bucket* de almacenamiento asociado.
- **Cuenta de servicio:** `golife-462914@appspot.gserviceaccount.com` (App Engine default service account)

- **Servicio:** identidad de ejecución de la API en App Engine.
- **Roles:**
  - Administrador de App Engine
  - Administrador de almacenamiento
  - Editor
  - Usuario de cuenta de servicio
- **Propósito:** ejecutar la aplicación y acceder a recursos del proyecto necesarios para el servicio.

- **Cuenta humana** `golifepfg@gmail.com`
  - **Rol:** Propietario (gestión global del proyecto).

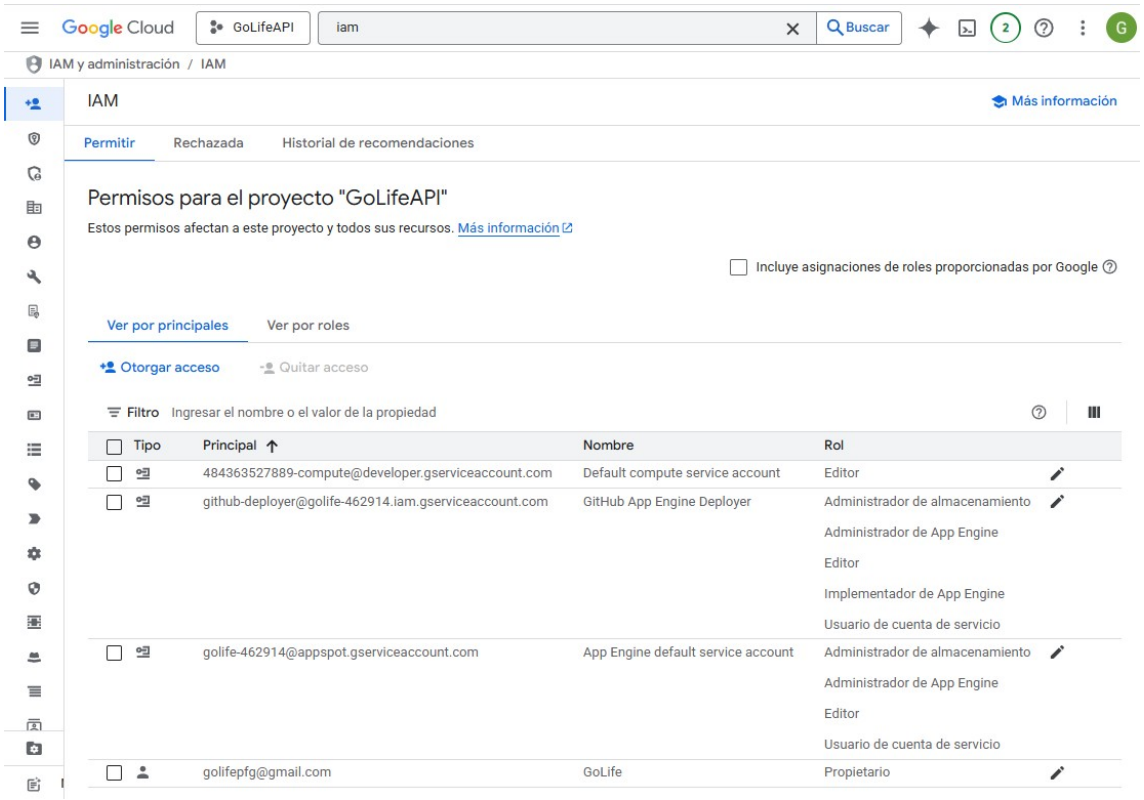


Figura 7.4: IAM de GoLifeAPI: Cuentas de servicio y permisos

7.3.3. GoLifeAuth

El proyecto **GoLifeAuth** concentra los servicios de autenticación y custodia de claves. Sus identidades y permisos IAM se organizan así:

- **Cuenta de servicio:** `firebase-adminsdk-fbsvc@...` (gestionada por Firebase)
  - **Servicio:** soporte del Admin SDK/servicios de Firebase en el proyecto.
  - **Roles:**
    - Administrador de Firebase Authentication

  - Agente de servicios del administrador del SDK de Firebase Admin
  - Creador de tokens de cuenta de servicio
- **Propósito:** operaciones administrativas de Firebase Authentication y tareas internas del Admin SDK.

- **Cuenta de servicio:** `golife-462914@appspot.gserviceaccount.com` (App Engine default service account - proyecto *GoLifeAPI*)
  - **Servicio:** identidad de ejecución de la API que necesita operar contra Auth/KMS en este proyecto (*acceso cross-project*).
  - **Roles:**
    - Administrador de Firebase Authentication
    - Verificador y firmante de *CryptoKeys* de Cloud KMS
    - Visualizador de Cloud KMS
  - **Propósito:** gestionar cuentas de Firebase Auth (principalmente borrado) y **firmar/verificar (MAC)** en KMS para seudonimización, con visibilidad mínima sobre recursos de KMS.
- **Cuenta humana** `golifepfg@gmail.com`
  - **Rol:** Propietario (gestión global del proyecto).

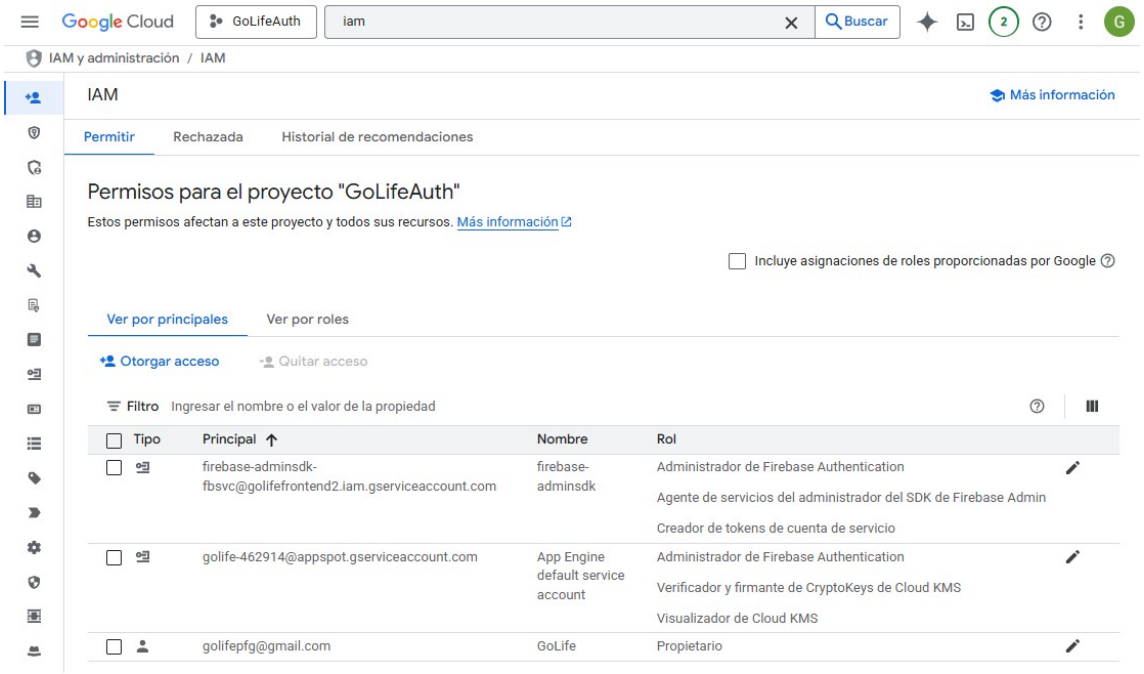


Figura 7.5: IAM de GoLifeAuth: Cuentas de servicio y permisos

7.4. Modelo de comunicaciones y protocolos

En esta sección describimos **cómo se comunican los componentes entre sí**, qué **protocolos** emplean y las **reglas** que rigen cada conexión.

Supuestos: no se utilizan redes privadas (VPC/peering/VPN).

7.4.1. Comunicación del front end (UI) con Firebase Auth.

- **Canal:**
  El front end usa el **Firebase Web SDK (versión modular)** para comunicarse con Firebase Authentication por **HTTPS/TLS**. Las funciones del SDK (p. ej., alta, login, obtención de tokens) llaman a los endpoints gestionados de Firebase sin que el cliente implemente protocolos de bajo nivel (44).
- **Auth:**
  Tras el inicio de sesión, el SDK obtiene un **ID Token (JWT)** y un *refresh token*. El SDK gestiona su **renovación automática** cuando expira (mediante el *refresh token*) (44).
- **Políticas:**
  Se configuran **dominios autorizados** (desde qué orígenes web puede usarse Auth) y la **persistencia de sesión** del SDK (local, sesión o memoria) según las necesidades del proyecto y del navegador (44).
- **Seguridad:**
  El intercambio de credenciales y la obtención del ID Token se realiza siempre **cifrado** (HTTPS/TLS). Las credenciales sensibles del usuario se envían *directamente* a Firebase Auth.(44).
- **Observación:**
  Una vez autenticado, el cliente puede **obtener el ID Token vigente** mediante el SDK cuando lo necesita; el propio SDK monitoriza el estado de autenticación y refresca el token sin intervención manual (44).

7.4.2. Comunicación del front end (UI) con la API

- **Canal:**
  El front end se sirve en `https://golife-deployment.web.app` y consume la API en `https://golife-462914.oa.r.appspot.com`. Toda la comunicación es **HTTPS/TLS** con certificados gestionados por Google. Ejemplo:

  ```
  GET https://golife-462914.oa.r.appspot.com/api/usuarios
  ```

- **Auth:**
  Cada petición incluye el *ID Token (JWT)* de Firebase Authentication en:

  ```
  Authorization: Bearer <ID Token>
  ```

  Si el token falta o es inválido, la API **rechaza** la petición.
- **Políticas:**
  CORS restringe llamadas del navegador al origen permitido `https://golife-deployment.web.app`. Cuando aplica, el navegador hace `OPTIONS` (*preflight*) y la API responde con `Access-Control-Allow-Origin` y métodos/cabeceras permitidas (p. ej., `Authorization`, `Content-Type`). Peticiones desde otros orígenes quedan bloqueadas por el navegador.
- **Seguridad:**
  Tráfico **cifrado en tránsito** (HTTPS/TLS) extremo a extremo; no se aceptan contenidos mixtos ni orígenes no autorizados. El uso de `Bearer` colabora a corroborar el origen y veracidad de las peticiones y los datos proporcionados a la API.

- **Observación:**
  Se implementa rate limiting mediante un algoritmo tipo **Token Bucket** para contabilizar por sesión/usuario y *throttlear* el exceso, evitando sobrecargas y picos contra la base de datos. Tras validar el `Bearer`, la API puede rechazar peticiones si un usuario excede el umbral de peticiones por minuto.

7.4.3. Comunicación de la API con Firebase Auth.

- **Canal:**
  La API usa el **Firebase Admin SDK** para Java (62) y descarga puntualmente las **claves públicas** de Google desde el endpoint:

  `/robot/v1/metadata/x509/securetoken@system.gserviceaccount.com`

  siempre por **HTTPS/TLS**. Estas claves se **cachean** respetando `Cache-Control: max-age` (104).

- **Auth:**
  No hay llamada a Firebase Authentication por petición. La verificación del *ID Token (JWT)* se realiza **localmente** en el servidor usando el Admin SDK, conforme a la guía oficial (104; 61).

- **Políticas:**
  Renovación **automática** del juego de claves cuando expira la caché; **timeouts** de red breves en la obtención inicial/renovación. El resto de validaciones no requiere red.

- **Seguridad:**
  El intercambio de claves públicas ocurre sobre **TLS**; los tokens inválidos se **rechazan** en servidor siguiendo las recomendaciones oficiales (104).

- **Observación:**
  En operación normal, la única conectividad necesaria es la **descarga/renovación** de claves públicas; la validación de cada petición es **local**, lo que reduce latencia y acoplamiento con el servicio de autenticación (104; 61).

7.4.4. Comunicación de la API con Cloud KMS

- **Canal:**
  La API usa el **cliente oficial de Java** y abre un canal **gRPC** hacia los endpoints regionales de KMS; todas las llamadas van **cifradas por TLS** (gRPC sobre HTTP/2) (60; 105; 106).

- **Auth:**
  Se emplean **Application Default Credentials (ADC)** de la cuenta de servicio para obtener automáticamente un token OAuth 2.0 y adjuntarlo en cada llamada gRPC; no se gestionan secretos en el código (106; 105).

- **Políticas:**
  Timeouts (*deadlines*) por llamada y **reintentos** con *exponential backoff* ante fallos transitorios (p. ej., `UNAVAILABLE`, `DEADLINE_EXCEEDED`); el cliente **reutiliza conexiones** gRPC/HTTP2 para evitar costes de establecimiento repetidos (105; 106).

- **Seguridad:**
  Las **claves nunca abandonan KMS**; sólo se devuelven resultados (MAC o verificación). Todo el tráfico está **cifrado en tránsito** (TLS) y el acceso se controla por IAM con roles mínimos (p. ej., *signer/verifier*) (105).

- **Observación:**
  Las claves residen en `europe-west6` (la ubicación va en el *resource name*); mantener API y KMS en la misma región **reduce latencia**. Operaciones usadas: `macSign` y `macVerify` (105).

7.4.5. Comunicación de la API con MongoDB Atlas

- **Canal:**
  La API usa el **MongoDB Java Driver (Sync)** y se conecta a Atlas con URI `mongodb+srv://`. La conexión es **cifrada** de extremo a extremo (TLS); no hay tráfico en claro, y la conversación de base de datos viaja con el **MongoDB Wire Protocol** (`OP_MSG` con comandos BSON) (65; 107).

- **Auth:**
  Autenticación con **usuario y contraseña** de base de datos (SCRAM-SHA-256) dentro de la conexión cifrada (65).

- **Políticas:**
  Se aplican **tiempos de espera** y **reintentos** ante fallos de red. El cliente mantiene un **pool** de conexiones y **reconecta** automáticamente si cambia el nodo principal del clúster (65).

- **Seguridad:**
  Todo el tráfico va **cifrado en tránsito** (TLS) y se verifican certificados del servidor (107).

- **Observación:**
  El uso de `mongodb+srv://` permite **descubrir el clúster** vía DNS y evita depender de hosts/IPs concretos (65).

7.5. Configuración de componentes

Esta sección resume **cómo se han configurado los componentes** desde las **consolas de GCP, Firebase y MongoDB Atlas**: qué se activó, qué se parametrizó y qué se gestionó para poner en marcha cada pieza. El foco está en ajustes visibles de interfaz (proyectos y regiones, cuentas de servicio y permisos, dominios/orígenes, llaveros/clave, clústeres y usuarios).

7.5.1. Front end (Firebase Hosting)

Firebase Hosting es uno de los componentes que menos configuración ha requerido: tras cada despliegue, la nueva versión del *front end* se **propaga por la red CDN** de Firebase y queda disponible casi al instante. En el panel de la Figura 7.6 puede verse el estado del último despliegue publicado desde la consola del sitio.

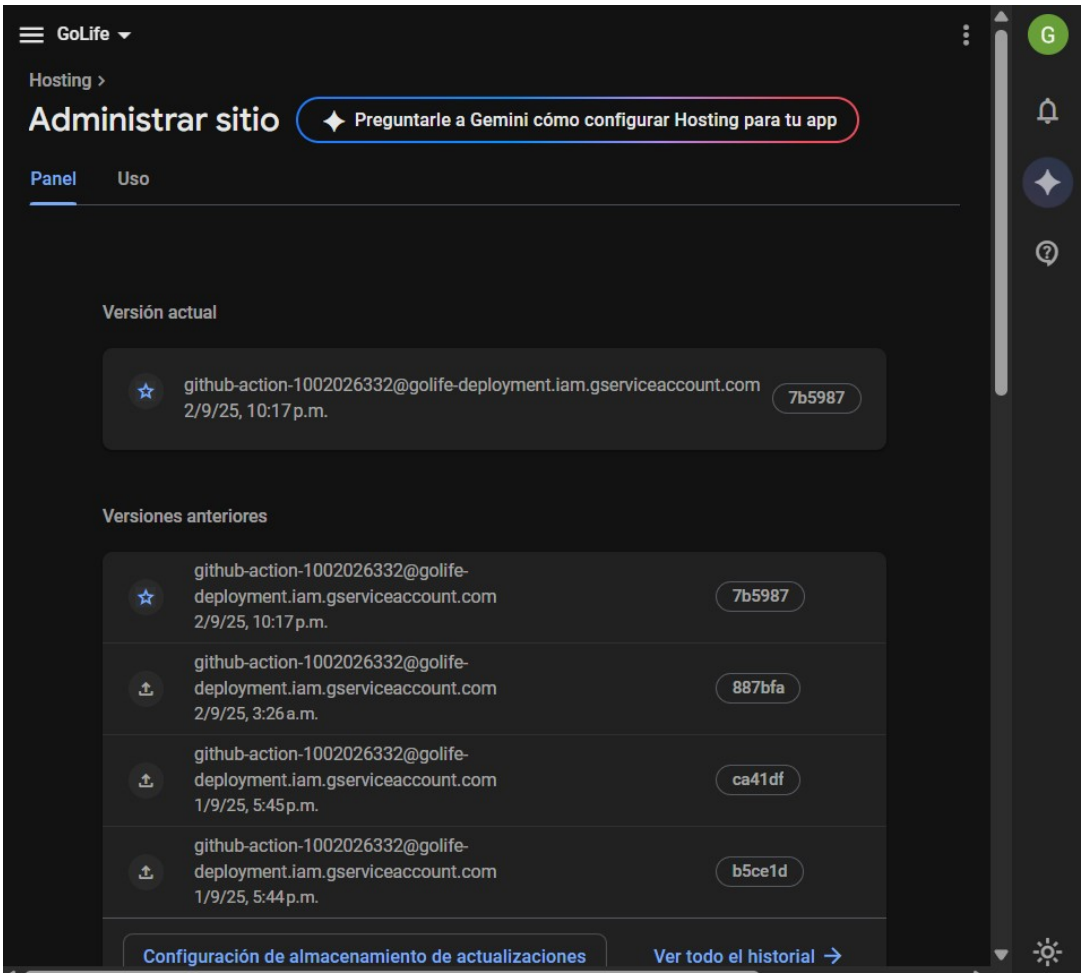


Figura 7.6: Panel de Firebase Hosting

Para facilitar la validación y mitigar riesgos, Hosting se ha configurado para **mantener "vivas" las dos versiones anteriores** a la última durante un tiempo tras cada despliegue. De este modo es posible revisar cambios mediante **enlaces temporales** y, si fuera necesario, **realizar *rollback***. La Figura 7.7 muestra el histórico de versiones disponibles.

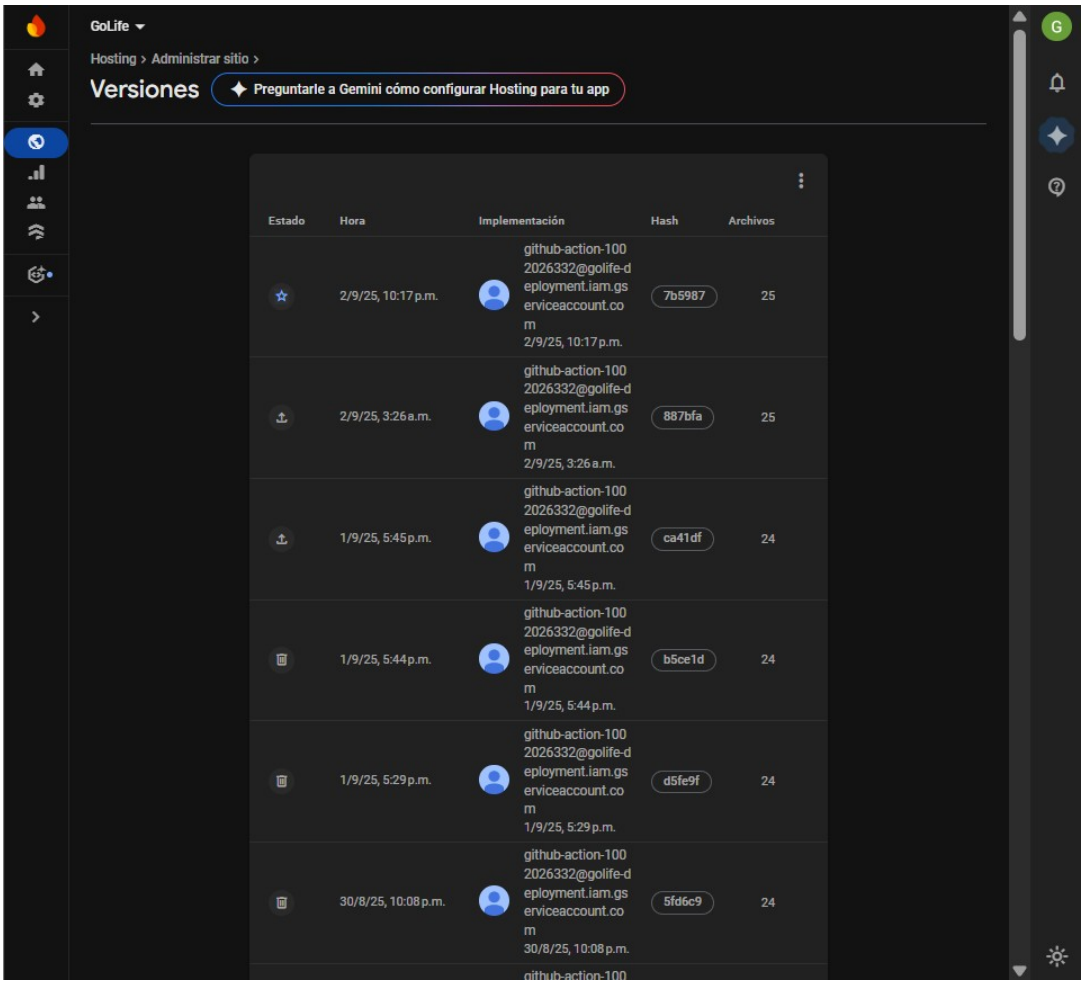


Figura 7.7: Histórico de versiones en Firebase Hosting

Adicionalmente, se han definido los **dominios de acceso** al *front end*. Cualquiera de las direcciones configuradas apunta a la **misma versión publicada** servida desde la CDN (véase Figura 7.8).

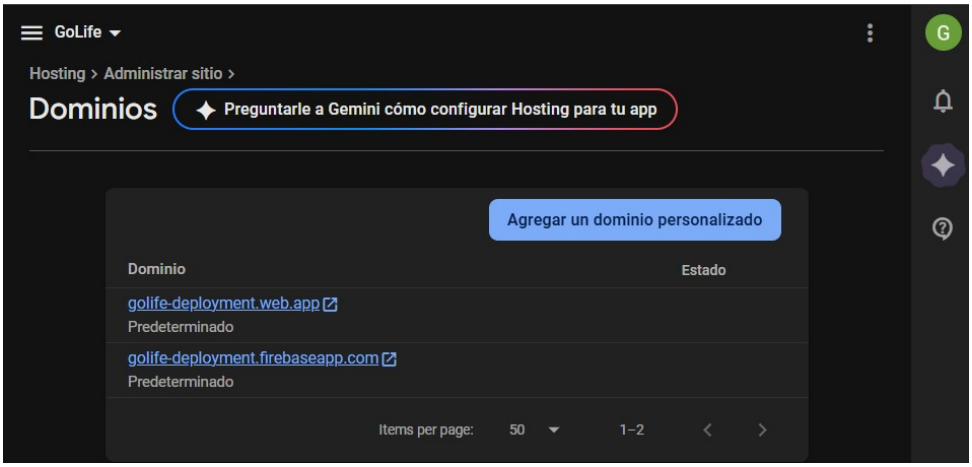


Figura 7.8: Dominios configurados en Firebase Hosting

7.5.2. Firebase Authentication

Firebase Authentication fue uno de los servicios más sencillos de configurar gracias a su consola y a que es un servicio totalmente gestionado: no requiere desplegar software propio y proporciona una *base de datos de usuarios* con altas, estado y proveedor de acceso. En la Figura 7.9 se muestra como es el listado de cuentas gestionadas.

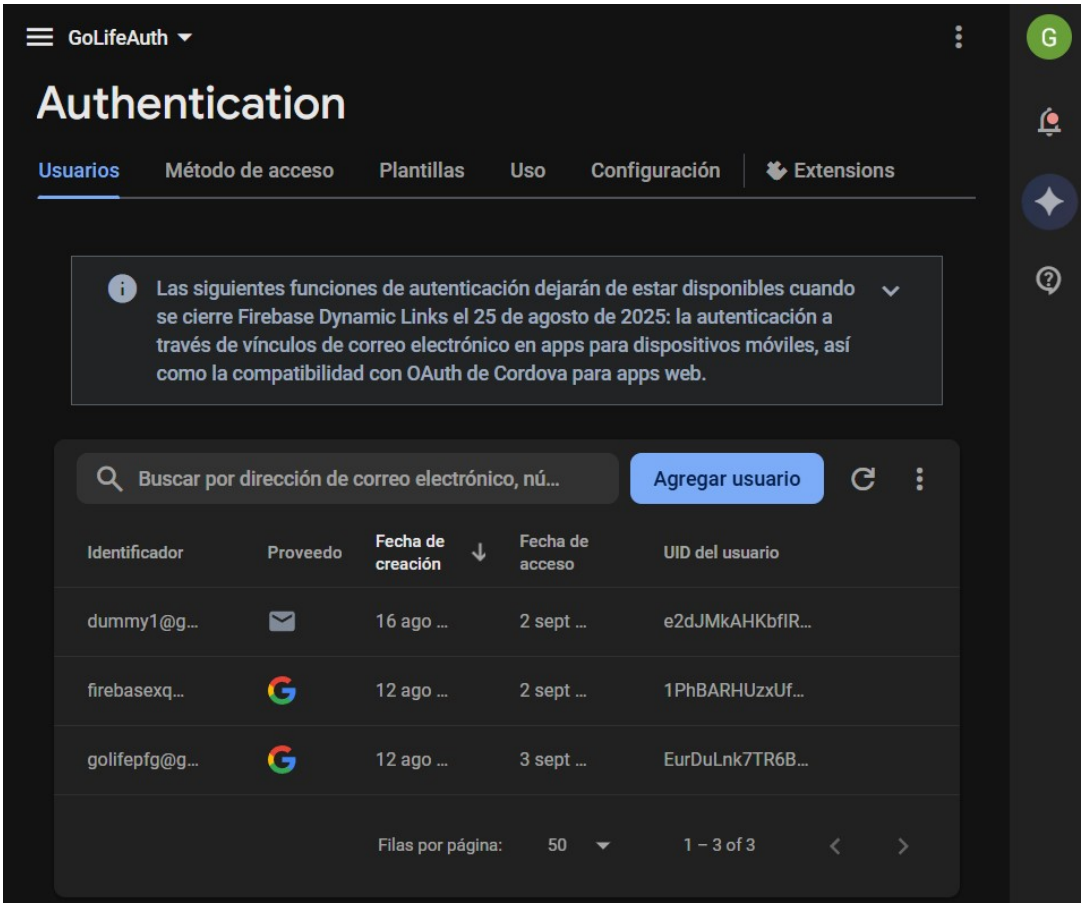


Figura 7.9: Usuarios gestionados por Firebase Authentication

Para el *inicio de sesión* y la *creación de cuenta* se habilitaron dos métodos: **Google** y **Email/Password**. Esto permite que el usuario use su cuenta de Google o, si lo prefiere, directamente correo y contraseña (Figura 7.10).

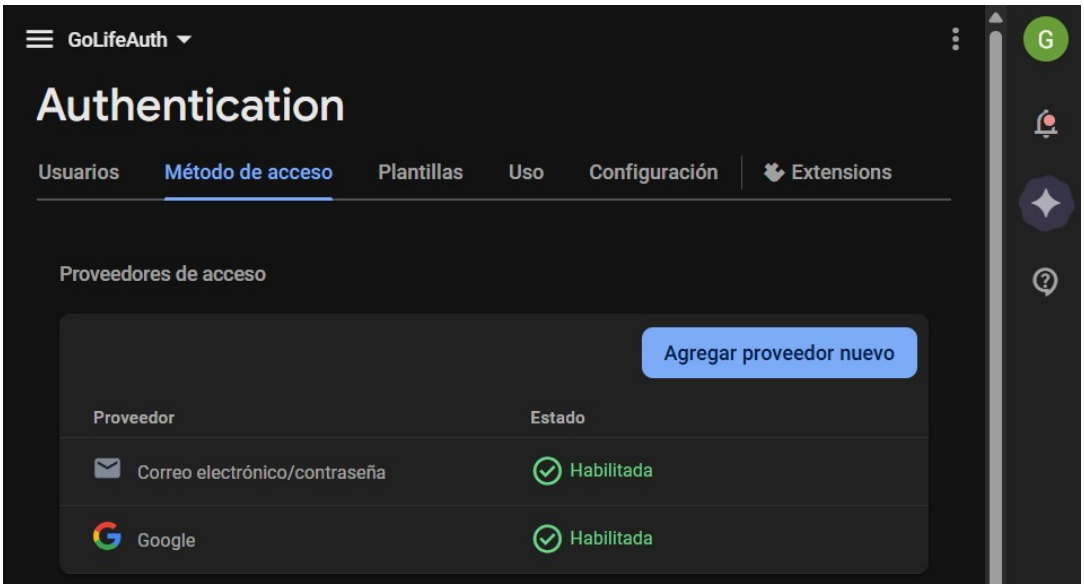


Figura 7.10: Proveedores/métodos de acceso habilitados

Asimismo, se limitaron las **acciones** disponibles a las necesarias para el proyecto: que el usuario pueda *crear* su cuenta y *eliminarla* cuando lo desee, como se aprecia en la Figura 7.11.

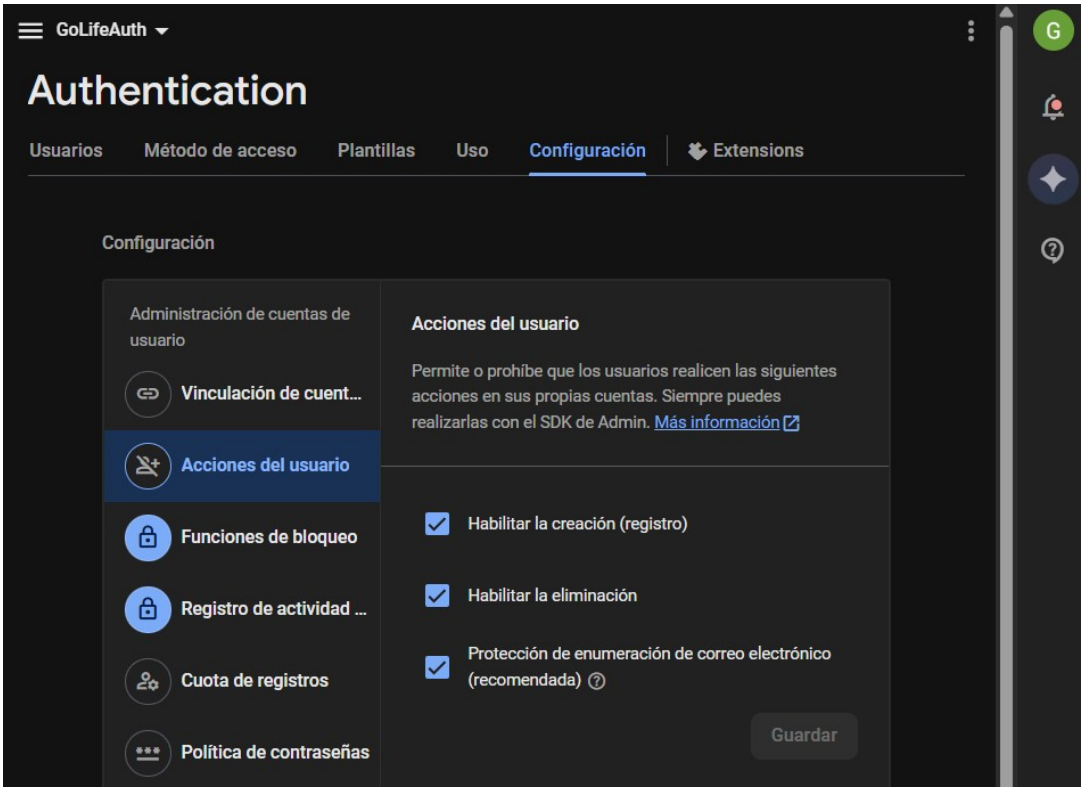


Figura 7.11: Configuración de acciones de cuenta

Para acotar la carga de usuarios y puesta a punto se configuró una **cuota diaria** de altas: **100 usuarios nuevos/día**. Esta medida evita crecimientos accidentales durante la validación de la aplicación (Figura 7.12).

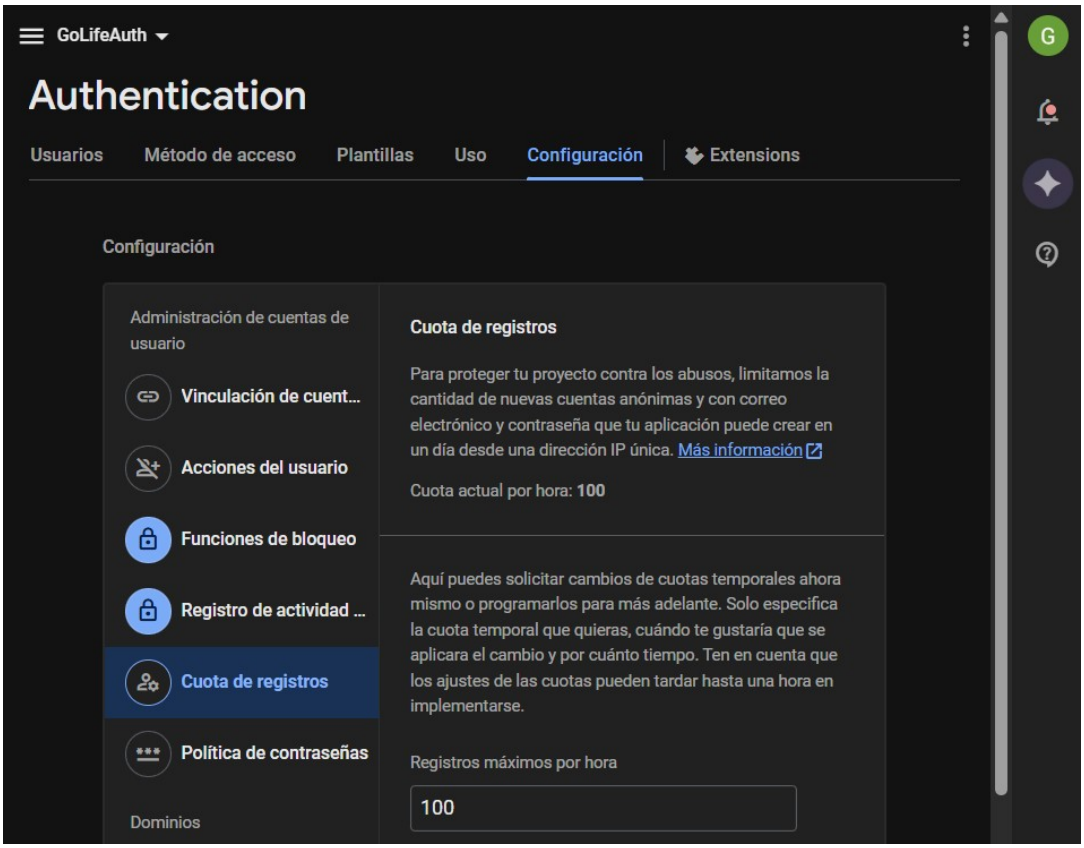


Figura 7.12: Límite de altas diarias configurado

En cuanto a **seguridad**, se definió una política de contraseñas siguiendo recomendaciones de buenas prácticas: mínimo **12 caracteres**, con **mayúsculas**, **minúsculas**, **números** y **caracteres especiales** (108). La Figura 7.13 muestra la configuración aplicada.

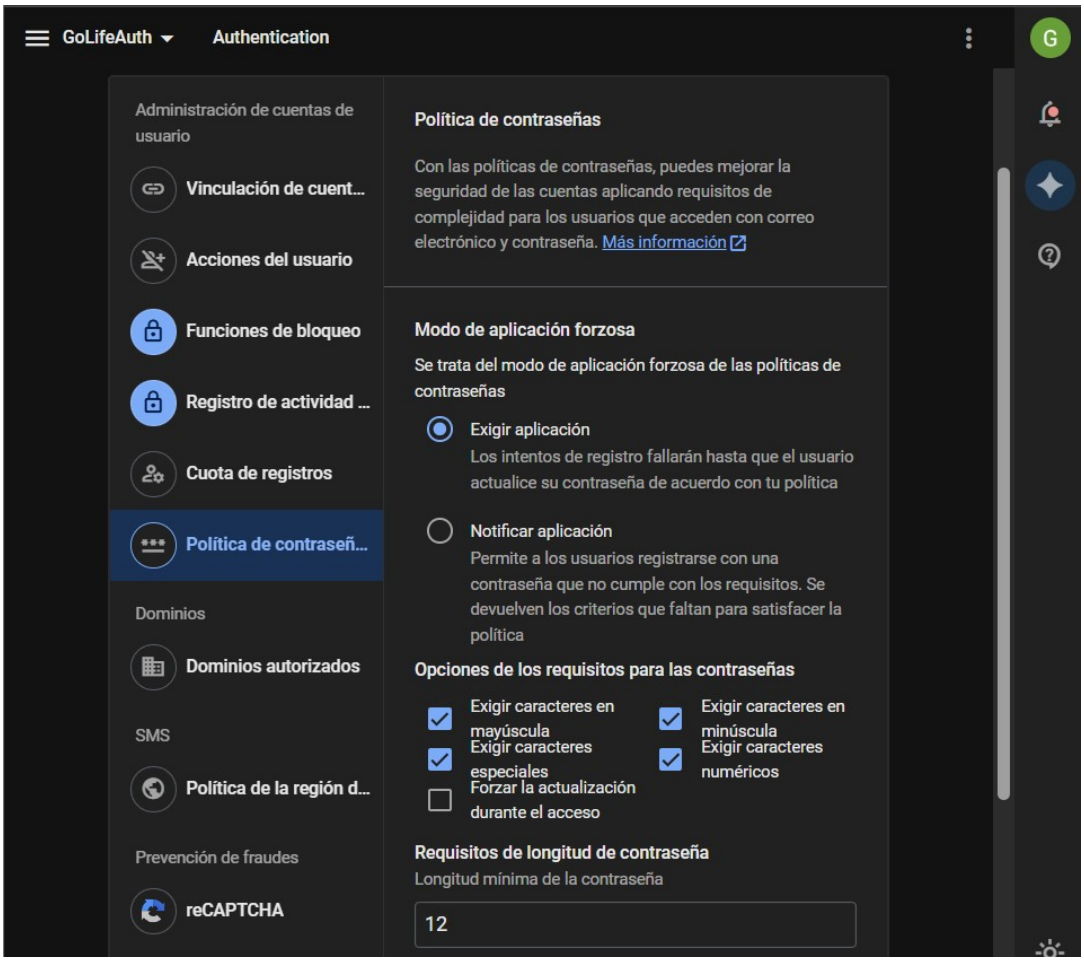


Figura 7.13: Política de contraseñas en Firebase Authentication

Finalmente, para la **recuperación de contraseña** se personalizó la **plantilla de correo** que reciben los usuarios con el enlace de restablecimiento, tal y como se observa en la Figura 7.14.

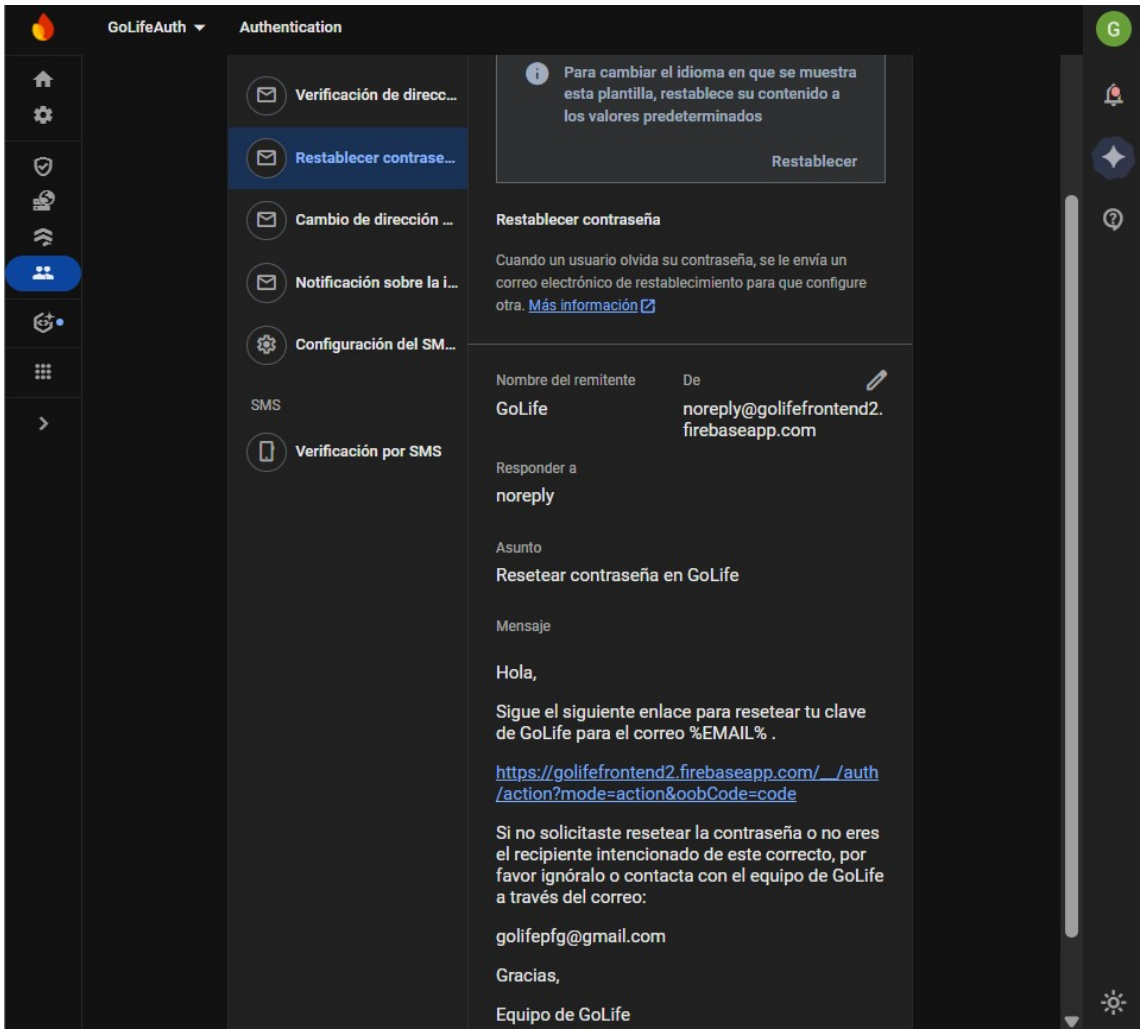


Figura 7.14: Plantilla de correo para restablecimiento de contraseña

7.5.3. API (App Engine)

La **API de GoLife** se despliega en **App Engine**, que arranca las máquinas virtuales necesarias según la configuración indicada en un fichero `app.yaml`.

Este fichero se toma durante el *pipeline* de CI/CD y define el **runtime**, la **clase de instancia**, el **entrypoint** (cómo iniciar la aplicación), las **políticas de escalado** (mínimo/máximo de instancias) y las **variables de entorno** que la API necesita para conectarse con los demás componentes. En nuestro caso usamos **Java 17** como runtime y la clase de instancia **F1** (la más sencilla y económica). Para mantener controladas las operaciones por segundo contra la base de datos (límite de 100/s) configuramos **máximo 1 instancia**; es suficiente para un proyecto acotado y de corta vida.

A continuación se muestra el `app.yaml` utilizado:

```
1 runtime: java17
2 instance_class: F1
3
4 entrypoint: java -jar target/GoLifeAPI-1.0.jar
```

```
5
6  automatic_scaling:
7    max_instances: 1
8
9  env_variables:
10   DB_CONNECTION_STRING: "DB_CONNECTION_STRING_PLACEHOLDER"
11   DB_NAME: "DB_NAME_PLACEHOLDER"
12   FIREBASE_PROJECT_ID: "FIREBASE_PROJECT_ID_PLACEHOLDER"
13   KMS_CRYPTO_KEY: "KMS_CRYPTO_KEY_PLACEHOLDER"
```

Listado 7.1: Fichero `app.yaml` de despliegue en App Engine

App Engine mantiene un historial de **versiones** de la aplicación, pero sólo la **versión activa** recibe tráfico. En la Figura 7.15 puede verse cómo el **100 % del tráfico** está asignado a la última versión publicada desde la consola de GCP.

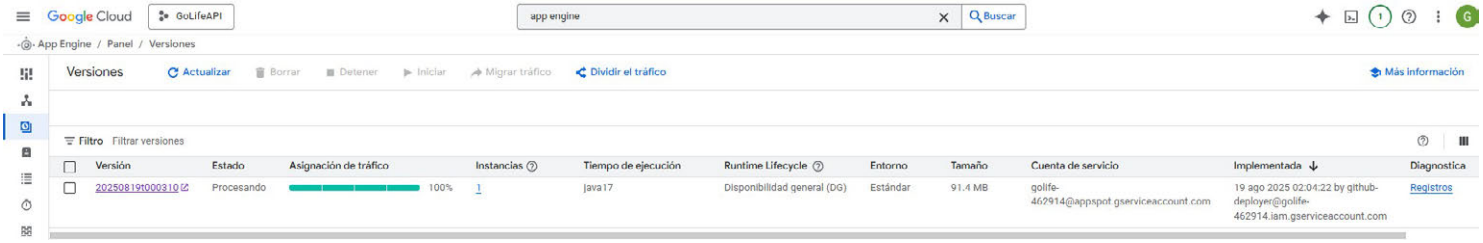


Figura 7.15: Versiones de la API en App Engine y asignación de tráfico

Respecto a **instancias**, en escenarios con varias instancias App Engine reparte el tráfico de forma equilibrada. En nuestro despliegue esto no aplica porque **limitamos a una sola instancia**. Además, la instancia es **dinámica**: si no hay tráfico se detiene y, cuando llegan peticiones, App Engine la vuelve a crear automáticamente. La Figura 7.16 muestra el estado de la instancia.

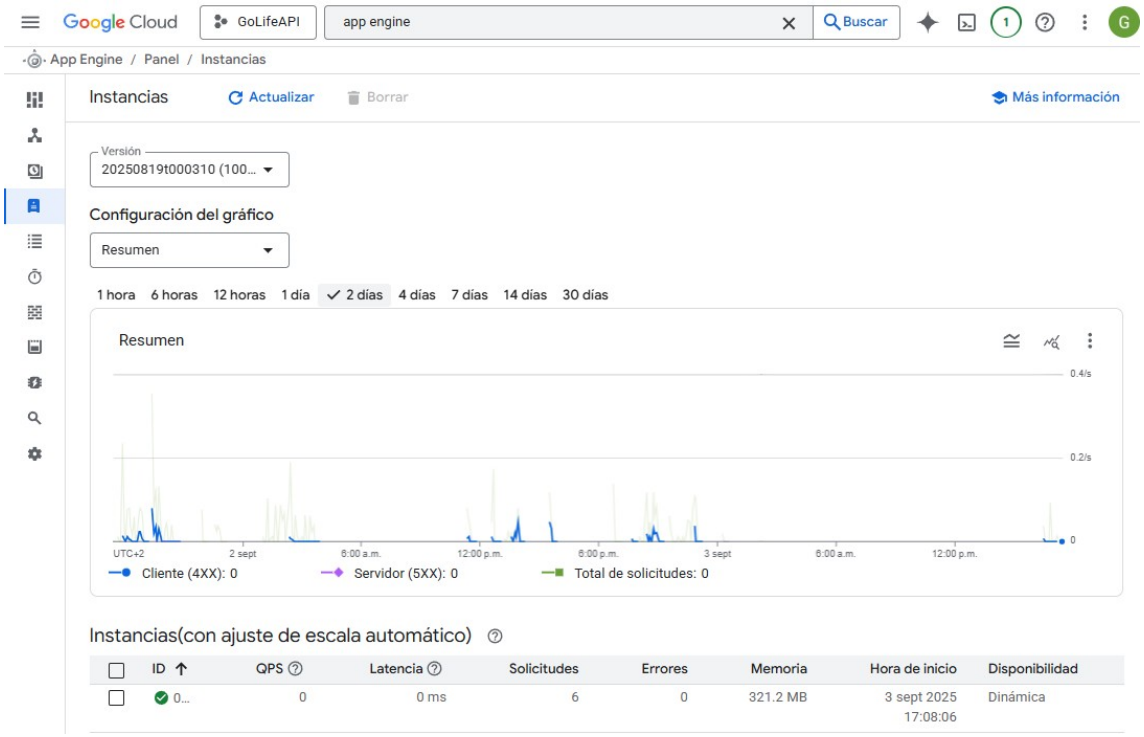


Figura 7.16: Instancia dinámica de la API en App Engine

Por último, en **firewall** conviene comentar una decisión práctica. Sería deseable filtrar por IP de origen y permitir únicamente la del front end, pero al servirse éste desde la **CDN de Firebase Hosting** su IP no es fija. Además, en App Engine no podemos expresar reglas de firewall por listas de IP dinámicas de la CDN. Por ello, se optó por **permitir el tráfico entrante** en las reglas de App Engine (véase Figura 7.17) y **restringir el origen efectivo** mediante la **política CORS** en la propia API (sólo acepta el dominio del front end).

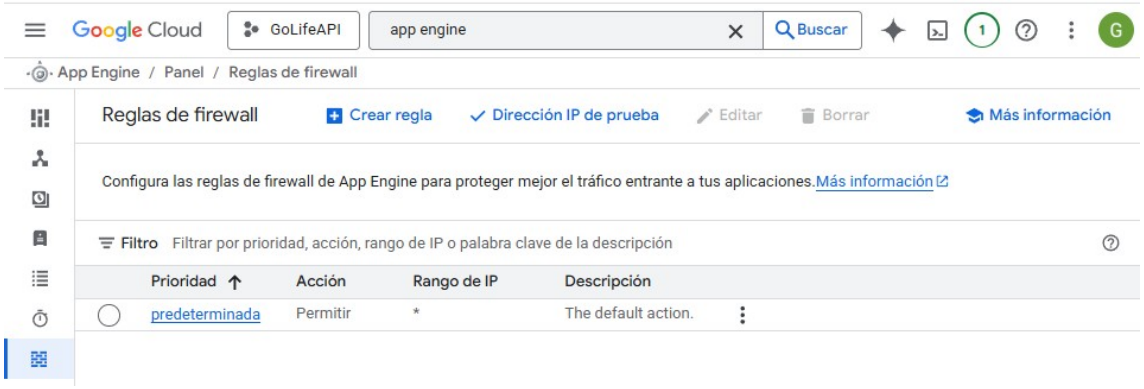


Figura 7.17: Reglas de firewall en App Engine

7.5.4. Cloud KMS

Cloud KMS es el servicio mediante el cual se **gestiona el llavero (keyring)** de la aplicación y se realizan operaciones criptográficas. Desde la API invocamos dos operaciones:

`macSign` (cálculo de un MAC) y `macVerify` (verificación del MAC) para seudonimizar y comprobar identificadores sin exponer el `uid`. En la Figura 7.18 se observa, desde la consola de GCP, un **histórico de llamadas** realizadas por App Engine a estas operaciones.

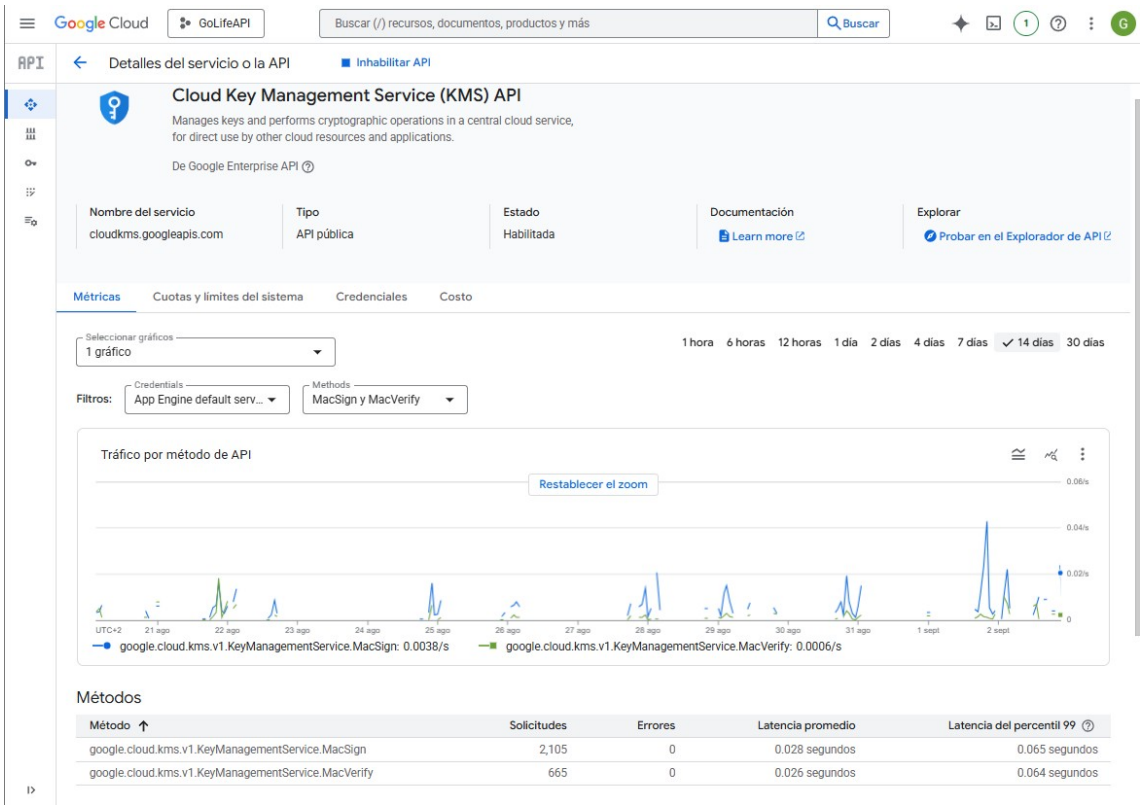


Figura 7.18: Histórico de llamadas a la API de Cloud KMS (macSign / macVerify)

El llavero reside en **Europa** y las claves están configuradas con **protección software**, suficiente para el alcance del proyecto. La Figura 7.19 muestra el **resumen de configuración** del servicio (región y nivel de protección).

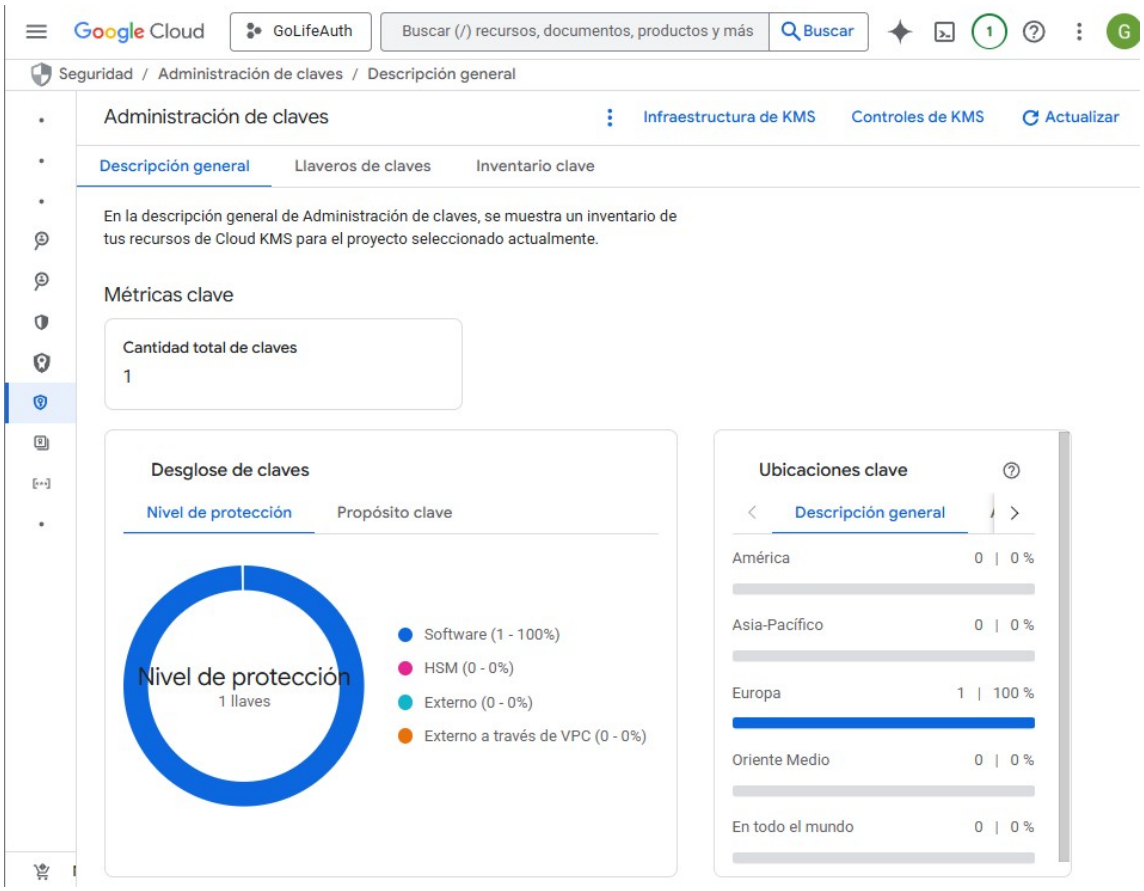


Figura 7.19: Descripción general de Cloud KMS (región y protección)

Nuestro llavero se denomina `GoLifeEncryption` y alberga la clave `uid-hmac-prod`. En la Figura 7.20 se aprecia el **keyring** y los **permisos IAM** otorgados a la cuenta de servicio de App Engine para poder usar la clave (firmar/verificar).

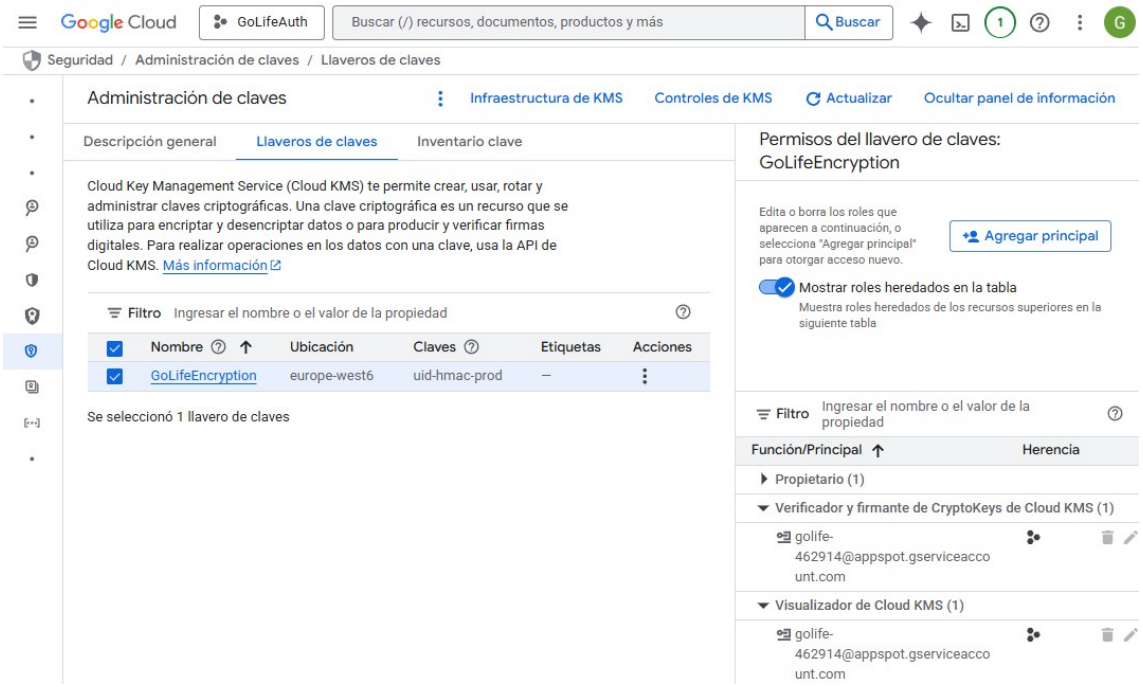


Figura 7.20: Llavero `GoLifeEncryption` y permisos de la cuenta de servicio

La clave `uid-hmac-prod` está creada con **uso restringido a MAC**, por lo que KMS únicamente permite las operaciones `macSign` y `macVerify`, tal como se muestra en la Figura 7.21. Esto reduce superficie y evita usos no previstos.

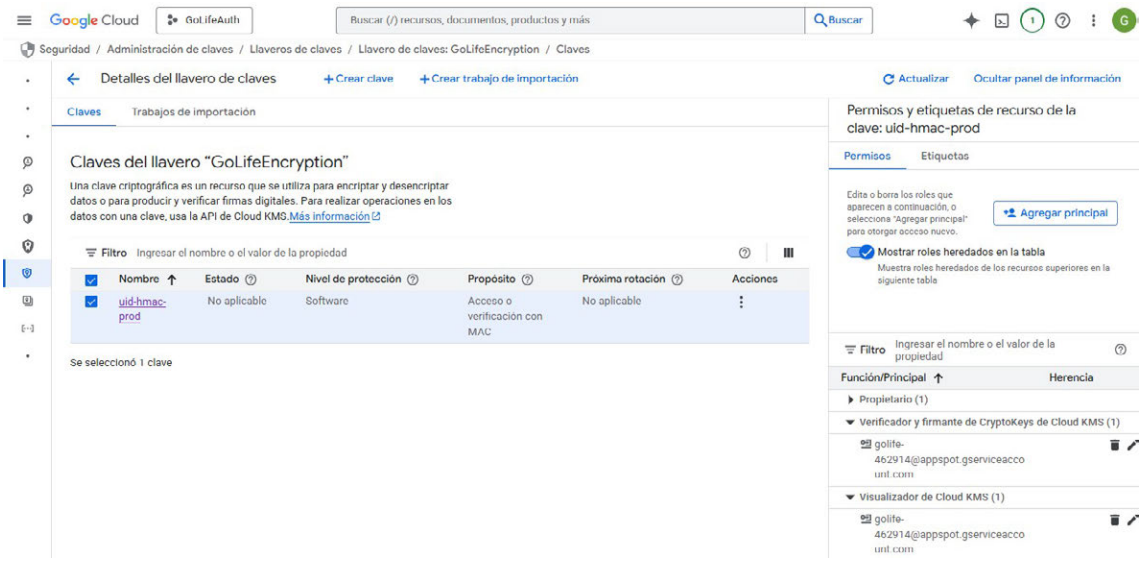


Figura 7.21: Clave `uid-hmac-prod` configurada para operaciones MAC

Durante el desarrollo y puesta en marcha se ha utilizado una **única versión** de clave, con algoritmo **HMAC-SHA256**. Este algoritmo es **no reversible** (genera seudónimos unidireccionales), **determinista** para el mismo `uid` (permite usar el valor MAC como clave externa consistente) y **alineado con RGPD** para fines de seudonimización; además, es el algoritmo *MAC* recomendado por Google Cloud KMS (109). La Figura 7.22 muestra el

historial de **versiones de la clave**; el material criptográfico es **generado por KMS** y no es accesible para los operadores (custodia gestionada).

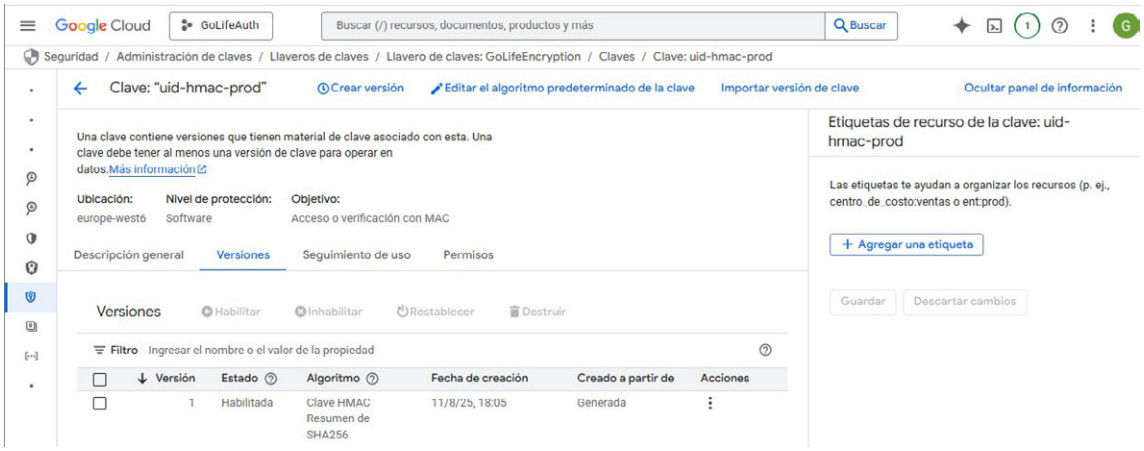


Figura 7.22: Versiones de la clave `uid-hmac-prod`

Finalmente, en la Figura 7.23 se resume la **configuración efectiva** de la clave: algoritmo HMAC-SHA256, ubicación europea, uso MAC (sign/verify) y nivel de protección software. Esta elección facilita la **seudonimización** de identificadores personales conforme a las *Guidelines on Pseudonymisation* del EDPB (110), manteniendo un equilibrio entre seguridad y simplicidad operativa.

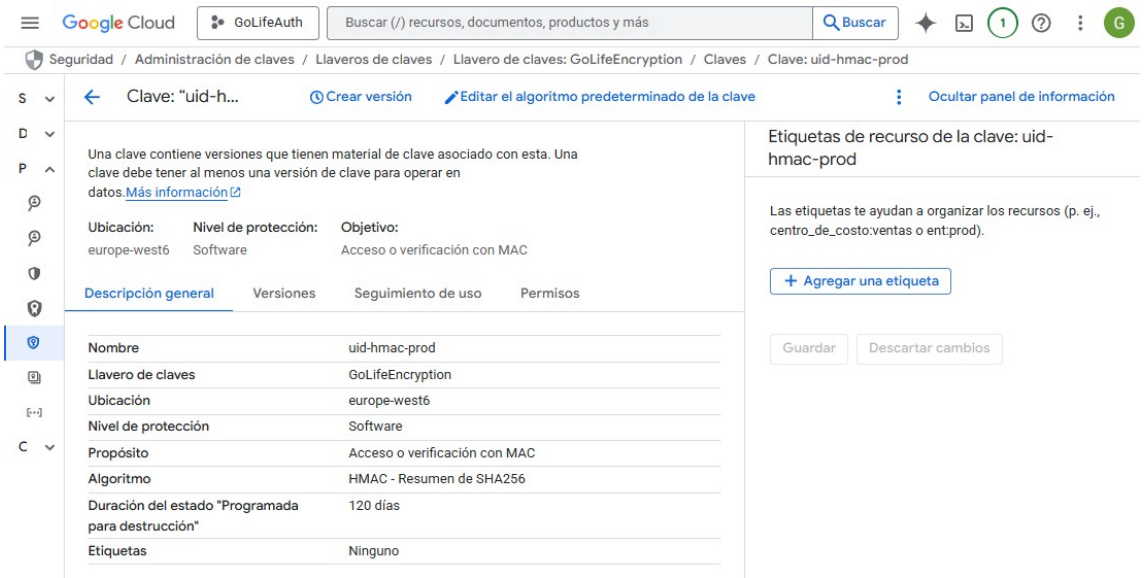


Figura 7.23: Resumen de propiedades de la clave `uid-hmac-prod`

7.5.5. MongoDB Atlas

MongoDB Atlas es el servicio gestionado que provee nuestra **base de datos**, desplegada en *clusters* administrados por MongoDB. En Atlas organizamos los recursos bajo una *organización* denominada `GoLife`; para este proyecto usamos el **cluster** `prod-golife-euwest-01` y la base de datos `GoLife`. La Figura 7.24 muestra el **resumen** en consola: nombre

del proyecto, **ubicación** y **plan** (`M0` - gratuito), junto con una vista de los **nodos** del despliegue.

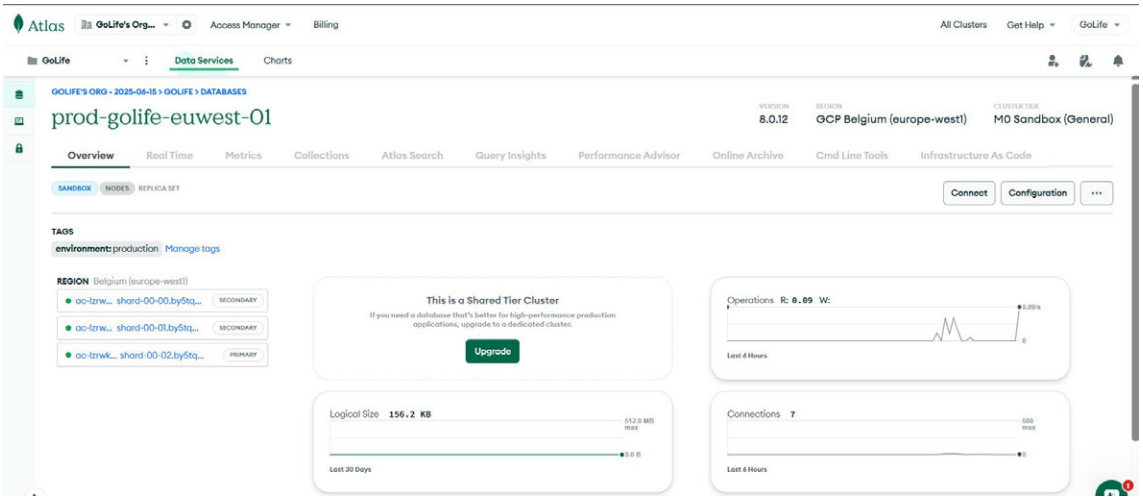


Figura 7.24: Resumen del proyecto y cluster en MongoDB Atlas

El **cluster** de Atlas está compuesto por **tres nodos**: un *primary* (único receptor de escrituras) y dos *secondaries* que **replican** el *oplog* (logs de inserción, actualización y eliminación) del primario y aseguran **alta disponibilidad** y durabilidad. La función de los secundarios es mantener un conjunto de datos consistente con el primario y participar en elecciones si fuera necesario. En la Figura 7.25 pueden verse **métricas por nodo** (transferencia de datos), que reflejan ese reparto de responsabilidades (111).

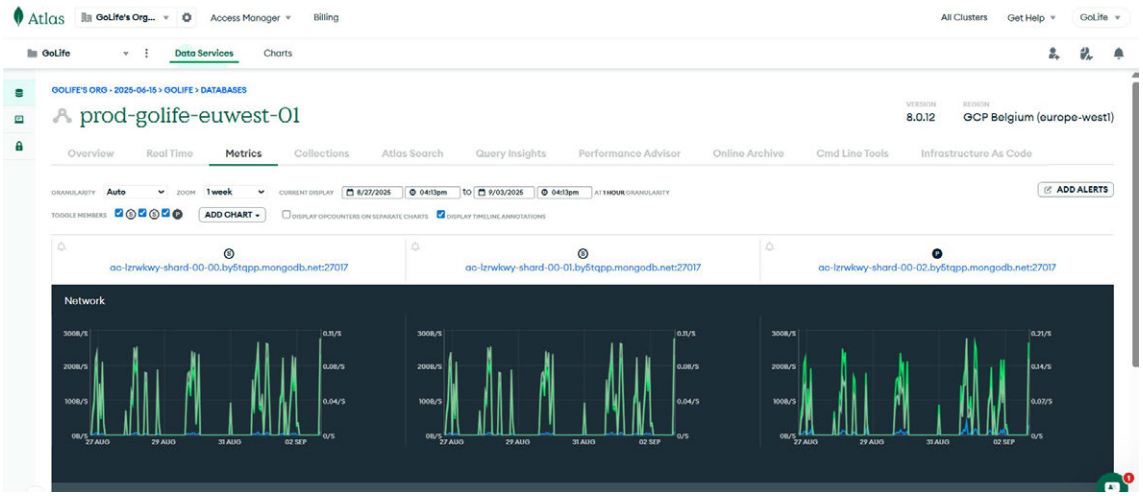


Figura 7.25: Métricas del cluster por nodo (primary/secondaries)

A nivel lógico, la base `GoLife` contiene dos **colecciones**: `Users` (datos base de usuario) y `Goals` (metas y sus registros). La Figura 7.26 muestra el listado de colecciones. Cada una dispone del **índice nativo** sobre `_id` y de un **índice adicional** para `uid` con fines de filtrado, tal y como se describió en la Subsubsección 6.3.4. En las Figura 7.27 y Figura 7.28 se ven los índices efectivos de `Users` y `Goals`, respectivamente.

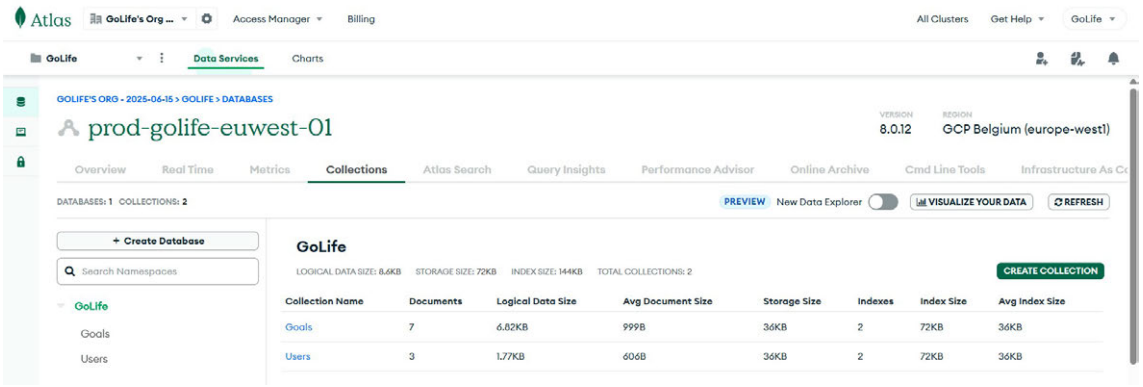


Figura 7.26: Colecciones `Users` y `Goals` en `GoLife`

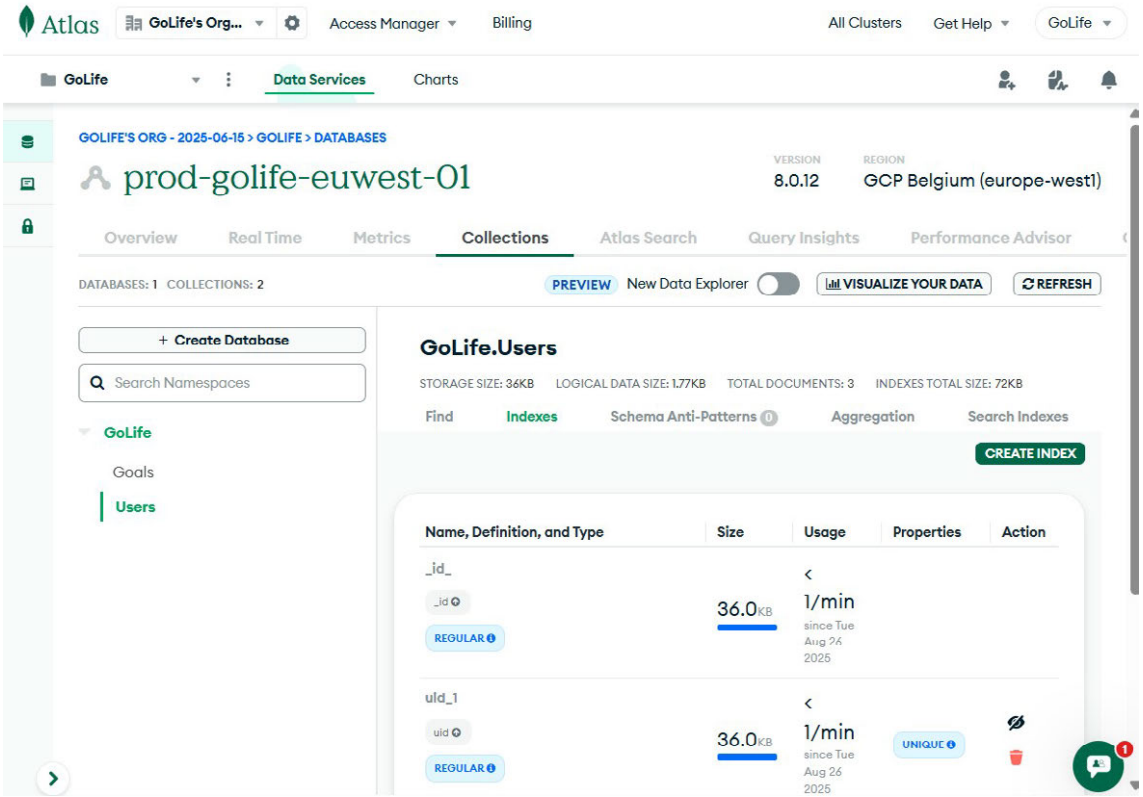


Figura 7.27: Índices de la colección `Users`

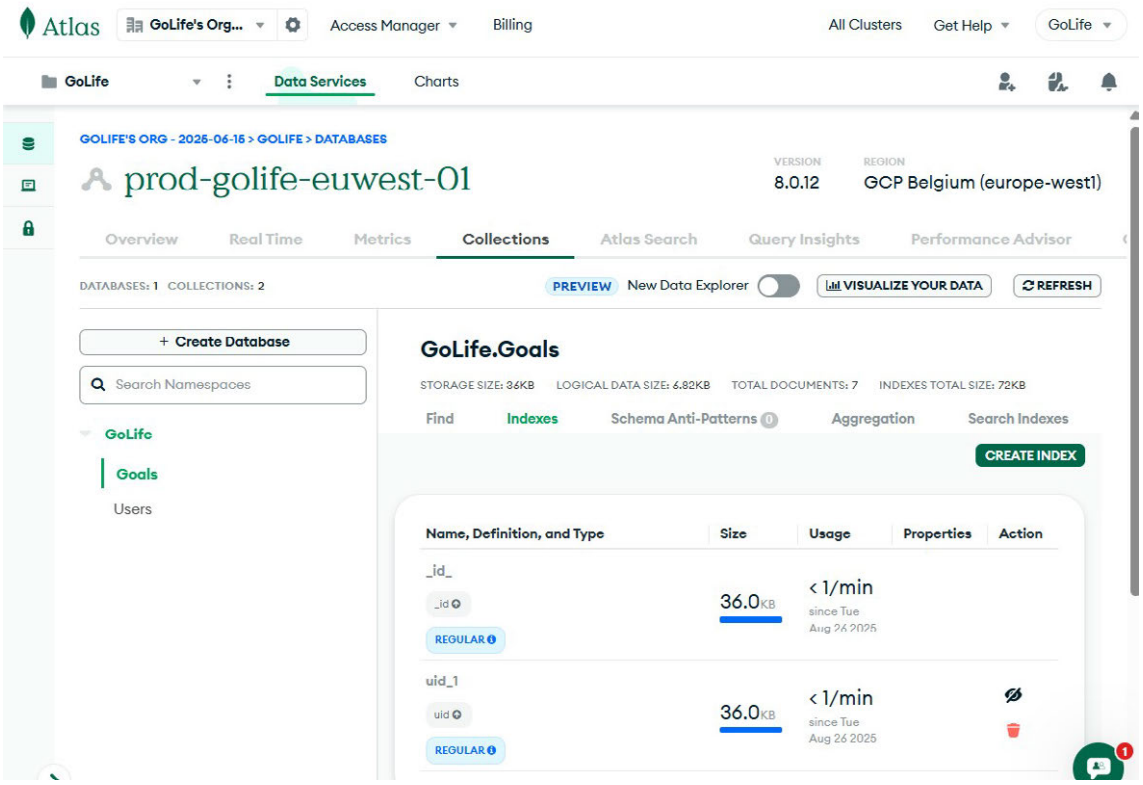


Figura 7.28: Índices de la colección `Goals`

Para el **acceso desde la API** se creó el usuario `golifeapi`, visible en la Figura 7.29. La autenticación usa **SCRAM** (basada en contraseña) dentro de un canal TLS. La cadena de conexión típica es:

```
mongodb+srv://<db_username>:<db_password>@prod-golife-euwest-01.by5tqpp
.mongodb.net/?retryWrites=true&w=majority&appName=prod-golife-euwest-01
```

El **rol** asociado a `golifeapi` sigue **mínimo privilegio**: permite `find`, `insert`, `remove` y `update` sobre `Users` y `Goals`, sin facultades para `drop` de colecciones ni tareas de administración de red o cluster. La Figura 7.30 recoge la definición de roles aplicada al usuario de aplicación.

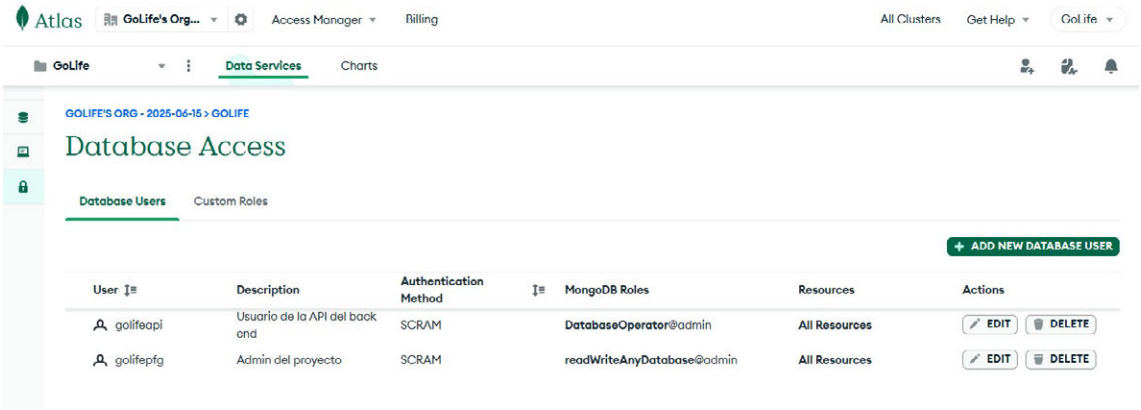


Figura 7.29: Usuarios de base de datos

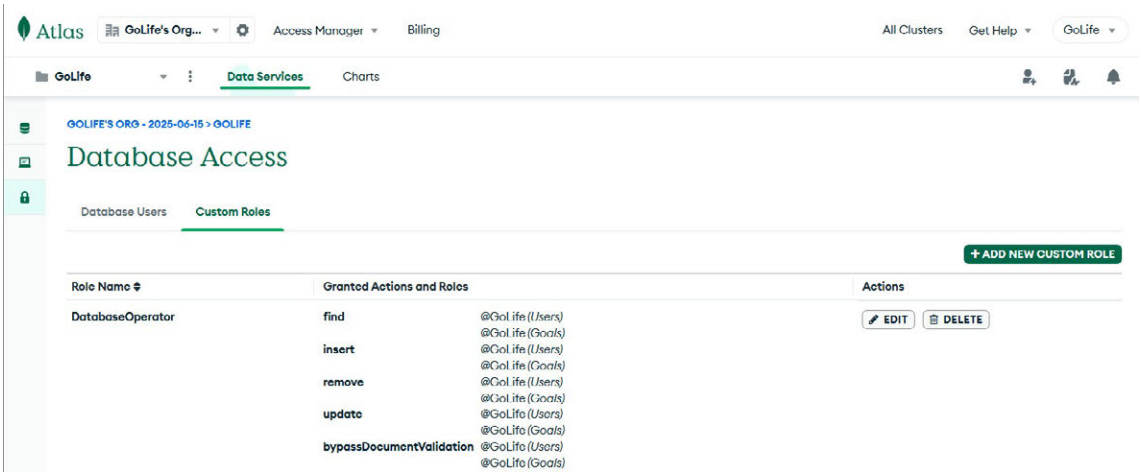


Figura 7.30: Rol personalizados de la base de datos

Para validar estructura y contenidos durante la puesta a punto, se conectó **MongoDB Compass**, la interfaz gráfica de usuario desarrollada por y para MongoDB, al clúster de Atlas con un usuario de administración (sólo para tareas de operación; la API usa un rol de mínimo privilegio). La Figura 7.31 muestra la base de datos `GoLife` con sus colecciones `Users` y `Goals` y documentos de **entorno de prueba**.

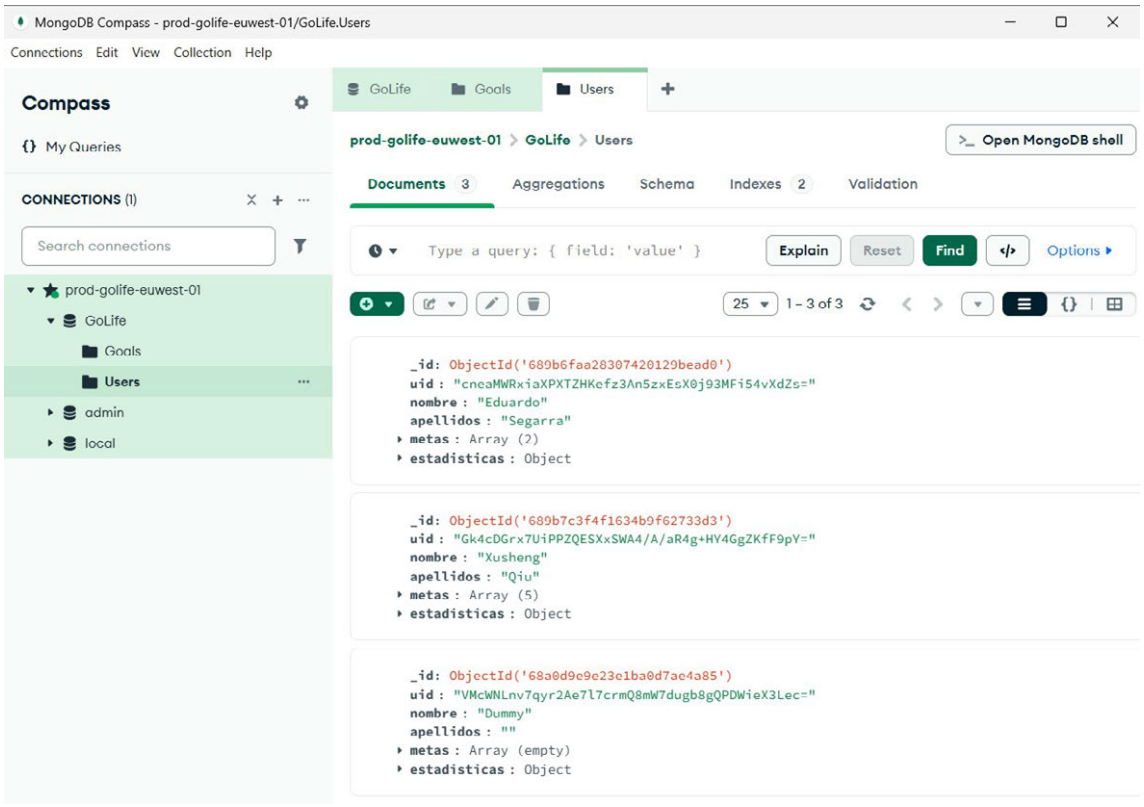


Figura 7.31: Vista de `GoLife` en MongoDB Compass

Por último, dado que el **plan M0** de Atlas **no ofrece copias de seguridad automáticas** en la nube, para *backups* y *restores* se recurre a las herramientas `mongodump` y `mongorestore`, según la guía oficial de Atlas (112). Esta estrategia es suficiente para el alcance del proyecto y su ventana de vida útil.

7.6. CI/CD y despliegues

En esta sección se explica **cómo el código pasa del repositorio a producción** mediante **GitHub Actions**: las **validaciones** previas (tests) y los **despliegues** automatizados de front end y API. Mostramos ejecuciones reales y los `.yml` que definen los pipelines para garantizar trazabilidad y reproducibilidad.

7.6.1. Pipeline a Firebase Hosting

El **front end** se despliega mediante un **pipeline de GitHub Actions** que se ejecuta al hacer *push* sobre la rama `main`. La Figura 7.32 muestra una ejecución completada, donde se aprecian las etapas encadenadas del flujo: primero la **batería de tests en CI** y, sólo si superan, la **construcción** del sitio y el **despliegue** a Firebase Hosting (canal `live`).

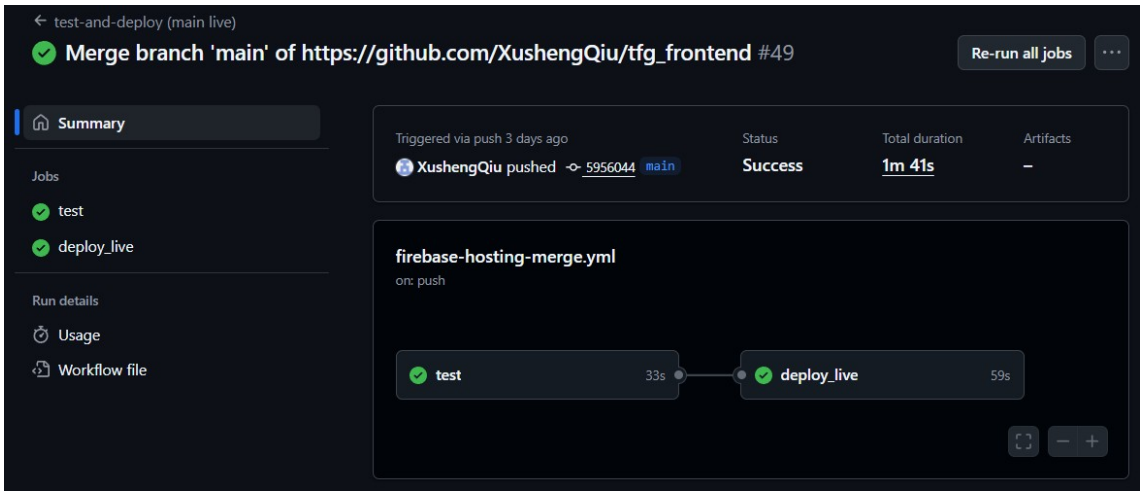


Figura 7.32: Ejecución del pipeline a Firebase Hosting en GitHub Actions

El workflow (Listado 7.2) define **dos jobs** en `ubuntu-latest`: `test` (tests unitarios) y `deploy_live` (construcción y despliegue a producción).

El job `deploy_live` declara `needs: test`, de modo que **sólo** publica si la fase previa ha terminado correctamente. En ambos jobs se realiza el *checkout* del repositorio y la configuración de **Node.js 20** con caché de `npm` (`actions/checkout@v4`, `actions/setup-node@v4`). El job `test` instala dependencias con `npm ci` y ejecuta `npm run test:ci` para validar la aplicación en CI. El job `deploy_live` genera un fichero `.env.production` a partir de `secrets` del repositorio (p. ej., `REACT_APP_FB_API_KEY`, `REACT_APP_FB_PROJECT_ID`, `REACT_APP_API_URL`, etc.) para **inyectarlos en tiempo de build** sin exponer credenciales; a continuación ejecuta `npm ci` y `npm run build` y, por último, publica con `FirebaseExtended / action-hosting-deploy@v0` indicando:

- `repoToken: ${ secrets.GITHUB_TOKEN }` (token del runner)
- `firebaseServiceAccount: ${ secrets.FIREBASE_SERVICE_ACCOUNT_GOLIFE_DEPLOYMENT }` (credencial JSON)
- `channelId: live` (despliegue directo a producción)
- `projectId: golife-deployment`.

```
1  # This file was auto-generated by the Firebase CLI
2  # https://github.com/firebase/firebase-tools
3
4  name: test-and-deploy (main live)
5
6  on:
7    push:
8      branches: [ main ]
9
10 jobs:
11   test:
12     runs-on: ubuntu-latest
```

```yaml
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'

      - name: Install
        run: npm ci

      - name: Run tests
        run: npm run test:ci

  deploy_live:
    needs: test
    if: success()
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Use Node.js 20
        uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'

      - name: Create .env.production from GitHub Secrets
        shell: bash
        run: |
          cat > .env.production <<'EOF'
          REACT_APP_FB_API_KEY=${{ secrets.REACT_APP_FB_API_KEY }}
          REACT_APP_FB_AUTH_DOMAIN=${{ secrets.REACT_APP_FB_AUTH_DOMAIN }}
          REACT_APP_FB_PROJECT_ID=${{ secrets.REACT_APP_FB_PROJECT_ID }}
          REACT_APP_FB_STORAGE_BUCKET=${{ secrets.REACT_APP_FB_STORAGE_BUCKET }}
          REACT_APP_FB_MESSAGING_SENDER_ID=${{ secrets.REACT_APP_FB_MESSAGING_SENDER_ID }}
          REACT_APP_FB_APP_ID=${{ secrets.REACT_APP_FB_APP_ID }}
          REACT_APP_FB_MEASUREMENT_ID=${{ secrets.REACT_APP_FB_MEASUREMENT_ID }}
          REACT_APP_API_URL=${{ secrets.REACT_APP_API_URL }}
          EOF

      - run: npm ci
      - run: npm run build

      - uses: FirebaseExtended/action-hosting-deploy@v0
        with:
          repoToken: ${{ secrets.GITHUB_TOKEN }}
          firebaseServiceAccount: ${{ secrets.FIREBASE_SERVICE_ACCOUNT_GOLIFE_DEPLOYMENT }}
          channelId: live
          projectId: golife-deployment
```

Listado 7.2: Workflow CI/CD: despliegue del front end a Firebase Hosting

En conjunto, este flujo asegura que:

1. Los **tests en CI bloquean** despliegues defectuosos.

2. La **configuración sensible** se gestiona como **secrets** e **inyecta** de forma controlada en el *build*.

3. El **despliegue** a Firebase Hosting (canal `live`) se realiza de forma **reproducible** y auditable.

7.6.2. Pipeline a App Engine

La API se despliega mediante un **pipeline de GitHub Actions** que se ejecuta al hacer *push* a la rama `master`. La Figura 7.33 muestra una ejecución completada, donde pueden verse las etapas encadenadas del flujo: primero las baterías de *tests*, y sólo si superan, la construcción y el despliegue en App Engine.

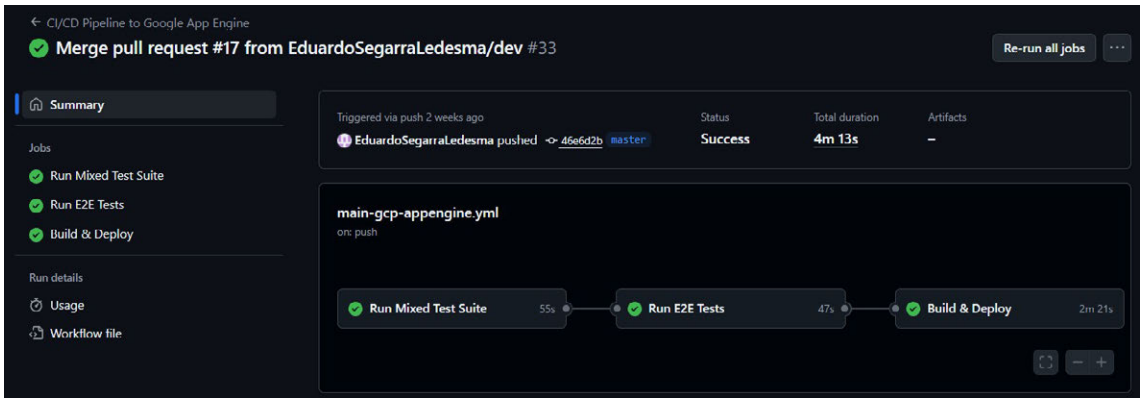


Figura 7.33: Ejecución del pipeline a App Engine en GitHub Actions

El workflow (Listado 7.3) define **tres jobs** en `ubuntu-latest`: `mixed-tests` (tests unitarios/mixtos), `e2e-tests` (pruebas de extremo a extremo) y `deploy` (construcción y despliegue).

El job `deploy` declara `needs` sobre los dos anteriores, de modo que **sólo** se publicará si las pruebas han pasado. En cada job se realiza el *checkout* del repositorio, se configura **Java 17** (`actions/setup-java@v4`, distribución *Temurin*) y se activa la **caché de Maven** para acelerar las ejecuciones. Los jobs de pruebas ejecutan `mvn test` (suite de tests mixtos) y `mvn verify` (suite de tests e2e o end-to-end); el job de despliegue construye el artefacto (`mvn package`, omitiendo tests porque ya se han ejecutado) y, antes de publicar, **sustituye variables** en `app.yaml` con valores provenientes de `secrets` del repositorio:

- `DB_CONNECTION_STRING`, `DB_NAME` (conexión a MongoDB Atlas y nombre de la base de datos)
- `FIREBASE_PROJECT_ID` (integración con Firebase Auth.)
- `KMS_CRYPTO_KEY` (recurso de Cloud KMS)

La autenticación contra GCP se realiza con `google-github-actions/auth@v2` usando la credencial JSON de una cuenta de servicio almacenada como `secrets.GCP_SA_KEY`.

Finalmente, la publicación se lleva a cabo con `google-github-actions/deploy-appengine@v2`, indicando `deliverables: app.yaml`. De este modo, el job `deploy` **promueve** a App Engine únicamente si las etapas anteriores han sido satisfactorias y con la configuración de entorno correcta inyectada en tiempo de pipeline.

```yaml
name: CI/CD Pipeline to Google App Engine

on:
  push:
    branches:
      - master

jobs:

  mixed-tests:
    name: Run Mixed Test Suite
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Java 17
        uses: actions/setup-java@v4
        with:
          distribution: 'temurin'
          java-version: '17'

      - name: Cache Maven local repository
        uses: actions/cache@v3
        with:
          path: ~/.m2/repository
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: |
            ${{ runner.os }}-m2-

      - name: Build & run tests
        run: mvn --batch-mode clean test

  e2e-tests:
    name: Run E2E Tests
    needs: mixed-tests
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Java 17
        uses: actions/setup-java@v4
        with:
          distribution: 'temurin'
          java-version: '17'

      - name: Cache Maven local repository
        uses: actions/cache@v3
        with:
          path: ~/.m2/repository
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: |
            ${{ runner.os }}-m2-
```

```
55
56      - name: Build & run E2E Tests
57        run: mvn --batch-mode clean verify -DskipUnitTests=true
58
59  deploy:
60    name: Build & Deploy
61    needs: e2e-tests
62    runs-on: ubuntu-latest
63    if: github.ref == 'refs/heads/master'
64    steps:
65      - name: Checkout code
66        uses: actions/checkout@v4
67
68      - name: Set up Java 17
69        uses: actions/setup-java@v4
70        with:
71          distribution: 'temurin'
72          java-version: '17'
73
74      - name: Cache Maven local repository
75        uses: actions/cache@v3
76        with:
77          path: ~/.m2/repository
78          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
79          restore-keys: |
80            ${{ runner.os }}-m2-
81
82      - name: Build package (skip tests)
83        run: mvn --batch-mode clean package -DskipTests
84
85      - name: Replace env vars in app.yaml
86        env:
87          DB_CONNECTION_STRING: ${{ secrets.DB_CONNECTION_STRING }}
88          DB_NAME:              ${{ secrets.DB_NAME }}
89          FIREBASE_PROJECT_ID:  ${{ secrets.FIREBASE_PROJECT_ID }}
90          KMS_CRYPTO_KEY:       ${{ secrets.KMS_CRYPTO_KEY }}
91        run: |
92          sed -i "s|DB_CONNECTION_STRING_PLACEHOLDER|$DB_CONNECTION_STRING|g" app.yaml
93          sed -i "s|DB_NAME_PLACEHOLDER|$DB_NAME|g" app.yaml
94          sed -i "s|FIREBASE_PROJECT_ID_PLACEHOLDER|$FIREBASE_PROJECT_ID|g" app.yaml
95          sed -i "s|KMS_CRYPTO_KEY_PLACEHOLDER|$KMS_CRYPTO_KEY|g" app.yaml
96
97      - name: Authenticate to Google Cloud
98        uses: google-github-actions/auth@v2
99        with:
100         credentials_json: '${{ secrets.GCP_SA_KEY }}'
101
102     - name: Deploy to App Engine
103       uses: google-github-actions/deploy-appengine@v2
104       with:
105         deliverables: app.yaml
```

Listado 7.3: Workflow CI/CD: despliegue de la API a App Engine

En conjunto, este flujo asegura que:

1. Las **pruebas** (unitarias y E2E) **bloquean** despliegues defectuosos.
2. La **configuración sensible** se gestiona como **secrets** y se inyecta de forma controlada.
3. El **despliegue** a App Engine se realiza de forma **reproducible** y auditable.

Capítulo 8

Implementación del front end

8.1. Visión

El *front end* de **GoLife** se ha construido como una *single–page application* (SPA) con React 18 y React Router, priorizando fluidez de navegación, claridad de interfaz y seguridad sin fricción. La autenticación se apoya en Firebase Auth y todas las llamadas al back end se canalizan por una capa de servicios Axios. La UI sigue un sistema ligero de *design tokens* en CSS (colores y espaciados coherentes) e incorpora tutorial contextual (Driver.js).

8.1.1. Intención

La *interfaz* de **GoLife** persigue una experiencia clara y de baja fricción para crear y seguir metas personales. Los objetivos de diseño son: (i) *rapidez de interacción* (render inmediato y pocas recargas), (ii) *transparencia* legal previa a la alta e inicio de sesión, (iii) *consistencia visual* con un sistema de diseño ligero basado en variables CSS, y (iv) *seguridad* sin complicar el flujo (ID token renovado automáticamente y rutas protegidas).

8.1.2. Arquitectura interna

Decisiones de diseño

- **SPA con React:** Se elige una *single–page application* para reducir latencia percibida, mantener estado de UI entre transiciones y poder instrumentar guías paso a paso (tutorial).
- **Arquitectura por páginas y servicios**. Páginas (`Login`, `Onboarding`, `Dashboard`, `Profile`) como *shells* de navegación; lógica de red en un servicio Axios con interceptor de `Authorization`.
- **Rutas privadas y guardas**. `onAuthStateChanged` como fuente de verdad para decidir acceso y redirigir a `/login` si la sesión expira o se revoca.
- **UI coherente con *design tokens***. Paleta y espaciados mediante variables CSS (`:root`), sin marcos utilitarios externos (se evita deliberadamente Tailwind).
- **Accesibilidad pragmática**. Diálogos modales con `role="dialog"`, foco gestionado y textos alternativos; tutorial no bloqueante.

Arquitectura del front end

- **Páginas**: `Login` (autenticación email/Google y recuperación), `Onboarding` (alta de perfil), `Dashboard` (lista de metas, registros, estadísticas y gráficas) y `Profile` (datos del usuario, logout y eliminación de cuenta).
- **Servicios**: capa HTTP con Axios; interceptor que obtiene `ID_TOKEN` con `user.getIdToken()` (renueva automáticamente si ha caducado); manejo conservador de `401` (reintento único + logout).

- **Estado de UI**: estado local por página; `AuthContext` para exponer usuario/profile y reacciones a cambios de sesión.
- **Despliegue**: CI/CD con GitHub Actions (previews en PR y `live` en `main`) hacia Firebase Hosting.

8.1.3. Diseño UX/UI

Principios

Se prioriza una experiencia *clara*, *predecible* y *recuperable*. Las decisiones siguen heurísticas de Nielsen (visibilidad del estado, control del usuario, prevención de errores) y buenas prácticas de formularios (etiquetas visibles, ayudas contextuales, validación inmediata y CTAs deshabilitados hasta cumplir requisitos).

Sistema visual (tokens) La coherencia se garantiza con variables CSS globales:

Token	Uso de referencia
`-brand` / `-brand-600`	Color primario y *hover* en CTAs y focos activos.
`-accent`	Acciones destacadas (p. ej., llamada a crear primera meta).
`-bg-soft`	Fondos suaves en paneles amplios (área de gráficas/estadísticas).
`-title-w`	Ancho de títulos/sidebars para mantener ritmo visual.

Se evita *framework* utilitario externo; el layout se compone con CSS propio para reducir dependencias y ajustar finamente el diseño.

Patrones por pantalla Login. Formulario con ayuda de contraseña (mín. 12, mezcla de tipos); enlaces a legales en el pie; botón de Google; *estado de carga* que evita dobles envíos.

Onboarding. Campos mínimos (nombre, apellidos opcional); modales de Términos/Tratamiento con *scroll obligado* para consentimiento informado.

Dashboard. Lista de metas a la izquierda (acciones: *Entrada*, *Editar*, *Finalizar*, *Eliminar*); metas finalizadas con *fila gris* para diferenciación; al seleccionar una meta, se despliega *registros* y *descripción*. En la derecha, *estadísticas* arriba y área de *gráficas* principal. Botón *Crear meta* junto al de perfil.

Profile. Tarjeta con datos; acciones: *Volver*, *Cerrar sesión*, *Editar*, *Eliminar cuenta*. La eliminación exige doble éxito (back end + Firebase); si uno falla, no se cambia el estado y se informa en la misma vista.

Formularios y validación Etiquetas visibles, *helper text* breve, mensajes *inline* de error y CTA deshabilitado hasta que el formulario sea válido. **Crear meta**: periodo (*indefinido* o cantidad + unidad {días, semanas, meses, años}), *objetivo* (numérico+unidad o *check*), *fecha* con icono de calendario que abre selector, y descripción. Validaciones inmediatas reducen correcciones posteriores.

Tutorial contextual El recorrido con Driver.js guía tareas clave sin bloquear; destaca elementos, usa pasos cortos y lenguaje directo. Ajustes post-testing: mini-tutorial al cerrar

la bienvenida (señala el botón de tutorial), aclaración extra en "Crear meta" (relación valor+unidad vs. *check*) y limpieza del buscador al terminar el tour para evitar listas vacías.

Vacíos, feedback y microcopy Estados vacíos "amables" (p. ej., nuevo usuario → "Crea tu primera meta"), mensajes breves y específicos en confirmaciones (*"Finalizar meta: acción irreversible"*, *"Nuevo registro de: <Meta>"*). Se evita jerga y se prioriza el verbo de acción.

Accesibilidad (WCAG 2.2 AA) Orden de tabulación coherente, foco visible, `role="dialog"` en modales con `aria-modal="true"`, retorno del foco al disparador al cerrar, `aria-live="polite"` en banners de error, y contraste suficiente en estados (p. ej., fila gris de finalizadas con texto legible). La interacción es totalmente por teclado.

Responsive y adaptabilidad Diseño de dos columnas en escritorio (lista/acciones + panel derecho) y de una columna en móvil (prioriza tabla y acciones contextuales). La rejilla y los espaciados mantienen ritmo vertical constante; los gráficos escalan al ancho disponible sin perder legibilidad de etiquetas.

Rendimiento percibido Bloqueo visual mínimo en llamadas; botones muestran *loading* y los formularios mantienen datos ante error para reintento. Uso moderado de dependencias y Firebase Hosting con caché de estáticos *immutable*.

Racional de cambios tras tests con usuarios *Título en alta de registro*, *mini-tutorial*, *limpieza de buscador tras tour*, *mejor explicación de objetivo/unidad* y *modal informativo en contraseña inválida* corrigen ambigüedades observadas y reducen errores de interacción.

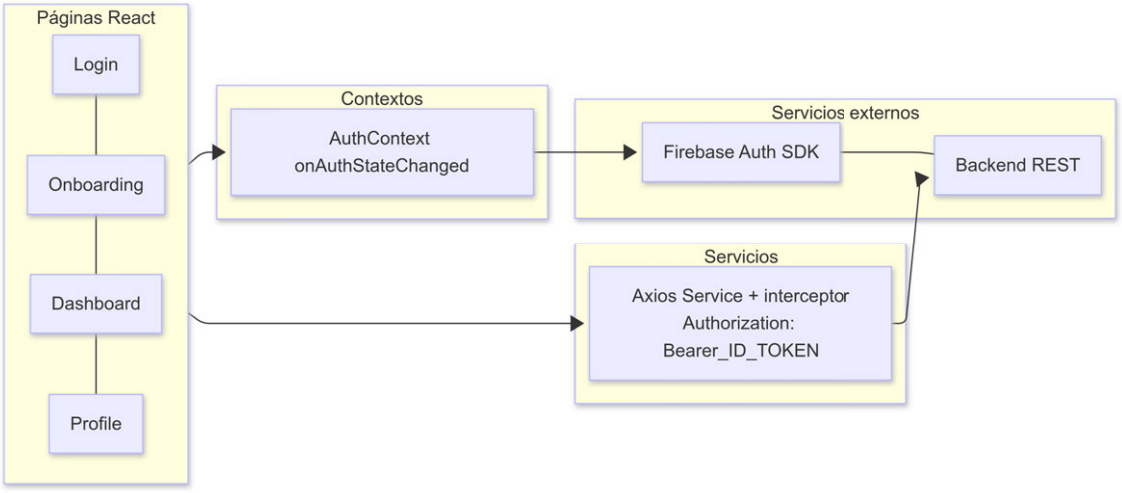


Figura 8.1: Arquitectura del *front end*

8.2. Acceso a API

El *front end* consume una API REST propia y autenticada. Toda la comunicación se centraliza en una capa de **servicios** sobre un cliente HTTP (`Axios`) configurado con: (i)

`baseURL` desde `REACT_APP_API_BASE_URL` (secreto de CI), (ii) cabecera `Authorization: Bearer <ID_TOKEN>` obtenida desde Firebase Auth y (iii) tratamiento homogéneo de errores (401 con refresco y reintento único; resto, delegados a UI).

8.2.1. Cliente HTTP y configuración

- **Base** `Axios` con `baseURL` desde variables de entorno; `timeout` razonable (p. ej., 10 s).
- **Autenticación** por `Bearer` (ID token de ∼1 h; renovación automática al solicitarlo).
- **Interceptors**: de *request* (inyecta token) y de *response* (401 → `getIdToken(true)` + *un* reintento; si persiste, `logout`).
- **Formato**: JSON; fechas en ISO-8601 (`YYYY-MM-DD`); unidades explícitas en metas numéricas.

```
1  import axios from 'axios';
2  import { getAuth } from 'firebase/auth';
3
4  export const api = axios.create({
5    baseURL: import.meta.env.VITE_API_BASE_URL ||
6    process.env.REACT_APP_API_BASE_URL,
7    timeout: 10000,
8  });
9
10 api.interceptors.request.use(async (config) => {
11   const user = getAuth().currentUser;
12   if (user) {
13     const idToken = await user.getIdToken(); // refresca si caduc
14     config.headers.Authorization = `Bearer ${idToken}`;
15   }
16   return config;
17 });
18
19 // 401 => refresco forzado y un reintento como mximo
20 let retried = new WeakSet();
21 api.interceptors.response.use(
22   (r) => r,
23   async (err) => {
24     const { response, config } = err;
25     if (response?.status === 401 && config && !retried.has(config)) {
26       retried.add(config);
27       const user = getAuth().currentUser;
28       if (user) {
29         const fresh = await user.getIdToken(true);
30         config.headers.Authorization = `Bearer ${fresh}`;
31         return api.request(config);
32       }
33     }
34     return Promise.reject(err);
35   }
36 );
```

Listado 8.1: Cliente Axios del front end (esquema simplificado)

8.2.2. Convenciones y políticas

- **Filtros de serie**: se soporta `from=...&to=...` (rango manual), `period=MONTH|WEEK|DAY` o `count=`N (últimos N registros).
- **Idempotencia**: acciones como `finalize` toleran reenvío; `DELETE` debe ser *safe to repeat*.
- **Ordenación/búsqueda**: el buscador por nombre se realiza en cliente; la lista por defecto llega ordenada por fecha.
- **Errores**: respuesta JSON con `status` HTTP y cuerpo con `code/message`; el mapeo UI se detalla en la sección de Gestión de errores.

8.2.3. Servicios de dominio (capa *services*)

```
export async function listGoals() {
  const { data } = await api.get('/goals');
  return data;
}

export async function createGoal(payload) {
  // payload: { name, period, objective, startDate, description }
  const { data } = await api.post('/goals', payload);
  return data;
}

export async function addEntry(goalId, entry) {
  // entry: { date, value }
  const { data } = await api.post(`/goals/${goalId}/entries`, entry);
  return data;
}

export async function finalizeGoal(goalId) {
  await api.post(`/goals/${goalId}/finalize`);
}

export async function deleteGoal(goalId) {
  await api.delete(`/goals/${goalId}`);
}
```

Listado 8.2: Servicios representativos

Alta de meta → registro → refresco de serie:

```
const g = await createGoal({
  name: 'Alcanzar peso sano',
  period: { amount: 6, unit: 'MONTHS', indefinite: false },
  objective: { type: 'NUMBER', target: 72.5, unit: 'Kg' },
  startDate: '2025-06-01',
  description: '...'
});
await addEntry(g.id, { date: '2025-09-16', value: 75.1 });
const series = await api.get(`/goals/${g.id}/series`,
{ params: { count: 28 }});
```

```
11 renderChart(series.data);
```

Listado 8.3: Uso de servicios en un flujo de UI

8.3. Autenticación y ciclo de tokens

La aplicación web utiliza **Firebase Authentication** como capa de identidad y sesión. Se ofrecen dos vías de acceso: correo/contraseña y Google (OAuth 2.0). Tras autenticarse, el SDK mantiene una sesión persistente en el navegador y expone un *ID token* (JWT) con una caducidad aproximada de ∼1 hora, además de un *refresh token* persistente que permite renovar el primero de forma transparente. En el *front end* nunca se guardan manualmente tokens: el ciclo de vida queda delegado al SDK.

8.3.1. Obtención/renovación del ID token y tratamiento de 401

Antes de cada petición al back end se adjunta `Authorization: Bearer` *`ID_TOKEN`*. La llamada `user.getIdToken()` *ya* comprueba la caducidad y, si procede, *refresca* el token con el *refresh token* persistente, devolviendo siempre un ID token válido. Por eso no es necesario implementar lógica de reloj en cliente.

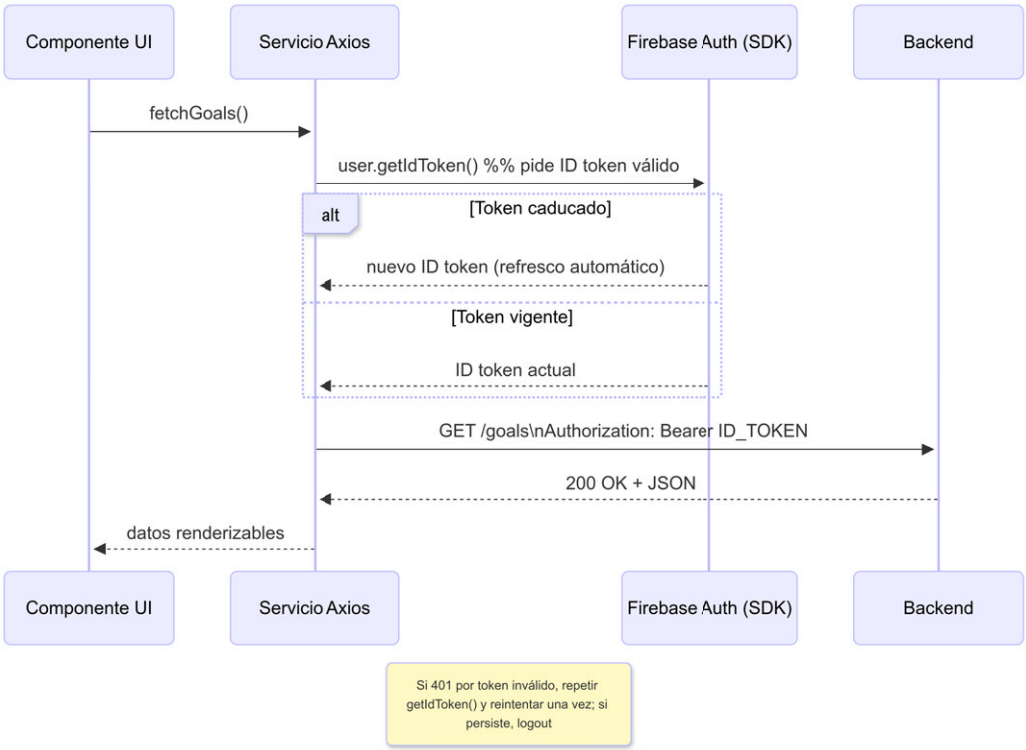


Figura 8.2: Secuencia de petición segura

Si el back end devuelve `401 Unauthorized` (p. ej., token revocado o reloj desfasado), se realiza un único reintento con refresco forzado; si vuelve a fallar, se invalida la sesión y se redirige a `/login`.

```
import { getAuth, signOut } from "firebase/auth";
api.interceptors.response.use(
  (r) => r,
  async (error) => {
    const { response, config } = error || {};
    const auth = getAuth();
    if (response?.status === 401 && !config._retry) {
      config._retry = true;
      const idToken = await auth.currentUser?.getIdToken(true);
      if (idToken) {
        config.headers.Authorization = `Bearer ${idToken}`;
        return api(config); // reintento unico
      }
    }
    if (response?.status === 401) {
      await signOut(auth); // sesin invlida o revocada
      // aqu: navegacin a /login (router)
    }
    return Promise.reject(error);
  }
);
```

8.3.2. Interceptor de solicitudes (Axios)

Responsabilidades cruzadas del interceptor

- **Autenticación**: obtener un *ID token* válido (`getIdToken()`) y fijar `Authorization: Bearer <ID_TOKEN>`.
- **Normalización de formato**: asegurar `Accept: application/json` y `Content-Type: application/json` en `POST/PUT/PATCH`.
- **Política de errores**: 401 → refresco forzado + **un** reintento; resto se propagan a la UI.
- **Neutralidad de UI**: el interceptor *no* muestra toasts/diálogos; sólo prepara/filtra tráfico de red.

Orden de ejecución y encadenado Axios aplica primero los *request interceptors* (en orden de registro inverso) y, tras la respuesta, los *response interceptors*. Este orden permite:

1. Añadir cabeceras y serializar `params` antes de enviar.
2. Medir latencia y aplicar la política de reintento *después*.

Cabeceras y formato por defecto

```
api.interceptors.request.use(async (config) => {
  // Cabeceras "simples"
  config.headers = {
    Accept: 'application/json',
    ...config.headers,
  };
  if (['post','put','patch'].includes((config.method||'')
```

```
                                                .toLowerCase())) {
    config.headers['Content-Type'] = 'application/json';
  }

  // Auth: token siempre fresco
  const user = getAuth().currentUser;
  if (user) {
    const idToken = await user.getIdToken(); // renueva si caduca
    config.headers.Authorization = `Bearer ${idToken}`;
  }

  // Evitar enviar cookies (no usamos CSRF por cookies)
  config.withCredentials = false;
  return config;
});
```

Cancelación y *timeout* Las peticiones ligadas a vistas o modales deben ser cancelables para evitar fugas al desmontar componentes.

```
const ac = new AbortController();
const req = api.get('/goals', { signal: ac.signal, timeout: 10000 });
// ... si el usuario navega:
ac.abort(); // axios v1 respeta AbortController
```

Serialización de `params` (arrays y filtros) Por defecto Axios serializa `params` simples. Si envías arrays (`?id=a&id=b`) conviene fijar una estrategia estable:

```
// Opcion sin dependencias (arrays repetidos):
api.defaults.paramsSerializer = {
  serialize: (params) => {
    const usp = new URLSearchParams();
    Object.entries(params).forEach(([k,v]) => {
      Array.isArray(v) ? v
            .forEach(x => usp.append(k, x)) : usp.set(k, v);
    });
    return usp.toString();
  }
};
// Resultado: ?id=1&id=2&id=3
```

Concurrencia y refresco coordinado. Varias peticiones simultáneas pueden invocar `getIdToken()`. El SDK de Firebase coordina el **refresh** internamente: sólo una golpea la red y el resto esperan al mismo resultado. Por eso no es necesario (ni recomendable) implementar *locks* propios en el interceptor.

Registro (sólo en desarrollo). Útil para depurar latencias y "qué se mandó/recibió" sin exponer datos sensibles. Nunca se registra el token completo.

```
api.interceptors.request.use((cfg) => {
  if (import.meta.env.DEV) cfg.metadata = { t0: performance.now() };
  return cfg;
});
api.interceptors.response.use((res) => {
  if (import.meta.env.DEV && res.config.metadata) {
    const dt = (performance.now() - res.config.metadata.t0).toFixed(0);
```

```
    const method = (res.config.method || 'get').toUpperCase();
    console.debug(`[API] ${method} ${res.config.url} -> ${res.status} (${dt}ms)`);
  }
  return res;
});
```

8.3.3. Estado de sesión, guardas y persistencia multi-pestaña

El listener `onAuthStateChanged` informa de cambios de sesión; cuando el usuario es `null` se redirige a `/login`, y cuando existe se permite el acceso a rutas privadas (Dashboard, Profile). Esta estrategia evita pantallas intermedias con estado inconsistente.

Firebase almacena de forma segura las credenciales en *IndexedDB*/*localStorage* y sincroniza el estado entre pestañas. El *front end* no serializa tokens ni los expone en *query strings*.

8.3.4. Riesgos y mitigaciones

- **XSS y exposición del token**: políticas CSP, sanitización en UI y dependencia mínima de HTML peligroso; no se accede manualmente al almacenamiento del token.
- **Reloj del sistema desfasado**: el refresco automático mitiga expiraciones aparentes; el reintento forzado cubre respuestas 401 transitorias.
- **Revocación remota**: un 401 repetido provoca *sign out* y retorno a `/login`.

8.3.5. Resumen operativo

El token caduca aproximadamente cada hora y se *renueva automáticamente*. En esta implementación, `getIdToken()` garantiza que cada petición viaja con un token válido; `onAuthStateChanged` actúa como fuente de verdad de la sesión y evita estados intermedios. El manejo de 401 es conservador (un único reintento + logout) para priorizar seguridad y claridad de flujo.

8.4. Gestión de errores

La aplicación prioriza una experiencia clara y recuperable ante fallos. Se combinan salvaguardas en la **capa de red** (interceptores Axios + política 401), **UI/UX** (mensajes y diálogos consistentes) y **flujos críticos** con confirmación previa. Los principios a seguir son:

- **Claridad** para el usuario: mensaje corto, causa entendible y opción de reintento cuando procede.
- **Seguridad y consistencia**: nunca entrar en bucles de reintentos; no dejar estados parciales conocidos.
- **Responsividad** de la UI: bloqueo temporal del CTA mientras se resuelve; gestión del foco; `aria-live` en banners.
- **Registro y trazabilidad** durante desarrollo (consola y reporte en CI); extensible a telemetría/Sentry si se requiere.

8.4.1. Taxonomía de errores y respuesta

Tipo	Ejemplos	Respuesta UI/APP
Validación cliente	Campos obligatorios, formato de email, contraseña débil	Mensajes *inline* por campo; deshabilitar enviar
401/ Revocación	Sesión expirada o inválida	Refresco forzado + **un** reintento; si persiste: logout y `/login`
403 Permisos	Recurso prohibido	Banner "sin permisos"; ofrecer volver al `/dashboard`
404 Inexistente	Meta/registro eliminados	Banner + refresco de lista
409 Conflicto	Duplicados/estado inconsistente	Mensaje contextual y reintento guiado
Red (0)	Sin conexión / CORS	Banner "sin conexión"; reintento manual
5xx Servidor	Error interno o temporal	Diálogo no bloqueante + reintento; si persiste, contacto/soporte

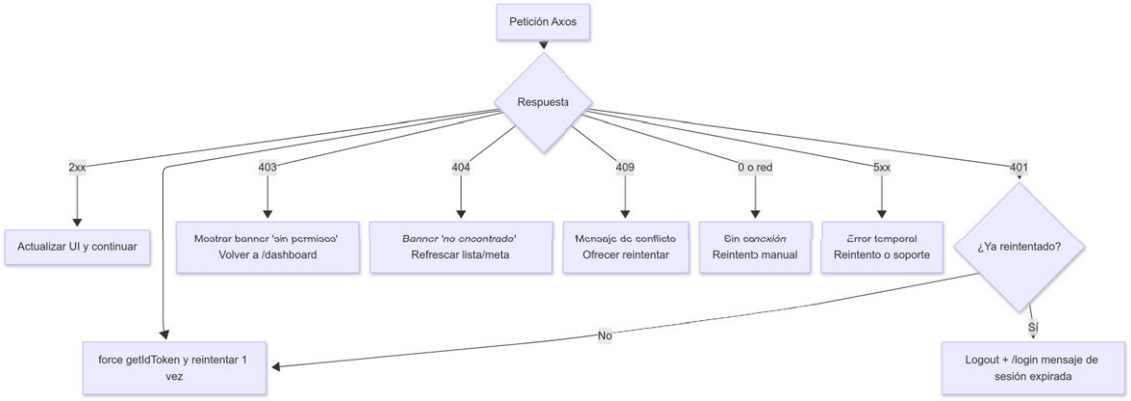


Figura 8.3: Decisión de errores en la capa de red

8.4.2. Capa de red: interceptores y política de reintentos

Todas las peticiones pasan por un servicio Axios que añade la cabecera `Authorization: Bearer ID_TOKEN` y trata centralizadamente los errores.

```
1 import axios from 'axios';
2 import { getAuth } from 'firebase/auth';
3
4 const api=axios.create({baseURL: import.meta.env.VITE_API_BASE_URL});
5
```

```
api.interceptors.request.use(async (config) => {
  const user = getAuth().currentUser;
  if (user) {
    // getIdToken() refresca automticamente si caduc (~1h)
    const idToken = await user.getIdToken();
    config.headers.Authorization = `Bearer ${idToken}`;
  }
  return config;
});

let retried401 = new WeakSet();

api.interceptors.response.use(
  r => r,
  async (error) => {
    const { response, config } = error;
    // 401 -> refresco forzado y un nico reintento
    if (response?.status === 401 && config &&
    !retried401.has(config)) {
      retried401.add(config);
      const user = getAuth().currentUser;
      if (user) {
        const fresh = await
        user.getIdToken(/* forceRefresh */ true);
        config.headers.Authorization = `Bearer ${fresh}`;
        return api.request(config);
      }
    }
    // Propagar: 403/404/409/5xx sern gestionados en
    la UI por caso de uso
    return Promise.reject(error);
  }
);

export default api;
```

Listado 8.4: Interceptor Axios: token y tratamiento de 401

8.4.3. UI/UX: mensajes, diálogos y recuperación

- **Diálogos de confirmación** en acciones destructivas (eliminar meta/registro, finalizar meta, eliminar cuenta). CTA deshabilitado durante la petición.
- **Banners/Toasts** para errores no críticos (404, 5xx temporales) con opción *Reintentar*.
- **Errores de formulario** *inline* por campo y explicación compacta (p. ej., "mínimo 12 caracteres, con mayúscula, minúscula, número y símbolo").
- **Accesibilidad**: foco retorna al elemento problemático; `role=.ªlert"` o `aria-live="polite"` en mensajes.

8.4.4. Casos críticos de dominio

Eliminar cuenta (doble fase) La operación requiere éxito en **Firebase Auth** y en el **back end**. La app aplica "todo o nada" a nivel de UI: si falla cualquiera, se muestra error y se permanece en `/profile`.

```
async function deleteAccountFlow() {
  setBusy(true);
  try {
    // 1)Solicitar borrado en back end
    //  (marca y borra datos de usuario)
    await api.delete('/me');
    // 2)Borrar usuario en Firebase (revoca tokens y sesin)
    await deleteUser(getAuth().currentUser);
    // 3)Cerrar sesin local y navegar a /login
    await signOut(getAuth());
    navigate('/login');
  } catch (e) {
    showError('No se pudo eliminar la cuenta. Intntalo ms tarde.');
  } finally {
    setBusy(false);
  }
}
```

Listado 8.5: Secuencia de eliminación con doble chequeo

Riesgo: si back end borra y Firebase falla, no es trivial "desborrar". La mitigación habitual sería una marca "pendiente de borrado" y proceso asíncrono; queda como mejora futura.

Crear/editar meta y registros: Sin actualizaciones optimistas, se actualiza la UI tras `200 OK`. Ante error de red/5xx, se preserva el formulario abierto con el dato introducido y se ofrece reintento.

8.4.5. Fallback global: límites de error

Para errores React no capturados (*runtime*), se propone un **Error Boundary** que muestre una pantalla de cortesía y reporte el fallo.

```
class ErrorBoundary extends React.Component {
  state = { hasError: false };
  static getDerivedStateFromError() { return { hasError: true }; }
  componentDidCatch(err, info) { console.error(err, info); }
  render() { return this.state.hasError ? <CrashScreen/> :
  this.props.children; }
}
```

Listado 8.6: ErrorBoundary mínimo

8.5. Tests

8.5.1. Objetivo y alcance

Los tests del *front end* buscan asegurar el comportamiento de los componentes y de la lógica de dominio sin depender del navegador real ni del *back end*. Se emplean pruebas **unitarias** con *Vitest* sobre *jsdom* y *Testing Library*, centradas en:

(i) componentes con interacción (modales y formularios)
(ii) utilidades puras
(iii) servicios ligeros

Las páginas orquestadoras (`Dashboard.jsx` y `Profile.jsx`) se excluyen de la suite unitaria por ser casi pruebas de integración (estado externo, navegación, modales y llamadas encadenadas); su verificación se propone con E2E (p. ej., Playwright/Cypress) para cubrir el flujo completo.

La ejecución usa Vitest con entorno `jsdom` y el proveedor de cobertura nativa `@vitest/coverage-v8`. Los *scripts* de `package.json` permiten ejecutar en local con o sin cobertura y modo CI en el pipeline.

```
thresholds: {
  global: { branches: 70, functions: 80, lines: 80, statements: 80 }
}
```

Listado 8.7: Umbrales de cobertura globales en Vitest

Decisiones de calidad:

- **Branches 70 %**: fuerza a cubrir caminos condicionales relevantes sin frenar el desarrollo cuando hay ramificaciones de UI difíciles de ejercitar.
- **Functions/Lines/Statements 80 %**: garantiza que la lógica ejecutable y las APIs de componentes quedan razonablemente cubiertas.
- Los ficheros puramente infra/estáticos (estilos, contenido legal, `firebase.js`) y las páginas con gran efecto lateral se excluyen explícitamente para que el indicador represente código de negocio real.

8.5.2. Integración en CI/CD

El *pipeline* de GitHub Actions ejecuta la batería `npm run test:ci` en cada `push`/`pull_request`. Si los umbrales de cobertura o alguna prueba fallan, el *job* `deploy_live` no se dispara (`needs: test`), bloqueando el despliegue.

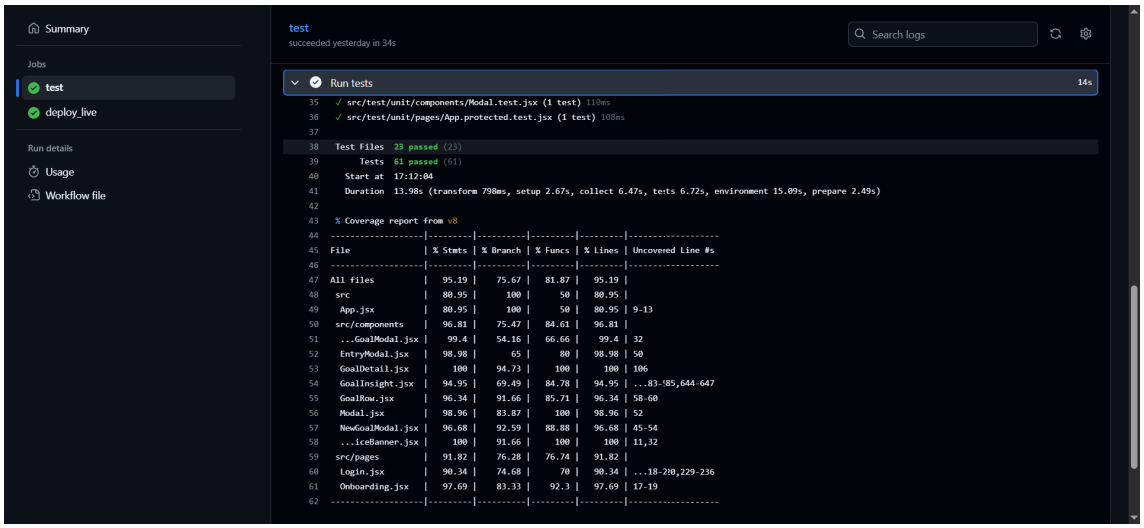


Figura 8.4: Ejecución de tests y reporte de cobertura en GitHub Actions.

8.5.3. Test unitarios

Qué se prueba

- **Componentes interactivos (modales y formularios)**: render, estados deshabilitado/activo, validaciones mínimas y eventos realistas con `@testing-library/user-event`.
- **Lógica pura y utilidades**: funciones sin efectos (formateos, validadores, mapeos DTO↔UI).
- **Servicios ligeros** (capas que orquestan llamadas): se aíslan con *mocks* de `fetch`/`axios` y de `getIdToken()` de Firebase Auth para verificar únicamente la lógica local (cabeceras, rutas, manejo de errores).

Patrones de prueba

- *Given-When-Then* para expresividad y mantenimiento.
- **Render helper** propio (si aplica) que envuelve en `AuthProvider` y `MemoryRouter` para no repetir *boilerplate*.
- **Mínimos de accesibilidad**: se consulta por rol/label (`getByRole`, `getByLabelText`) en lugar de selectores de clase.
- **Errores controlados**: se fuerzan respuestas `4xx/5xx` en los servicios para comprobar mensajes y reintentos.

Ejemplo representativo (servicio con token)

```
1  import { vi, describe, it, expect } from 'vitest'
2  import axios from 'axios'
3  import { getIdToken } from './mocks/firebase' // mock de getIdToken
4  import { fetchGoals } from '@/services/goals' // funcin bajo prueba
5
6  vi.mock('axios')
7  vi.mock('firebase/auth', () => ({
8    getAuth: () => ({}),
9    onAuthStateChanged: vi.fn(),
10   // getIdToken se exporta desde un mock local para simplificar
11 }))
12
13 describe('fetchGoals', () => {
14   it('enva Authorization: Bearer <ID_TOKEN> y devuelve datos',
15   async () => {
16     getIdToken.mockResolvedValue('ID_TOKEN_OK')
17     axios.get.mockResolvedValue({ data: [{ id: 'g1',
18     name: 'Meta' }] })
19
20     const data = await fetchGoals()
21
22     expect(axios.get).toHaveBeenCalledWith(
23       '/goals',
24       expect.objectContaining({
25         headers: expect.objectContaining({
```

```
26            Authorization: 'Bearer ID_TOKEN_OK',
27          }),
28        })
29      )
30      expect(data).toHaveLength(1)
31    })
32  })
```

Listado 8.8: Verificación de cabecera `Authorization` en un servicio

8.5.4. Resultados

Con la configuración anterior, la suite alcanza los umbrales definidos (70 % en ramas y 80 % en funciones/líneas/*statements*) y deja trazabilidad del informe LCOV/HTML en cada ejecución de CI.

Capítulo 9

Implementación de la API

9.1. Visión

En este capítulo presentamos la base conceptual de la API del proyecto. A partir de aquí, nos referiremos a ella como `GoLifeAPI`. El objetivo de la sección es enmarcar su papel dentro del sistema y sentar las ideas que guían su implementación.

9.1.1. Intención

La **intención** de `GoLifeAPI` es actuar como **operador y orquestador** del sistema cloud de GoLife, sirviendo de **columna vertebral** del sistema. Esto implica concentrar en un único punto: (1) la recepción de las acciones del usuario, (2) su validación y traducción a operaciones sobre los servicios en la nube, y (3) la devolución de respuestas coherentes y predecibles.

En términos de propósito, `GoLifeAPI` establece un **contrato único** para el producto, preserva la **consistencia** de los datos y las reglas, y coordina de forma ordenada la interacción con los componentes cloud.

9.1.2. Arquitectura Interna

`GoLifeAPI` Se trata de un servicio *REST* implementado con Spring: expone *endpoints* HTTP que reciben y devuelven **JSON**, aplicando validaciones, reglas de negocio y acceso a datos. Operativamente es **stateless**: el servidor no mantiene sesión entre peticiones.

Para favorecer un funcionamiento **entendible**, **seguible** y **mantenible**, `GoLifeAPI` adopta una **arquitectura por capas**, en la que los datos atraviesan cada una en la entrada y en la salida y son adaptados u operados según sus competencias. La Figura 9.1 muestra dicha arquitectura.

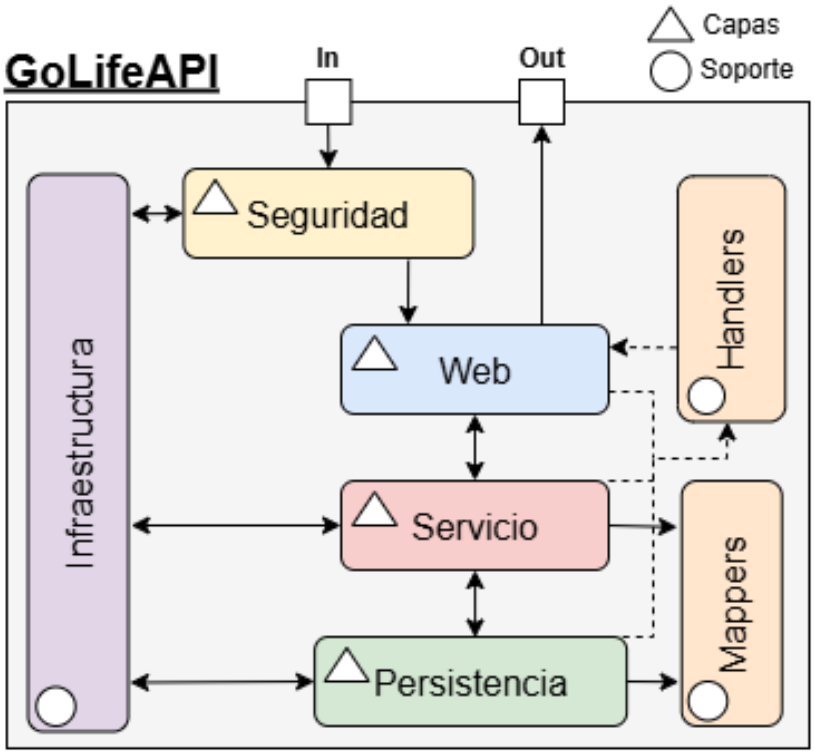


Figura 9.1: Arquitectura interna de `GoLifeAPI`

En el diagrama de la Figura 9.1 se observa el flujo principal a través de `seguridad` → `web` → `servicio` → `persistencia`. Las cajas representadas corresponden a los *packages* internos (carpetas) del software. Aunque estructuralmente estén al mismo nivel, solo `seguridad`, `web`, `servicio` y `persistencia` se consideran *capas* porque concentran el flujo principal de la información.

- `Seguridad`: verifica `headers` (`Authorization: Bearer ...`) y gestiona el `rate limiting` por usuario.
- `Web`: contiene los `REST controllers` y gestiona los `endpoints`.
- `Servicio`: aplica las **reglas de negocio** y de **dominio** sobre los datos.
- `Persistencia`: gestiona `documentos` (`bson`/`Document`) y ejecuta operaciones `CRUD` en la `base de datos`.

En cuanto al **soporte**, intervienen los siguientes *packages*:

- `Infraestructura`: centraliza las conexiones con servicios externos a la API y expone funciones básicas para operarlos (Cloud KMS, Firebase Auth y MongoDB Atlas).
- `Mappers`: realiza conversiones de tipos de datos para el paso entre capas.
- `Handlers`: define manejadores globales de errores; cualquier clase de las capas principales puede lanzar una excepción con mensaje contextual y esta "burbujea" hasta ser interceptada y formateada a un modelo de respuesta estándar, preservando mensaje y tipo de error.

Además de capas y soporte, existe un tercer grupo que no se refleja en el diagrama por no ser "operativo" en el sentido de orquestar el flujo, sino **estructural**:

- `model`: modelos de dominio base (p. ej., usuario, meta, registro).
- `dto`: modelos de *Data Transfer Object* de entrada/salida en `web`.
- `exception`: excepciones personalizadas para uso interno en la API.

Cada capa está diseñada para **operar de forma aislada** y ofrecer una **interfaz clara** a la capa inmediatamente superior. Esto favorece la **separación de responsabilidades**, facilita las **pruebas** (dobles de prueba por capa) y permite **evolucionar o sustituir** una capa sin afectar al resto. Este enfoque se refleja también en el **flujo de tipos de datos**: cada capa maneja un único tipo propio y transforma al siguiente. La Figura 9.2 lo ilustra.

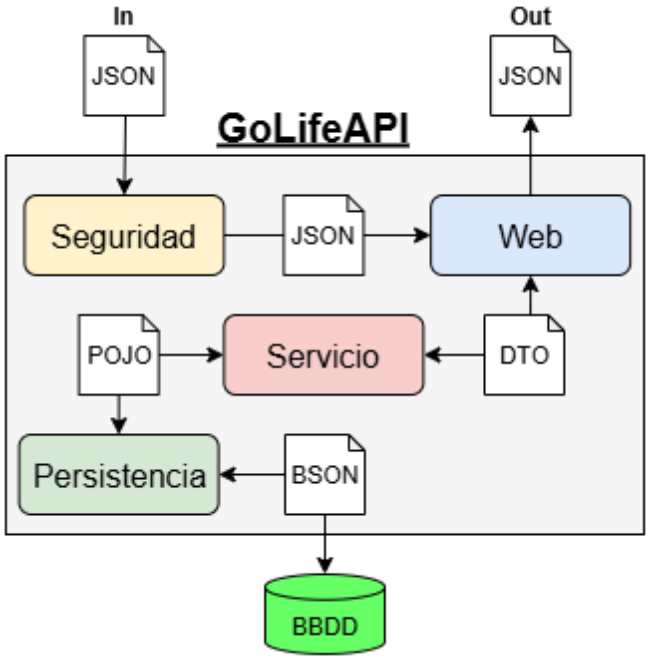


Figura 9.2: Flujo de tipos de datos a través de las capas en `GoLifeAPI`

La Figura 9.2 se centra en los tipos manejados y sus conversiones. `seguridad` recibe la petición (con *headers*) y valida la autenticación; `web` **parsea** el cuerpo **JSON** a `dto` y delega; `servicio` transforma `dto` en `pojo` (modelos del `model`) y aplica la lógica; `persistencia` convierte `pojo` a `bson`, el formato binario usado por MongoDB, ejecuta la operación en la base de datos y retorna hacia arriba el dato convertido en sentido inverso, hasta que `web` vuelve a serializar a **JSON** para la respuesta.

9.2. Operativa de cada capa

En esta sección describimos cada capa principal de `GoLifeAPI`. El foco está en el flujo `seguridad` → `web` → `servicio` → `persistencia`, explicando **qué hace** cada una, **qué datos maneja** y **cómo colabora** con las demás.

9.2.1. Seguridad

La capa `seguridad` valida las credenciales de cada petición, aplica `rate limiting` por usuario, configura `CORS` y asegura un funcionamiento **stateless** (sin sesión en servidor). En `GoLifeAPI` esto se implementa con dos cadenas de filtros (`SecurityFilterChain` declarado en `SecurityConfig`): una **pública** para rutas de salud/documentación y otra **protegida** para el resto de `/api/**`. La segunda fuerza autenticación, desactiva CSRF (ya que usamos el `Token JWT`), fija `SessionCreationPolicy.STATELESS` y ordena los filtros: primero el filtro de **autenticación Firebase** y, a continuación, el de **rate limiting** por usuario.

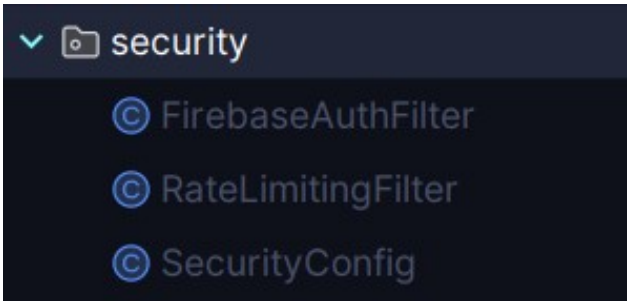


Figura 9.3: Ficheros de la capa `Seguridad` en `GoLifeAPI`

Autenticación Bearer JWT (Firebase)

`FirebaseAuthFilter` comprueba `Authorization: Bearer <ID Token (JWT)>`. Si falta o es inválido, responde `401` directamente; si es correcto, el `SDK de Firebase`, a través de `FirebaseService` (`infraestructura`), verifica el token y se establece en el `SecurityContext` un `Authentication` con el `uid` del usuario.

El filtro no altera el `JSON` del *body*; sólo controla el acceso y se registra **antes** del `rate limiting` para que éste pueda limitar por `uid`.

A continuación se muestra un extracto que ilustra este flujo:

```
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain)
    throws ServletException, IOException {

  String authHeader = request.getHeader("Authorization");
  if (authHeader == null || !authHeader.startsWith("Bearer ")) {
    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
    response.getWriter().write("Falta el encabezado Authorization");
    return;
  }

  String uid = firebaseService.verifyBearerToken(authHeader);
  if (uid == null) {
    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
    response.getWriter().write("Token invlido");
    return;
  }

  Authentication authentication = new UsernamePasswordAuthenticationToken(
```

```
                                    uid, null, List.of());
  SecurityContextHolder.getContext().setAuthentication(authentication);

  filterChain.doFilter(request, response);
}
```

Listado 9.1: Filtro de autenticación Bearer ID Token (JWT)

Rate limiting por usuario (Bucket4j)
Para proteger la capacidad de la base de datos, `RateLimitingFilter` limita las peticiones **por** `uid` usando un *bucket* por usuario en caché. La política es un cubo de **capacidad 30** con **relleno** de `1` ficha cada `2 s`, con una duración máxima en caché de hasta 15 minutos.

Las peticiones consumen una ficha; si no hay disponibilidad se responde `429 Too Many Requests`. Este patrón (token bucket) actúa como "ventana deslizante suave", permitiendo ráfagas acotadas y recuperación progresiva.

Un umbral razonable se puede estimar desde la capacidad total de la base de datos:

$$\textit{límite\_por\_usuario} = \frac{\textit{ops\_por\_segundo} \times 60}{\textit{usuarios\_concurrentes}} \times \textit{factor\_seguridad}.$$

Para un plan M0 de MongoDB Atlas (`100 ops/s`), `100` usuarios concurrentes y un `50 %` de margen:

$$\textbf{30 ops/min} = \left(\frac{100 \times 60}{100}\right) \times 0{,}5.$$

A este ritmo, un usuario puede realizar en promedio una operación cada $\sim$2 s sin `throttling`, en línea con prácticas habituales de *rate limiting* en APIs (113).

A continuación se muestra la configuración empleada por el filtro:

```
private final Cache<String, Bucket> userBuckets = Caffeine.newBuilder()
    .expireAfterAccess(Duration.ofMinutes(15))
    .maximumSize(100)
    .build();

private Bucket createNewBucket() {
  return Bucket.builder()
      .addLimit(limit -> limit
          .capacity(30)
          .refillIntervally(1, Duration.ofSeconds(2))
      ).build();
}
```

Listado 9.2: Configuración del cubo de *rate limiting*

En conjunto, `seguridad` actúa como puerta primera de entrada: **verifica credenciales**, **aplica límites de uso** y **prepara el contexto** (`uid/claims`) para el resto de capas, manteniendo la API **stateless** y protegida sin alterar el `JSON` de la petición.

9.2.2. Web

La capa `web` es la **puerta HTTP** de `GoLifeAPI`: expone los `endpoints REST`, recibe y devuelve `json`, realiza la **validación básica** de lo que llega y traduce ese `json` a objetos simples de intercambio antes de pasar la petición a `servicio`. También recupera el `uid` autenticado (aportado por `seguridad`) y devuelve códigos `http` claros (`201` cuando se crea

algo, 200 al consultar o actualizar, etc.). Si la entrada no es válida, responde con un error 400 comprensible.

Figura 9.4: Ficheros de la capa `web` en `GoLifeAPI`

Controladores y responsabilidades

- `UserRestController`: gestiona el **perfil de usuario** en `/api/usuarios` (alta, consulta, actualización y borrado).
- `GoalRestController`: ofrece las operaciones de **metas** en `/api/metas` (crear, leer, actualizar, finalizar y borrar), tanto para metas **booleanas** como **numéricas**.
- `RecordRestController`: maneja los **registros** asociados a cada meta en `/api/metas/{mid}/registros` (añadir y eliminar entradas).
- `HealthRestController`: publica `/api/salud`, una **comprobación pública** para saber si la API y sus dependencias están disponibles.

El siguiente extracto de `GoalRestController` ilustra el patrón típico: entrada `json` → **DTO validado** (`@Valid`) → delegación a `servicio` y respuesta con `201 Created` y el cuerpo actualizado del usuario/metas.

```
@PostMapping("/bool")
public ResponseEntity<ResponseUserDTO> postMetaBool(
                @AuthenticationPrincipal String uid,
                @Valid @RequestBody CreateBoolGoalDTO createBoolGoalDTO) {
    return ResponseEntity.status(HttpStatus.CREATED)
        .body(goalService.createBoolGoal(createBoolGoalDTO, uid));
}
```

Listado 9.3: Endpoint POST de metas booleanas

Si durante el flujo se produce algún **error de validación o de ejecución**, éste se lanza **en su punto de origen** (validación de `@Valid`, reglas en `servicio`, acceso a `persistencia`) y es formateado por los `handlers` globales. Por eso el endpoint puede mantenerse **simple y fino**: recibe el `JSON`, valida el `DTO` y delega; los errores no se capturan aquí, sino que `handlers` los convierten en respuestas HTTP coherentes.

En síntesis, `web` cumple su papel de **puerta HTTP** garantizando un **contrato claro** (json ↔ dto, códigos coherentes) y **separación de responsabilidades**: valida lo esencial y delega la lógica en `servicio`, mientras los `handlers` unifican los errores. El resultado son controladores **delgados**, un flujo **predecible** y una base **fácil de mantener y extender**.

9.2.3. Servicio

La capa `servicio` conecta la `web` con `persistencia` y es donde se aplican las **reglas de negocio**: comprueba que los datos solicitados **pertenecen al usuario** autenticado, valida el **tipo de meta** (booleana/numérica) antes de operar, evita cambios sobre metas **finalizadas** y calcula/actualiza **atributos derivados** (fechas de fin, contadores, indicadores). Para ello, orquesta mapeos (dto ↔ modelo), colabora con servicios de soporte (`KeyManagementService` para firmar/verificar `uid`) y delega las operaciones de almacenamiento en `persistencia`.

La capa se organiza en **interfaces** (`IUserService`, `IGoalService`, `IRecordService`) y sus **implementaciones**, lo que facilita el acoplamiento con `web` y la prueba aislada de la lógica.

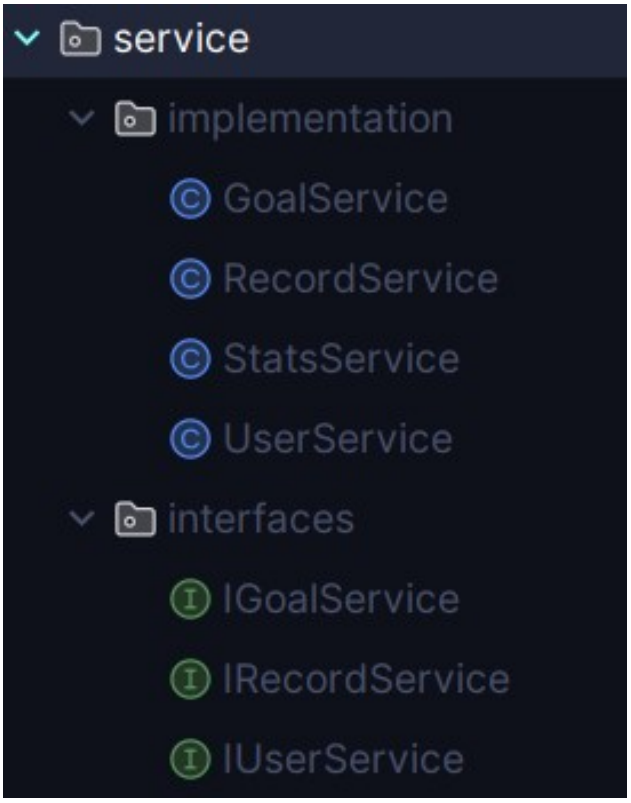


Figura 9.5: Ficheros de la capa `servicio` en `GoLifeAPI`

En términos funcionales:

- `UserService`: gestiona el **ciclo de vida del usuario** y su representación de salida.
- `GoalService`: centraliza la **operativa de metas** (crear, leer, actualizar, finalizar, borrar) y verifica la **pertenencia** antes de cada acción.
- `RecordService`: añade/elimina **registros** garantizando **fecha única** y coherencia con la meta.
- `StatsService`: calcula **atributos derivados** (fecha de fin según duración/unidad, primera entrada, variaciones de contadores, flags) que se **computan** antes de almacenarse o que directamente no se almacenan "a mano", sino que se **computan** cuando corresponde.

Como ejemplo representativo del trabajo en `servicio`, la operación de alta de una **meta booleana** muestra bien su papel: aplica reglas de negocio (cálculo de fecha final según duración), asocia la meta al `uid` del usuario (firma a traves de `keyManagementService` de `infraestructura`), realiza los **mapeos** necesarios (DTO ↔ modelo (POJO)) y coordina la actualización de **estadísticas** del usuario al persistir el cambio. Todo ello deja a `web` libre de lógica y concentra aquí la coherencia del dominio.

```
@Override
public ResponseUserDTO createBoolGoal(CreateBoolGoalDTO dto, String uid) {
    LocalDate finalDate = statsService.calculateFinalGoalDate(
            dto.getFecha(),
            dto.getDuracionValor(),
            dto.getDuracionUnidad());
    String signedUid = keyManagementService.sign(uid);
    BoolGoal goal = goalDtoMapper
            .mapCreateBoolGoalDtoToBoolGoal(dto, signedUid, finalDate);
    return userDtoMapper.mapUserToResponseUserDTO(
            goalPersistenceController.createBoolGoal(
                    goal,
                    statsService.getUserStatsUpdateDoc(+1, 0),
                    signedUid));
}
```

Listado 9.4: Servicio: alta de meta booleana (orquestación y reglas)

En resumen, `servicio` **orquesta** el flujo: calcula datos derivados, valida y vincula la operación al usuario, transforma objetos y delega en `persistencia`; devuelve a `web` un resultado ya preparado para responder. Si aparece alguna incoherencia (p. ej., datos incompatibles o reglas incumplidas), aquí se lanza la excepción correspondiente para que los `handlers` la conviertan en una respuesta HTTP uniforme.

9.2.4. Persistencia

La capa `persistencia` es la responsable de **guardar y recuperar** la información desde la base de datos. Está diseñada para que sus **controladores de persistencia** conozcan **lo mínimo** del motor de datos: actúan como *orquestadores* que reciben modelos del dominio (POJO), los convierten a documentos y delegan las operaciones concretas en los **DAO**, que sí encapsulan los detalles de MongoDB (inserciones, actualizaciones, borrados y manejo de documentos embebidos/listas).

Cada operación se ejecuta dentro de una **transacción** para que el conjunto de pasos sea **atómico** (todo o nada), lo que protege la consistencia cuando, por ejemplo, se crea una meta y a la vez se actualizan las estadísticas del usuario. Este patrón se observa en los controladores `UserPersistenceController`, `GoalPersistenceController` y `RecordPersistenceController` (orquestación + transacción) y en `UserDAO`/`GoalDAO` (operaciones específicas de MongoDB). La ejecución transaccional la gestiona `TransactionRunner` con opciones de lectura/escritura seguras.

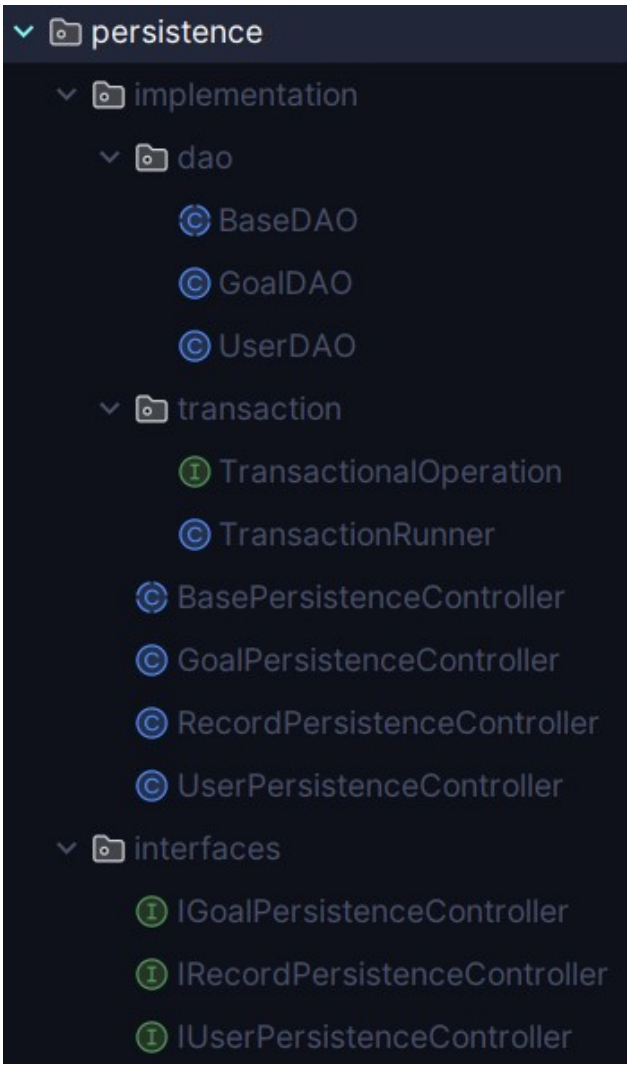


Figura 9.6: Ficheros de la capa `persistencia` en `GoLifeAPI`

Estructura y responsabilidades

- **Interfaces e implementaciones:** los controladores (`IUserPersistenceController`, `IGoalPersistenceController`, `IRecordPersistenceController`) exponen operaciones claras; sus implementaciones orquestan mapeos y transacciones.
- **DAO:** encapsulan el acceso de bajo nivel a MongoDB (consultas, `findOneAndUpdate`, inserción en listas embebidas, borrados, etc) para `Users` y `Goals`. Los controladores no construyen consultas: llaman a métodos de los DAOs.
- **Transacciones:** cada operación usa `TransactionRunner.run(...)` para asegurar atomicidad; si algo falla, se **revierte** y no queda estado intermedio.

Para ilustrar la **orquestación** típica en `persistencia` con interacción real con **DAO** y servicios de soporte, el siguiente método de borrado ejecuta, **en una única operación coordinada**, la eliminación del documento de usuario, el borrado en bloque de sus metas y, por último, la baja en **Firebase Auth**. La coherencia en base de datos se garantiza con

`TransactionRunner`: si algo falla en cualquier paso, se **aborta** la transacción y no queda estado intermedio; la excepción asciende para que las capas superiores respondan de forma uniforme.

```
@Override
public void delete(String dbUid, String fbUid) {
    try {
        transactionRunner.run(session -> {
            DeleteResult deleteUser = userDAO
                                        .deleteUserByUid(session, dbUid);
            if (!deleteUser.wasAcknowledged())
                throw new RuntimeException();
            if (deleteUser.getDeletedCount() == 0)
                throw new NotFoundException("Usuario no encontrado");

            DeleteResult deleteGoals = goalDAO
                                      .deleteManyGoalsByUid(session, dbUid);
            if (!deleteGoals.wasAcknowledged())
                throw new RuntimeException();

            if (!firebaseService.deleteFirebaseUser(fbUid))
                throw new RuntimeException();
            return null;
        });
    } catch (NotFoundException e) {
        throw e;
    } catch (RuntimeException e) {
        throw new RuntimeException("Error interno al borrar el usuario", e);
    }
}
```

Listado 9.5: Persistencia: borrado de usuario

Para simplificar y unificar el manejo de transacciones, `TransactionRunner` centraliza la apertura de sesión, el inicio/confirmación (o aborto) de la transacción y el cierre de recursos. Obtiene la **sesión de MongoDB Atlas** a través de `MongoService` (`infraestructura`) y ejecuta la operación que le pasa el controlador de persistencia; si todo va bien, hace `commit`, si algo falla, `abort` y propaga la excepción. Así, los controladores sólo expresan la *intención* y no se ocupan de la mecánica transaccional.

```
public <T> T run(TransactionalOperation<T> op) {
    ClientSession session = mongoService.getStartedSession();
    session.startTransaction(buildTransactionOptions());
    try {
        T result = op.apply(session);
        session.commitTransaction();
        return result;
    } catch (Exception e) {
        session.abortTransaction();
        throw e;
    } finally {
        session.close();
    }
}
```

Listado 9.6: `TransactionRunner`: apertura de sesión, commit/abort y cierre

Como ejemplo mínimo de `DAO` especializado, el siguiente método de `GoalDAO` elimina en bloque las metas asociadas a un `uid`; se ejecuta con la **sesión transaccional** que le cede el controlador de persistencia y que este a su vez recibe del `TransactionRunner`.

```
1 public DeleteResult deleteManyGoalsByUid(ClientSession session, String uid){
2     return mongoClient.getDatabase(DATABASE_NAME)
3             .getCollection(COLLECTION_NAME)
4             .deleteMany(session, new Document(USER_ID_NAME, uid));
5 }
```

Listado 9.7: GoalDAO: borrado de metas de usuario

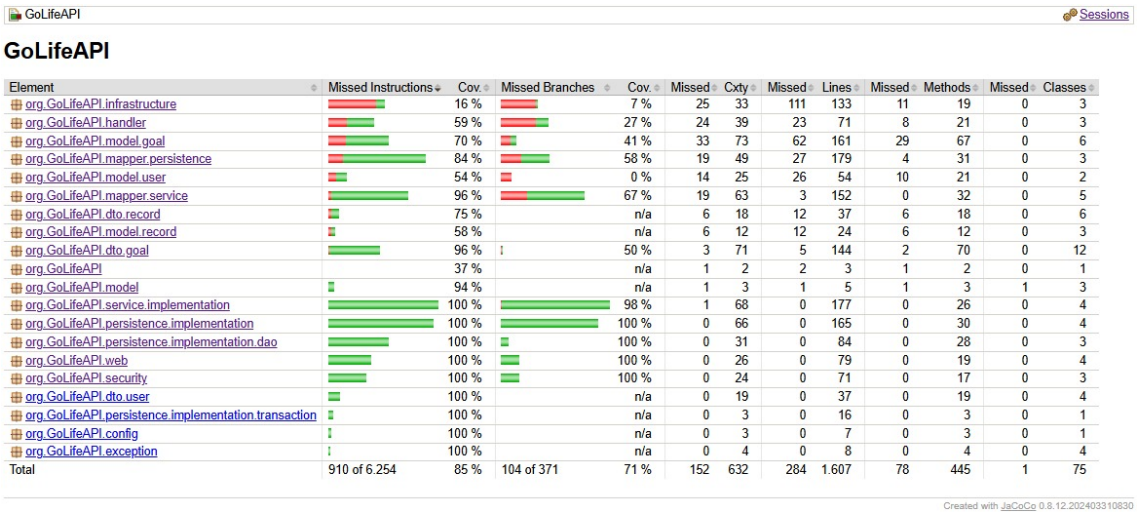
En resumen, `persistencia` mantiene un **contrato con la base de datos** sencillo y seguro: los controladores orquestan operaciones **atómicas** apoyándose en mapeadores y DAO especializados (que encapsulan las consultas de MongoDB), mientras `TransactionRunner` asegura la **consistencia** incluso cuando intervienen varios pasos relacionados.

9.3. Tests

En el **pipeline CI/CD de la GoLifeAPI** se ejecutan dos baterías complementarias de pruebas que validan tanto la lógica como el comportamiento real de la API.

- **Tests mixtos (unidad + integración):** centrados en reglas de dominio, servicios y la interacción con repositorios o dependencias externas mediante dobles de prueba o mockeos; cubren tanto casos "felices" como de error.
- **Tests end-to-end (E2E):** ejercen los endpoints HTTP reales de la API contra un entorno/datos de prueba, verificando contratos, flujos completos y códigos de estado; se centra en casos "felices".

La **cobertura de código** reportada por *JaCoCo* alcanza el **85 %** de instrucciones (5 344/6 254) y el **71 %** de ramas (267/371), como se aprecia en la Figura 9.7.


GoLifeAPI

Sessions

GoLifeAPI

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.GoLifeAPI.infrastructure		16 %		7 %	25	33	111	133	11	19	0	3
org.GoLifeAPI.handler		59 %		27 %	24	39	23	71	8	21	0	3
org.GoLifeAPI.model.goal		70 %		41 %	33	73	62	161	29	67	0	6
org.GoLifeAPI.mapper.persistence		84 %		58 %	19	49	27	179	4	31	0	3
org.GoLifeAPI.model.user		54 %		0 %	14	25	26	54	10	21	0	2
org.GoLifeAPI.mapper.service		96 %		67 %	19	63	3	152	0	32	0	5
org.GoLifeAPI.dto.record		75 %		n/a	6	18	12	37	6	18	0	6
org.GoLifeAPI.model.record		58 %		n/a	6	12	12	24	6	12	0	3
org.GoLifeAPI.dto.goal		96 %		50 %	3	71	5	144	2	70	0	12
org.GoLifeAPI		37 %		n/a	1	2	2	3	1	2	0	1
org.GoLifeAPI.model		94 %		n/a	1	3	1	5	1	3	1	3
org.GoLifeAPI.service.implementation		100 %		98 %	1	68	0	177	0	26	0	4
org.GoLifeAPI.persistence.implementation		100 %		100 %	0	66	0	165	0	30	0	4
org.GoLifeAPI.persistence.implementation.dao		100 %		100 %	0	31	0	84	0	28	0	3
org.GoLifeAPI.web		100 %		100 %	0	26	0	79	0	19	0	4
org.GoLifeAPI.security		100 %		100 %	0	24	0	71	0	17	0	3
org.GoLifeAPI.dto.user		100 %		n/a	0	19	0	37	0	19	0	4
org.GoLifeAPI.persistence.implementation.transaction		100 %		n/a	0	3	0	16	0	3	0	1
org.GoLifeAPI.config		100 %		n/a	0	3	0	7	0	3	0	1
org.GoLifeAPI.exception		100 %		n/a	0	4	0	8	0	4	0	4
Total	910 of 6.254	85 %	104 of 371	71 %	152	632	284	1.607	78	445	1	75

Created with JaCoCo 0.8.12.202403310830

Figura 9.7: Reporte de cobertura JaCoCo de la API

Todas las pruebas siguen el **patrón AAA (Arrange–Act–Assert)**: **Arrange** prepara el escenario (datos, *mocks* y estado del SUT), **Act** ejecuta la operación o *endpoint* a

validar y **Assert** comprueba resultados y efectos; este esquema mejora la **legibilidad**, **mantenibilidad** y reduce la **fragilidad** de los tests.

En las subsecciones siguientes se detalla cada batería, su alcance, datos y su aportación a la calidad global de GoLifeAPI.

9.3.1. Tests Mixtos

Esta batería se denomina **mixta** porque combina **varios tipos de tests** para cubrir de forma ligera y rápida las capas cruciales de la API. En total suma **252 pruebas**, constituyendo la **gran mayoría** del conjunto global por su bajo coste de ejecución.

Se testean las capas de **seguridad**, **web**, **servicio** y **persistencia**. Dado que en producción dependemos de *servicios gestionados*, en esta batería dichos servicios se **simulan con dobles de prueba** (*mocks/spies*), **excepto** en los tests de persistencia, donde se levanta una **instancia efímera de MongoDB con Docker**.

Como referencia, la Figura 9.8 muestra una ejecución local de esta batería con el recuento total de pruebas.

Figura 9.8: Ejecución local de la batería de tests mixtos (252 casos)

Tipos de test y foco:

- **Web (MockMVC / contrato HTTP):** validan controladores, validaciones de entrada, manejo de excepciones y forma de las respuestas JSON (códigos, `content-type`, campos), aislando la capa web al *mockear* el servicio.
  - **Foco:** contratos HTTP estables, validaciones claras y mensajes de error consistentes.
- **Servicio (unidad con mocks/spies):** comprueban reglas de dominio, mapeos DTO a modelo y viceversa, orquestación con persistencia y tratamiento de errores (`NotFound`, etc.).
  - **Foco:** lógica de negocio correcta, aislada de infraestructura.
- **Seguridad (unidad / filtros):** prueban autenticación basada en **Bearer JWT** a nivel de filtro: token válido → `200`; token inválido/ausente → `401`; y casos límite del encabezado `Authorization` (vacío, esquema no `Bearer`, mayúsculas/minúsculas, espacio tras `Bearer`). Además, se cubre el *rate-limiting* por usuario (fuera del umbral→`429`) como protección transversal.

  - **Foco:** verificación robusta del encabezado `Authorization` y políticas de control de abuso coherentes.

- **Persistencia (integración ligera):** ejecutan operaciones reales contra **MongoDB en Docker** mediante Testcontainers; verifican CRUD, índices/ filtrado y mapeo de errores (duplicados, no encontrados, borrados parciales). Estos tests no *mockean* la base de datos, por lo que se consideran de *integración*.

  - **Foco:** fiabilidad de acceso a datos y consistencia entre DAO/controlador de persistencia y el esquema real.

A modo de muestra, se incluye un test de la capa de **servicio** en el Listado 9.8, que ilustra claramente el enfoque **Arrange–Act–Assert**: intenta actualizar una meta booleana ya finalizada y verifica que se lanza `ConflictException` y que **no** se tratan de realizan escrituras en la capa de persistencia.

```
1  @Test
2  public void updateBoolGoal_whenFinalized_throwsConflictException() {
3    PatchBoolGoalDTO dto = new PatchBoolGoalDTO();
4    dto.setNombre("nuevoNombre");
5
6    BoolGoal goal = new BoolGoal(signedUid, new ObjectId(), "n", "d",
7        LocalDate.of(2025, 7, 1), true, 5, Enums.Duracion.Dias,
8        new GoalStats(false, LocalDate.of(2025, 7, 1)), new ArrayList<>());
9    when(goalPersistenceController.read(eq(mid))).thenReturn(goal);
10
11   Assertions.assertThatThrownBy(() ->
12       goalService.updateBoolGoal(dto, uid, mid)
13   ).isInstanceOf(ConflictException.class)
14    .hasMessage("La meta ya esta finalizada, no puedes modificarla");
15
16   verify(goalPersistenceController, never()).updateWithGoalStats(
17       any(Document.class), any(Document.class), any(Document.class),
18       eq(signedUid), eq(mid)
19   );
20 }
```

Listado 9.8: Tests mixtos: actualización de meta booleana ya finalizada

La combinación de pruebas rápidas por capa permite **detectar regresiones** en contratos HTTP, **anclar la lógica de negocio** con entradas realistas y **garantizar la robustez** de la persistencia en escenarios normales y de fallo.

El uso de dobles de prueba en seguridad/servicios acelera el ciclo en CI, mientras que la verificación con **MongoDB real en contenedor** aporta confianza en la interacción con la base de datos sin penalizar en exceso los tiempos de *pipeline*. En conjunto, esta batería proporciona una **cobertura amplia y eficiente** de la API, ideal como primera línea de defensa antes de los E2E.

9.3.2. Tests E2E

Esta batería **ejecuta los endpoints reales de la API** contra un entorno de prueba controlado para validar **flujos completos** de usuario, metas y registros. Consta de **22 pruebas** centradas en *happy paths* y verifican códigos HTTP, estructura del JSON y evolución coherente del estado (p. ej., estadísticas tras crear/finalizar/borrar).

Durante toda su ejecución se **simulan con *mocks*/*spies*** los componentes externos (p. ej., autenticación y firma), y se **levanta una máquina virtual de MongoDB 8.0.11 en Docker** mediante Testcontainers, que actúa como base de datos real para comprobar el flujo extremo a extremo. La Figura 9.9 muestra una ejecución local con el recuento total, y la Figura 9.10 la ejecución del contenedor de MongoDB.

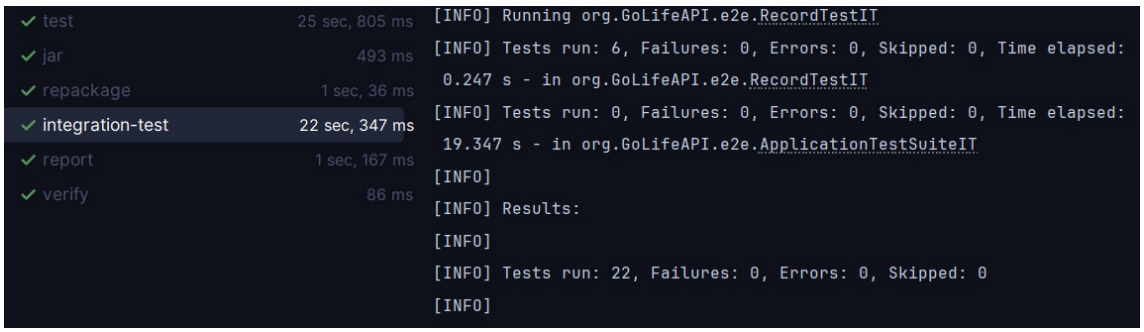


Figura 9.9: Ejecución local de la batería de tests E2E (22 casos)

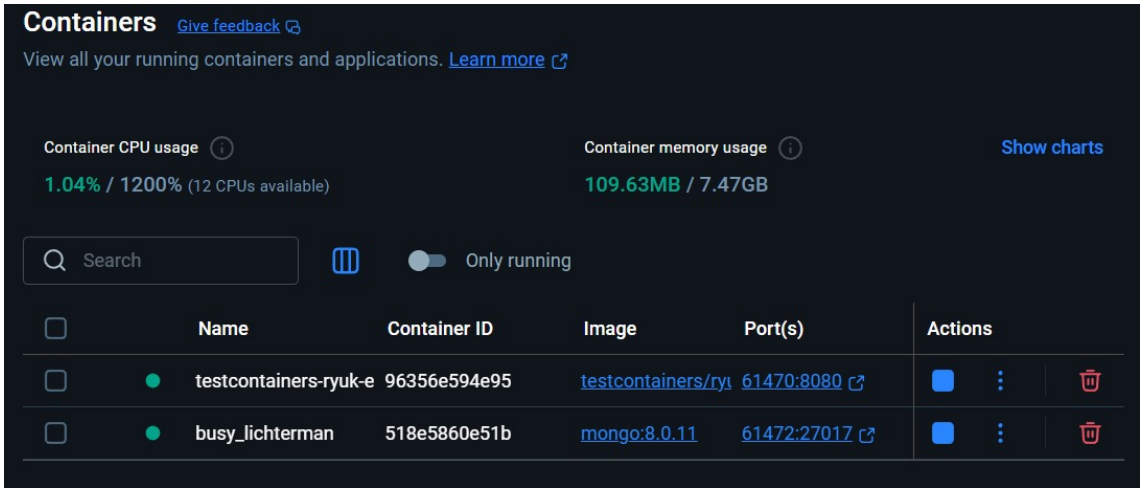


Figura 9.10: Instancia de MongoDB 8.0.11 en Docker durante los E2E

Ámbito y foco:

- **Entorno y dependencias:** Configuración común que **falsea** Firebase y el KMS, fija el `uid` de pruebas y proporciona un **MongoService real** con la URI del contenedor (Testcontainers).
  - **Foco:** reproducibilidad y aislamiento del entorno con base de datos real.
- **Usuarios (`/api/usuarios`):** creación, actualización, lectura y borrado, comprobando que el cuerpo de respuesta y las estadísticas asociadas sean consistentes.
  - **Foco:** contrato HTTP y estado inicial del sistema de un usuario.
- **Metas (`/api/metas` Bool/Num):** alta, modificación parcial, lectura, finalización y borrado; se valida la **evolución de estadísticas** (totales, finalizadas, porcentaje) y la correcta tipificación *Bool/Num*.

  - **Foco:** flujos nominales y efectos sobre el agregado de estadísticas.

- **Registros (`/api/metas/{mid}/registros`):** alta/borrado de registros para metas Bool/Num, reflejando el primer registro y valores consignados.

  - **Foco:** trazabilidad de entradas y coherencia de campos derivados.

- **Secuenciación:** uso de `@TestMethodOrder` y `@Order` para encadenar operaciones (crear→ modificar→ consultar→ finalizar→ borrar) y `JsonPath` para recuperar `mid` generados durante la ejecución.

  - **Foco:** validación del *journey* completo por recurso.

Para ilustrar el enfoque **Arrange–Act–Assert** en los E2E, el Listado 9.9 muestra el *happy path* de alta de usuario: se prepara el *payload*, se invoca el *endpoint* real con `Bearer` válido y se asertan el **201** y el **contrato JSON** de estado inicial.

```
@Order(1)
@Test
public void e2e_postUsuario_returns201_andJsonBody() throws Exception {
  String payload = "{\"nombre\":\"test-nombre\"," +
                   "\"apellidos\":\"test-apellidos\"}";

  ResultActions response = mockMvc.perform(post("/api/usuarios")
      .header("Authorization", "Bearer good.token")
      .contentType(MediaType.APPLICATION_JSON)
      .content(payload));

  response.andExpect(status().isCreated())
      .andExpect(jsonPath("$.nombre").value("test-nombre"))
      .andExpect(jsonPath("$.apellidos").value("test-apellidos"))
      .andExpect(jsonPath("$.metas").isEmpty())
      .andExpect(jsonPath("$.estadisticas").isNotEmpty())
      .andExpect(jsonPath("$.estadisticas.totalMetas").value(0))
      .andExpect(jsonPath("$.estadisticas.totalMetasFinalizadas").value(0))
      .andExpect(jsonPath("$.estadisticas.porcentajeFinalizadas").value(0));
}
```

Listado 9.9: Tests E2E: alta de usuario

Los E2E **complementan** a los tests mixtos al verificar los **flujos reales** con una **base de datos en ejecución**, garantizando contratos, integridad de datos y consistencia del estado a través de capas. Mantener *mocks/spies* en servicios externos acelera la ejecución y elimina variables no controladas, mientras que MongoDB en Docker asegura que los caminos nominales funcionan sobre persistencia real antes del despliegue.

Capítulo 10

Evaluación y resultados

10.1. Aplicación web resultante: GoLife

La aplicación resultante es **GoLife**: una web **100,% cloud** pensada para que cualquier persona **registre y siga sus metas** sin fricción. Su propuesta es simple: crear metas en segundos, registrar avances con rapidez y visualizar el progreso de forma clara y motivadora. Está orientada al uso desde **navegador de escritorio**.

GoLife se organiza en cuatro flujos o pantallas principales: **Login**, **Onboarding**, **Dashboard** y **Profile**. A continuación se muestran capturas representativas de cada flujo; con una breve explicación por imagen.

10.1.1. Login

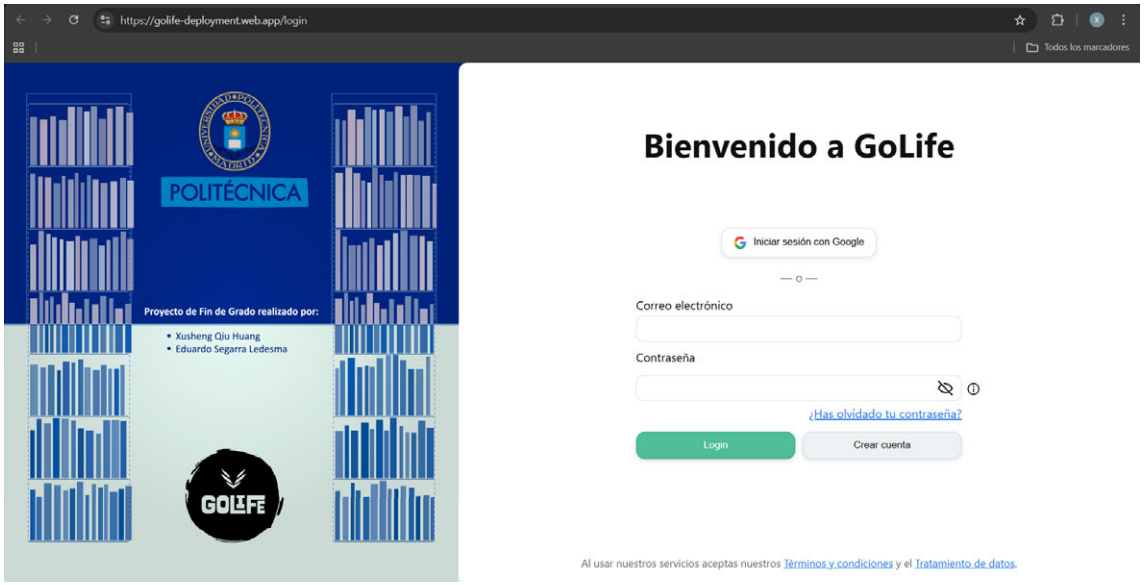


Figura 10.1: Pantalla de acceso con email/contraseña e inicio con Google

El pie enlaza Términos y Tratamiento de datos para transparencia previa. El botón de Login permanece deshabilitado hasta validar el formato básico. Diseño a dos columnas con bienvenida y formulario destacado.

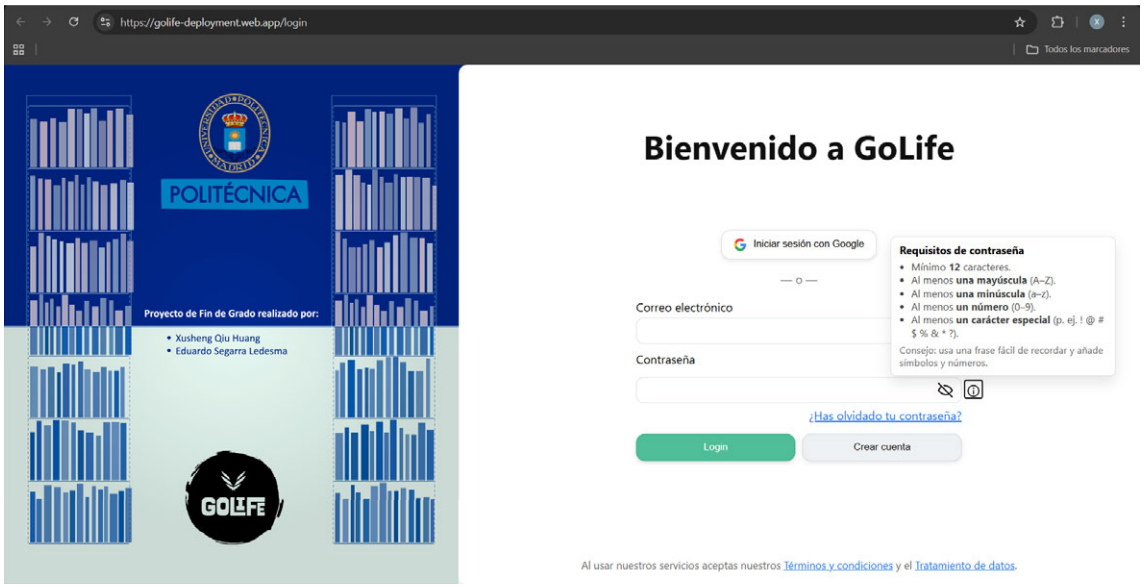


Figura 10.2: Ayuda contextual con requisitos mínimos de contraseña

Debe incluir mayúscula, minúscula, número y carácter especial. Se muestra al pulsar el icono de información junto al campo de contraseña. Refuerza credenciales fuertes y reduce errores de alta.

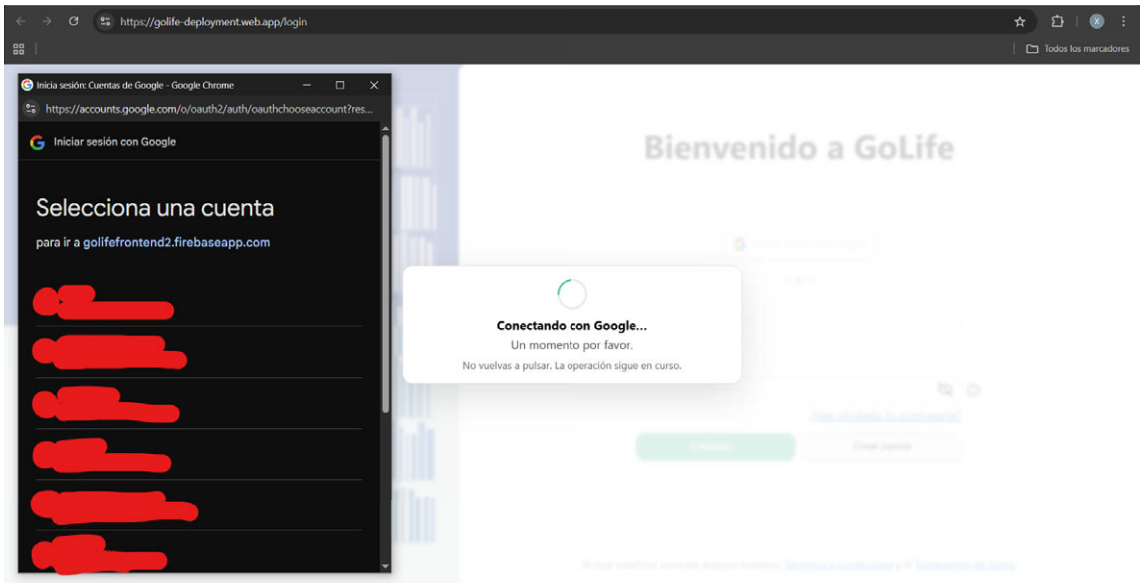


Figura 10.3: Flujo federado con Google

Estado intermedio “Conectando con Google. . . ” bloquea interacciones duplicadas. La interfaz se atenúa y evita reenvíos mientras se intercambian tokens. Si es primer acceso, se redirige automáticamente a Onboarding.

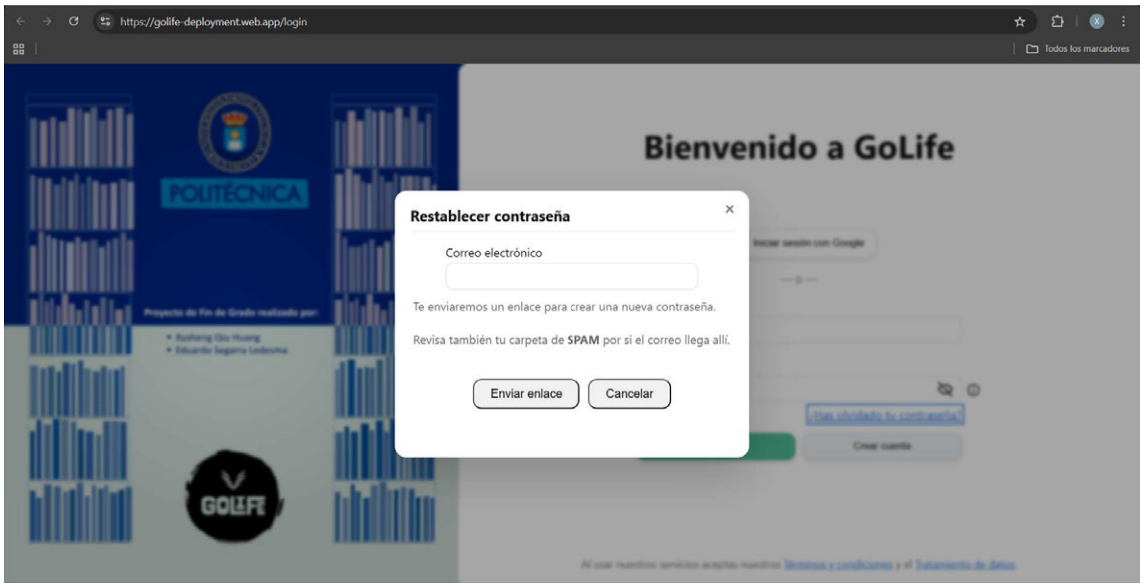


Figura 10.4: Restablecimiento por correo

El usuario introduce su email y confirma con *Enviar enlace*. Se indica revisar la carpeta de SPAM si no llega el mensaje. Al cerrar, se regresa al login manteniendo el estado anterior.

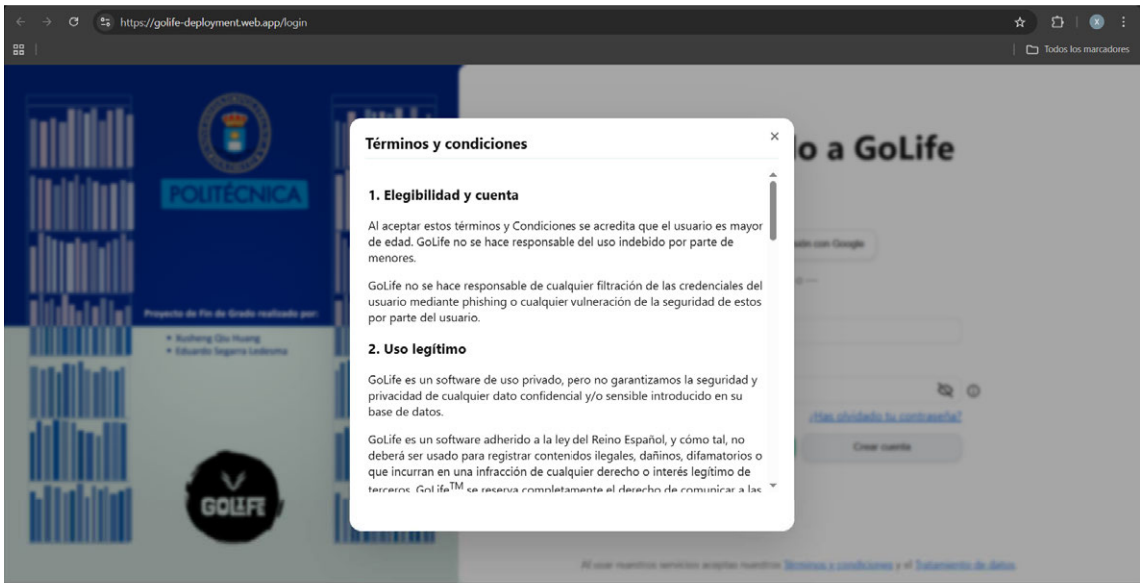


Figura 10.5: Términos y Condiciones en modal

Define elegibilidad, uso legítimo y responsabilidades del usuario. El cierre no modifica el estado de la sesión ni del formulario. Accesibles desde el pie del login para consulta previa al registro.

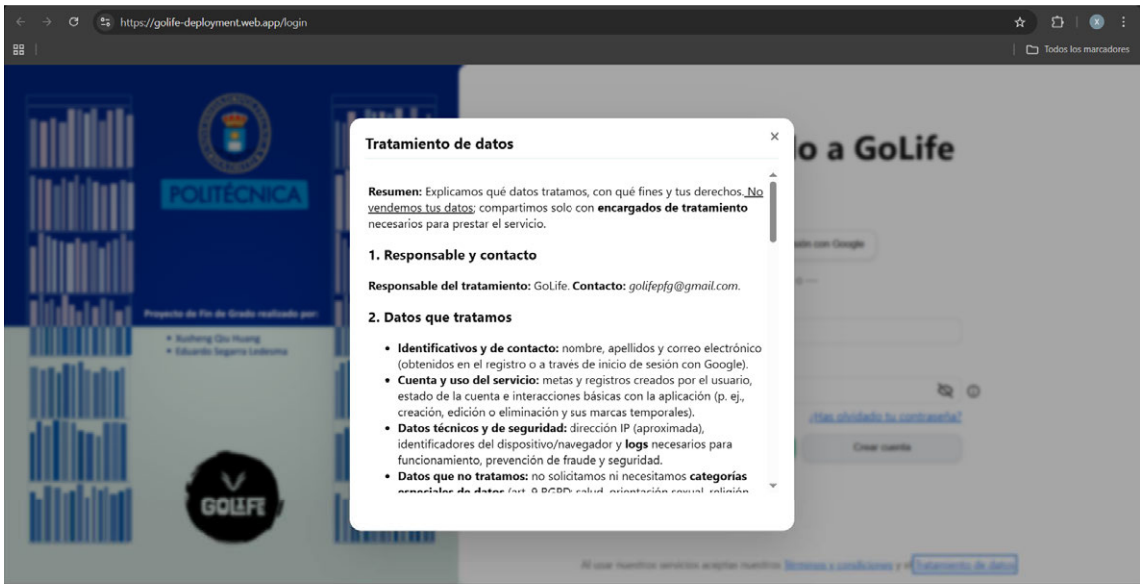


Figura 10.6: Política de Tratamiento de datos

Detalla categorías tratadas/no tratadas y base legal del servicio. Transparencia previa a crear cuenta o iniciar sesión con Google. Presentación consistente con el resto de modales legales de la app.

10.1.2. Onboarding

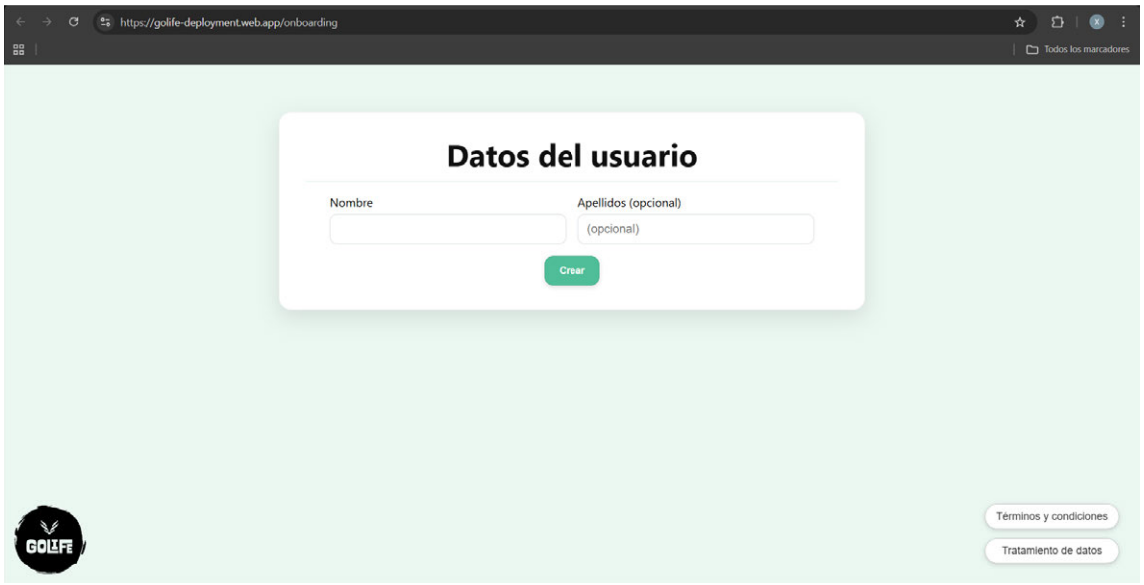


Figura 10.7: Pantalla de Onboarding con formulario de datos del usuario

El CTA *Crear* queda centrado bajo la línea divisoria del formulario. Accesos a *Términos y condiciones* y *Tratamiento de datos* en la esquina inferior derecha. Fondo suave para foco visual y continuidad con el diseño del Login.

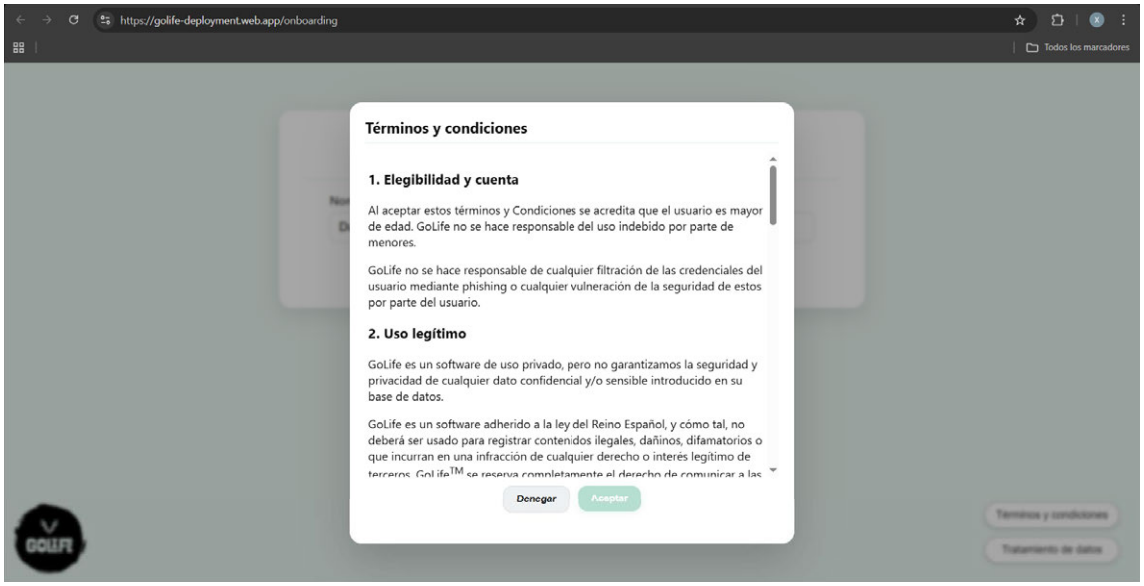


Figura 10.8: Primer estado del formulario 'Términos y condiciones'

Validaciones básicas de entrada y campos de texto con *placeholders* claros. El botón *Crear* se habilita sólo cuando el nombre es válido. Interfaz minimalista para reducir fricción en el primer alta.

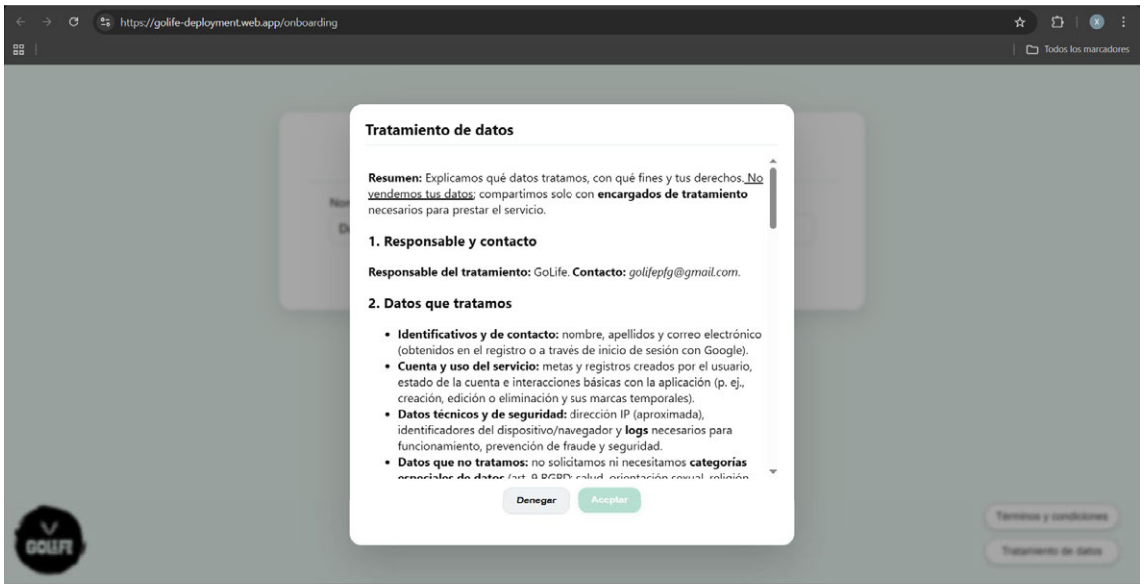


Figura 10.9: Segundo estado del formulario 'Tratamiento de datos'

Mantiene consistencia con la pantalla base: campos grandes y bordes suaves. El botón *Crear* aparece habilitado al cumplirse los mínimos. Tras confirmar, se crea el perfil y se redirige al Dashboard.

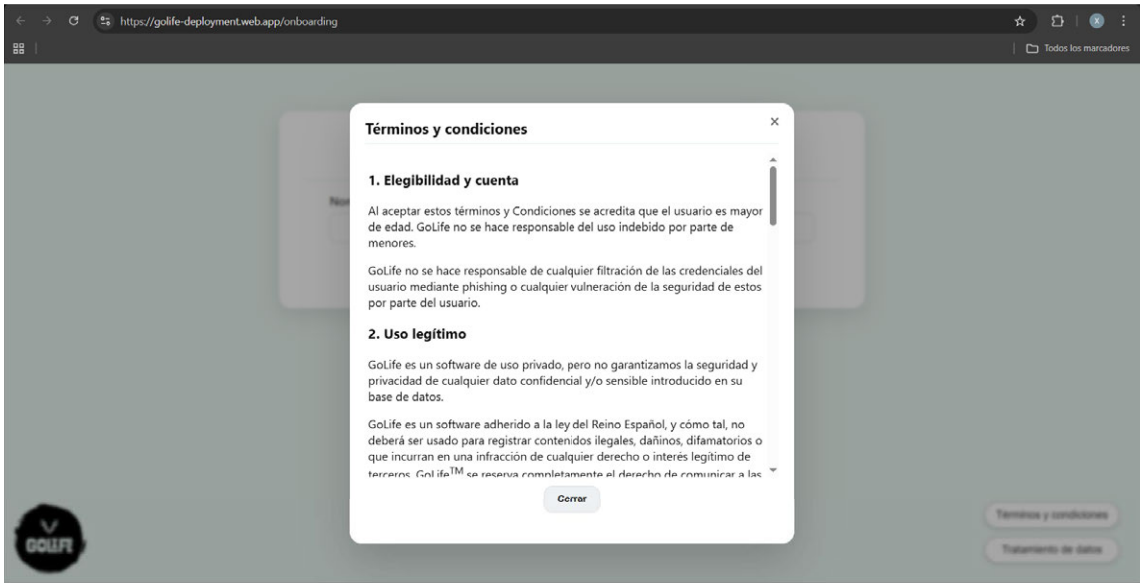


Figura 10.10: Modal de *Términos y condiciones*

Incluye secciones de elegibilidad, uso legítimo y responsabilidades del usuario. Botones *Denegar/Aceptar* al pie, con enfoque en consentimiento informado. El cierre del modal no altera el estado del formulario.

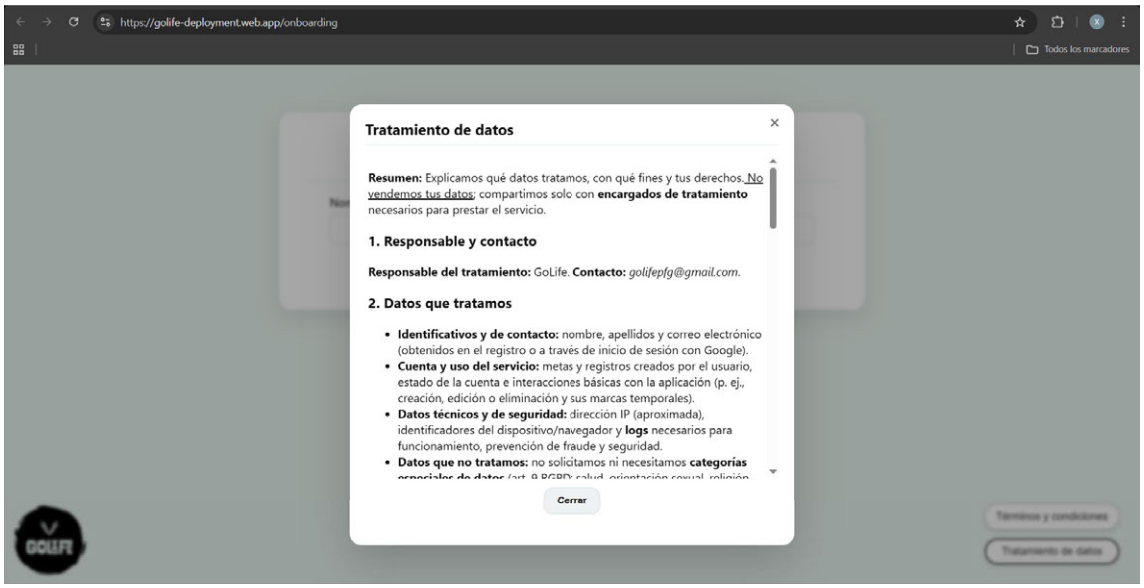


Figura 10.11: Modal de *Tratamiento de datos*

Describe finalidades, categorías tratadas/no tratadas y base legal. Redacción orientada a transparencia antes de crear la cuenta. Diseño unificado con los modales legales de Login y Profile.

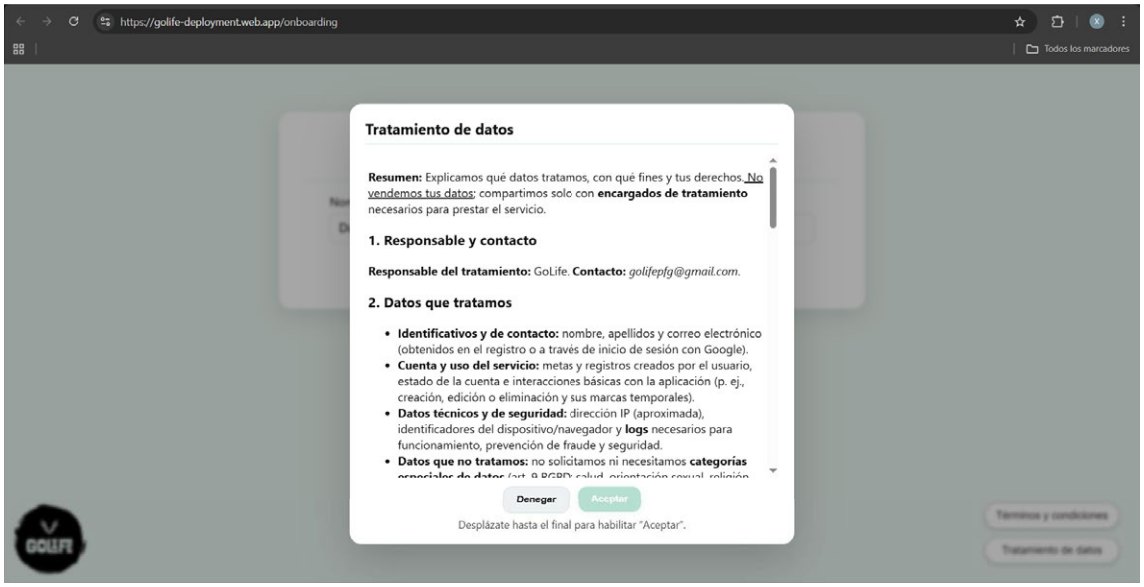


Figura 10.12: Indicador de *scroll obligatorio*

"Desplázate hasta el final para habilitar 'Aceptar'" guía la interacción. El botón *Aceptar* permanece deshabilitado hasta llegar al final del texto. Previene consentimientos sin lectura y mejora el cumplimiento.

10.1.3. Dashboard

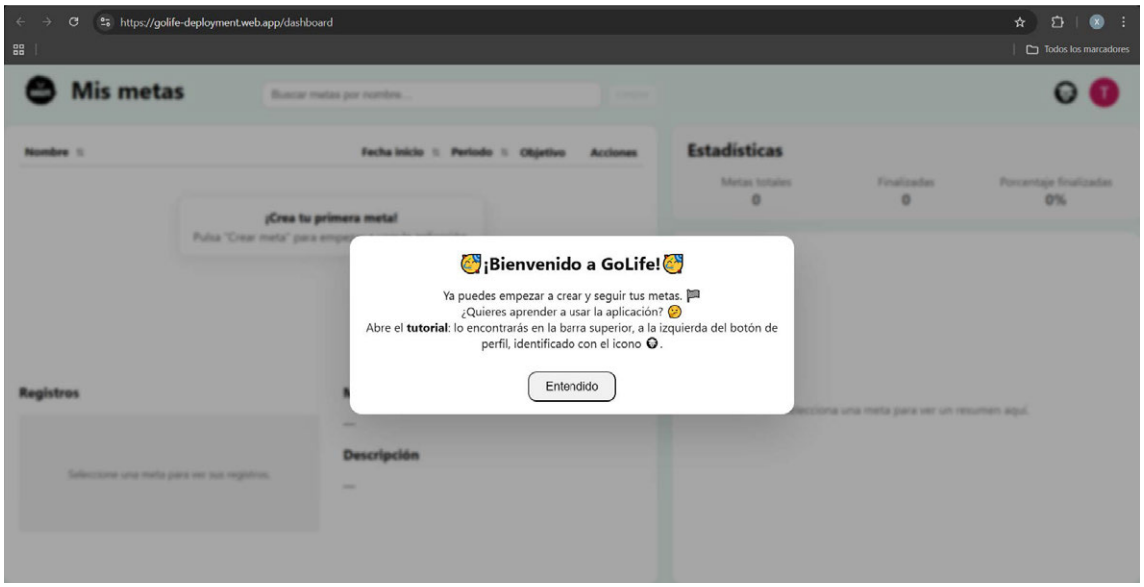


Figura 10.13: Mensaje de bienvenida al entrar primera vez

Indica que aquí se gestionan las metas y cómo avanzar por el recorrido. Controles *Anterior/Siguiente* para aprender a ritmo del usuario. Fondo oscurecido reduce distracción y dirige la atención al foco.

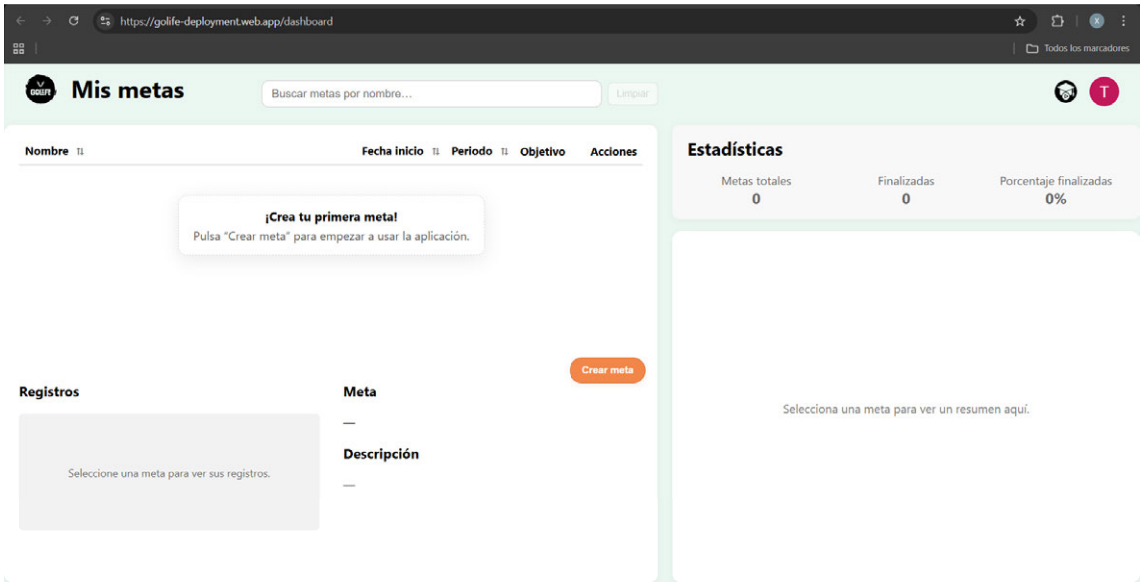


Figura 10.14: Estado vacío del tablero cuando no hay metas existentes

Tarjeta central invita a "Crear meta" para iniciar el uso de la aplicación. Estadísticas y gráficos permanecen limpios hasta que haya datos reales. El CTA lateral *Crear meta* se mantiene visible para acceso rápido.

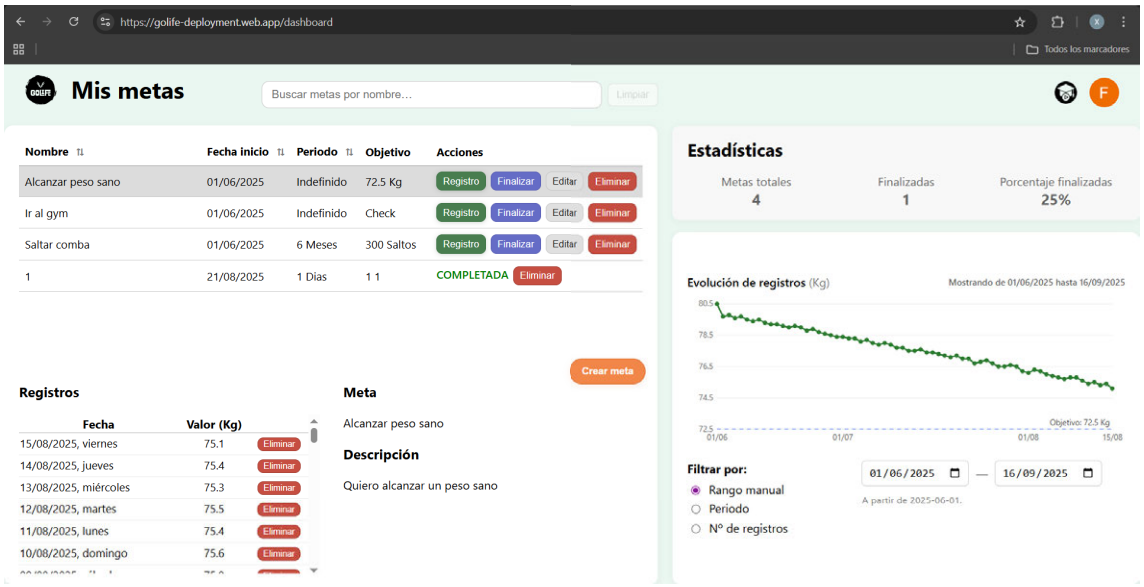


Figura 10.15: Vista de trabajo con metas numéricas

Tabla izquierda y detalle con registros abajo. Panel derecho con estadísticas y gráfico de evolución con filtros de rango/periodo. Acciones por fila: *Registro*, *Finalizar*, *Editar* y *Eliminar* con colores semánticos. Al seleccionar una meta se despliegan su descripción y su histórico asociado.

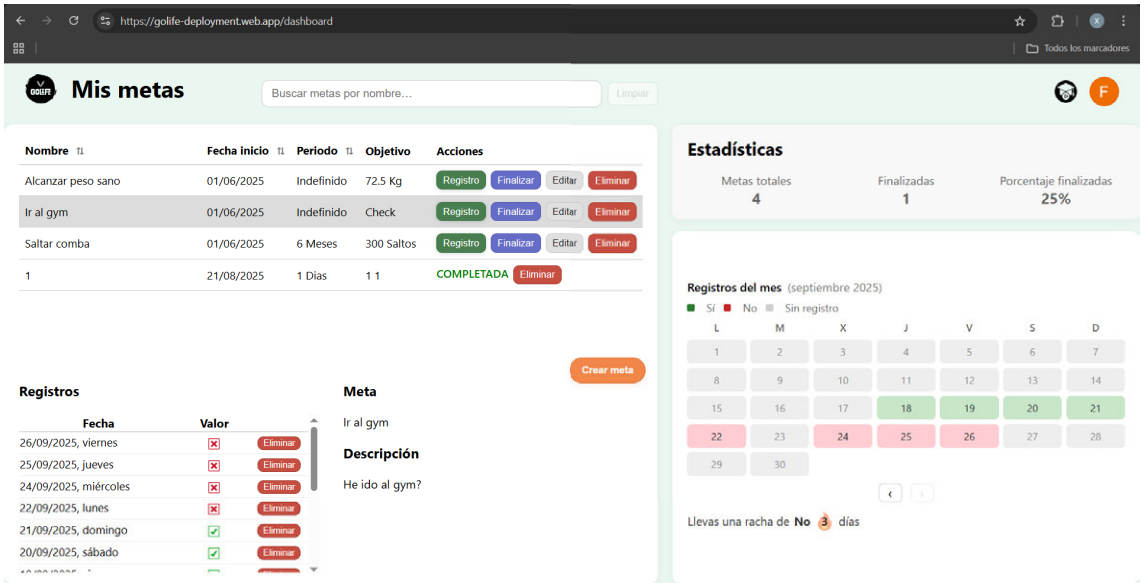


Figura 10.16: Vista de trabajo con metas booleanas

Colores verdes/rojos/grises codifican cumplimiento y omisiones de forma inmediata. Se muestra la racha actual como refuerzo motivacional y seguimiento. Las acciones operativas mantienen coherencia con el resto del tablero.

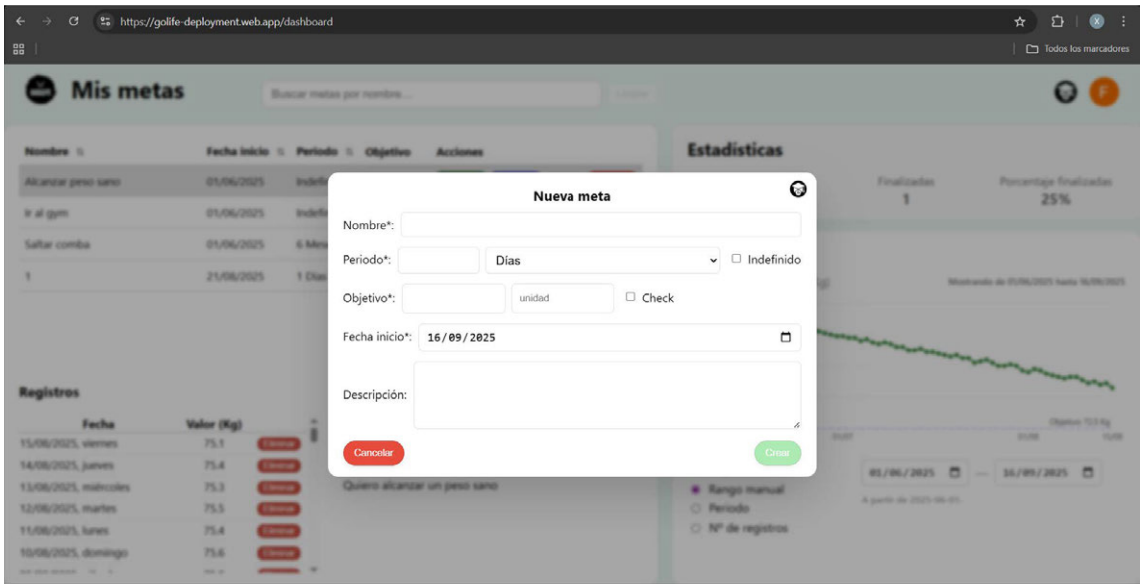


Figura 10.17: Formulario de nueva meta

El periodo admite número + unidad (días, semanas, meses, años) o casilla *Indefinido*. El objetivo puede ser numérico con unidad o de tipo *check* para hábitos binarios. La fecha inicia con el día actual y el icono abre un selector de calendario; incluye descripción.

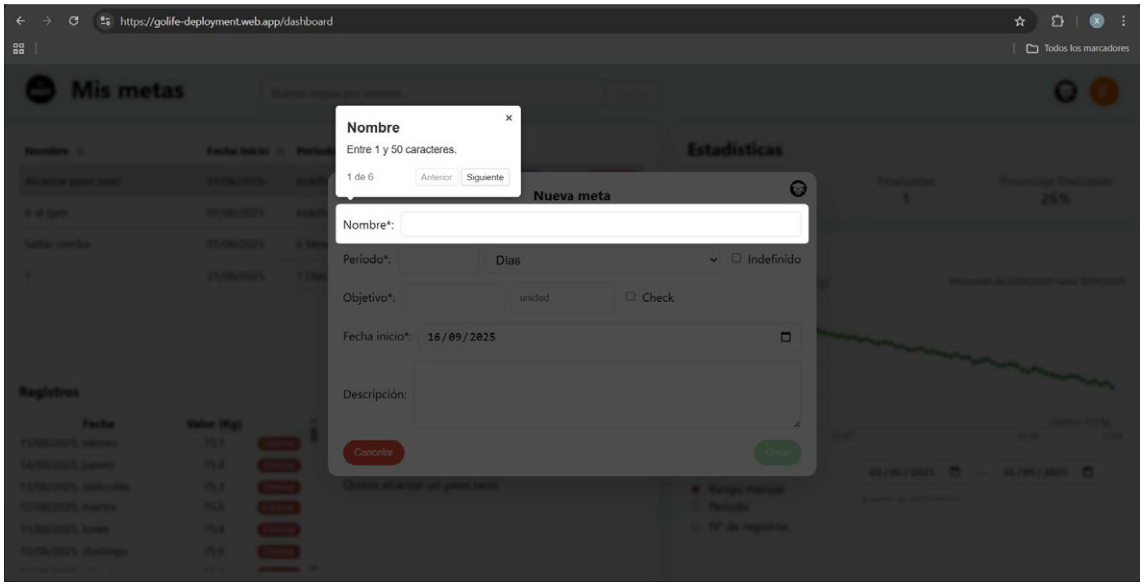


Figura 10.18: Paso del tutorial de Crear meta

Navegación *Anterior/Siguiente* y cierre accesible para no bloquear el flujo. Fondo atenuado para foco visual y prevención de clics accidentales fuera del paso. Orientado a la primera toma de contacto, reduce errores y curva de aprendizaje.

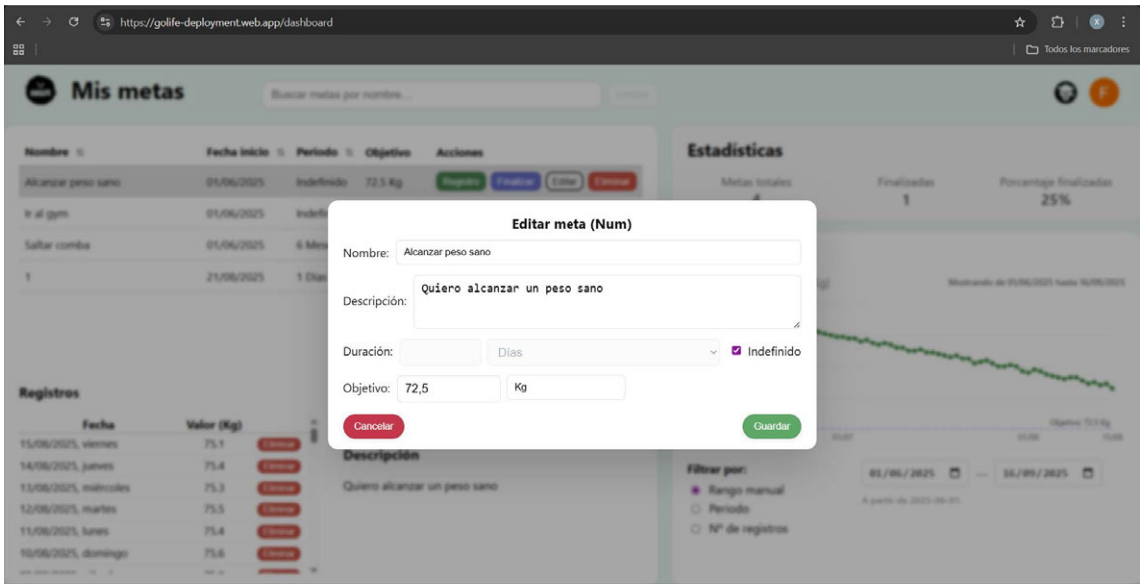


Figura 10.19: Edición de meta

Permite cambiar objetivo y duración (selector de unidad) o marcar *Indefinido*. Mantiene el histórico de registros sin pérdida de información. Botones *Guardar/Cancelar* con estados deshabilitados mientras valida.

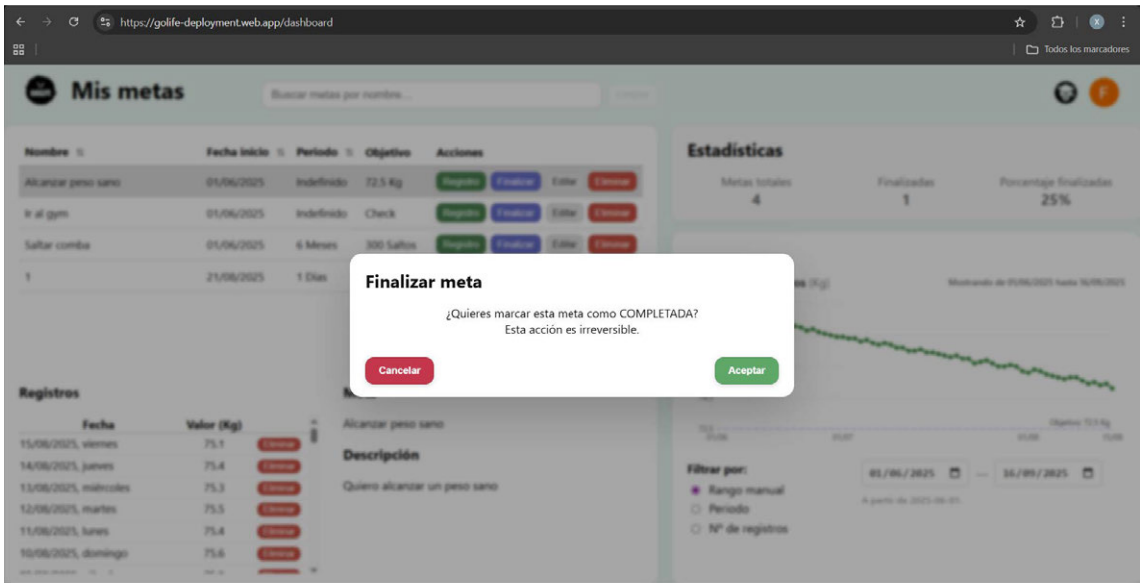


Figura 10.20: Confirmación de finalización con aviso de irreversibilidad

Al aceptar, la meta se marca como *COMPLETADA* y no admite nuevos registros. La fila correspondiente en la tabla pasa a fondo gris para diferenciarla. Las estadísticas se recalculan y reflejan el porcentaje de metas cerradas.

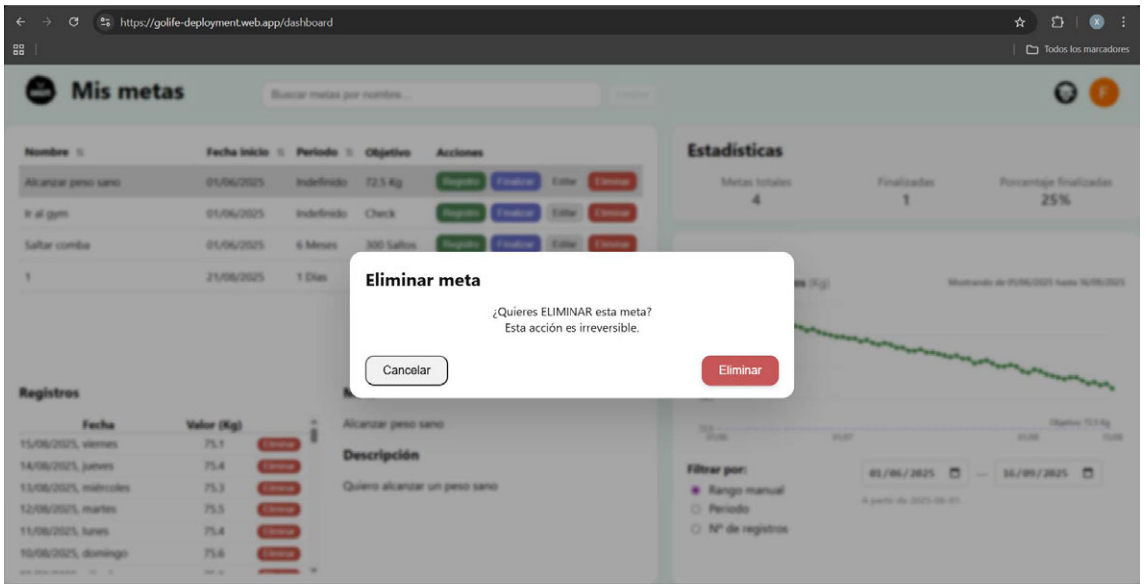


Figura 10.21: Diálogo de eliminación

Incluye alternativa *Cancelar* y botón de confirmación claramente diferenciado. Al confirmar, se borra la meta junto con todos sus registros asociados. Si el backend rechaza la operación, la interfaz no se altera y se comunica el error.

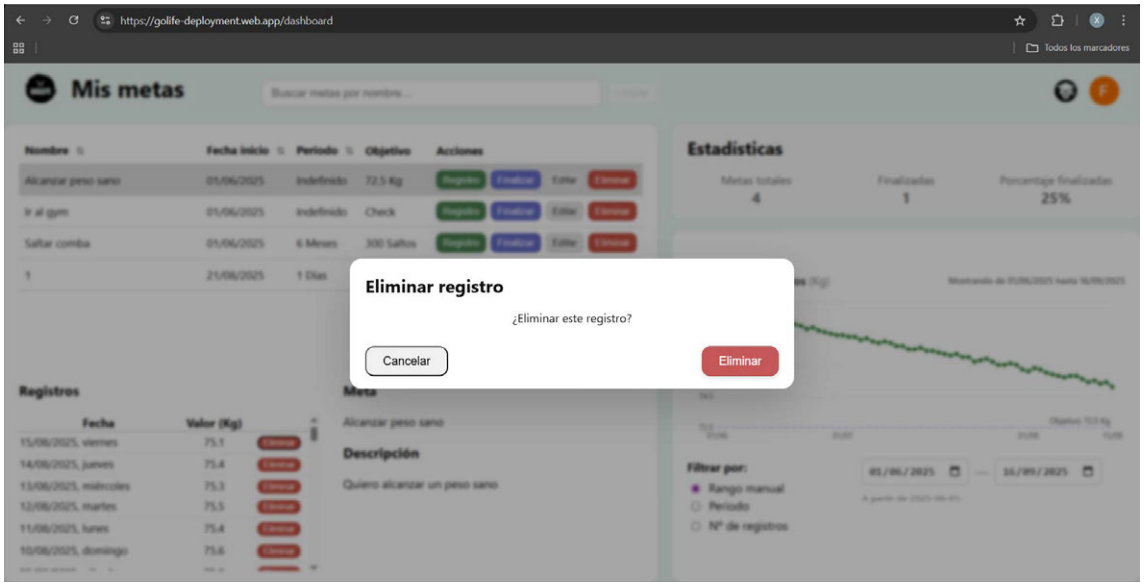


Figura 10.22: Confirmación para eliminar un registro de meta

Previene borrados accidentales en operaciones repetitivas de limpieza. La lista de registros se actualiza al instante y el gráfico refleja el cambio. Cerrar sin confirmar mantiene intactos los datos existentes.

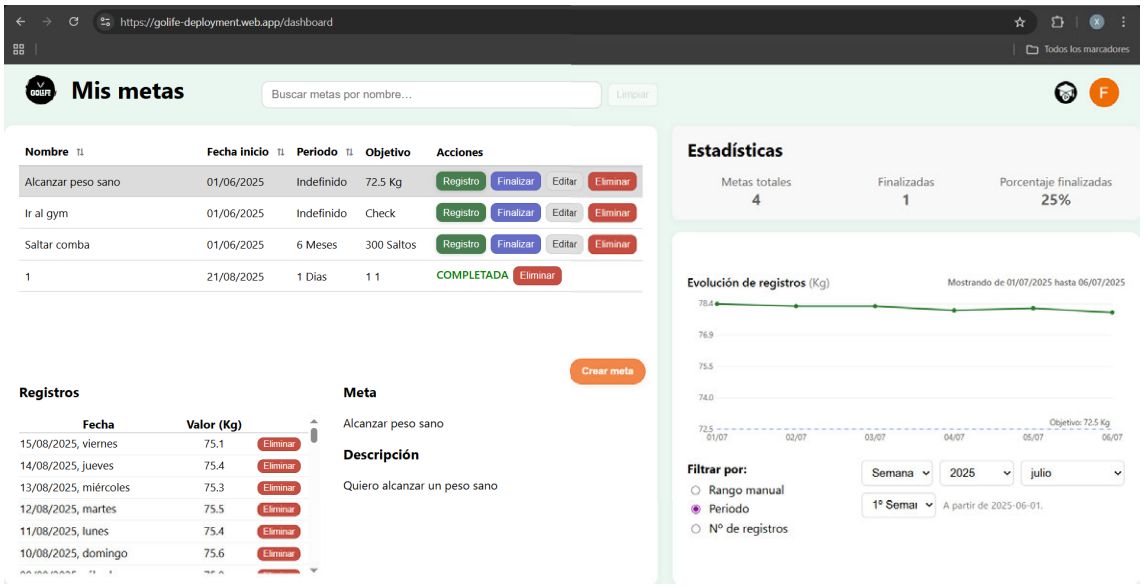
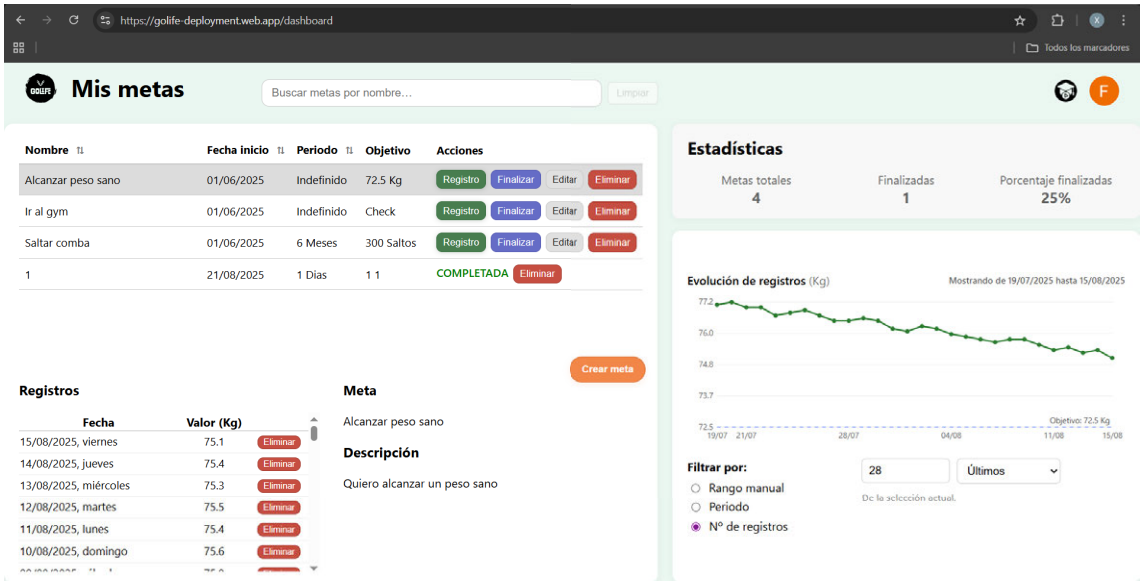


Figura 10.23: Gráfico con filtro por Periodo

Controles para semana/mes/año y selección de año/mes específico. Línea de referencia del objetivo visible para comparar progreso. Ejes y etiquetas se adaptan automáticamente al filtro aplicado.



áfico con filtro por últimos/primero N registros

Selector de cantidad (p. ej., “28 últimos”) para análisis reciente y ágil. Útil para evaluar tendencias cortas tras cambios de rutina o ajustes. Se mantiene la referencia al objetivo y una leyenda mínima.

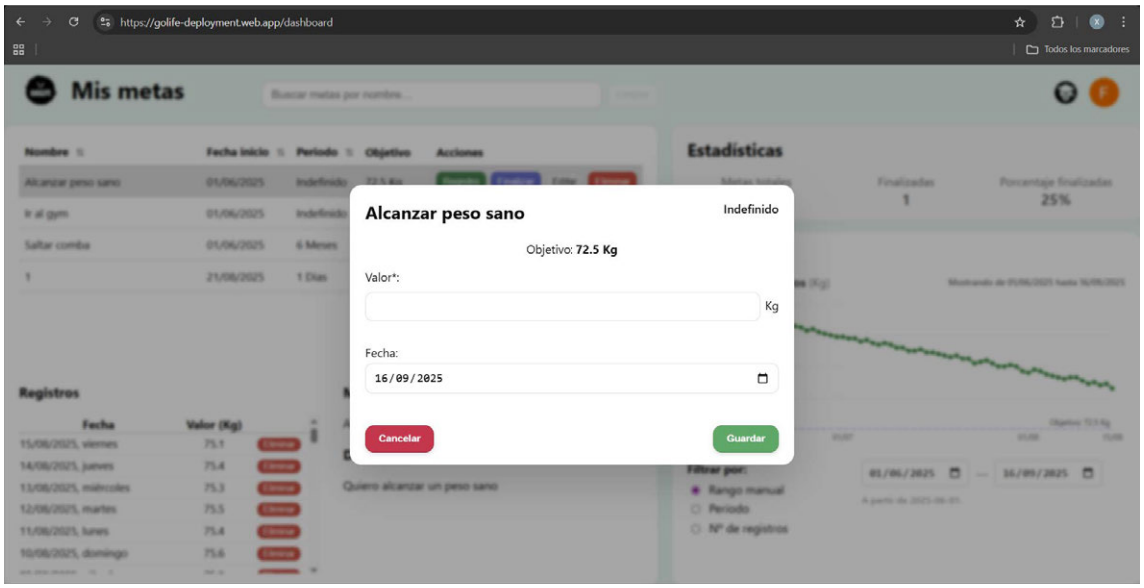


Figura 10.24: Crear registro

Campo numérico con unidad y selector de fecha editable mediante calendario. El objetivo aparece en cabecera como referencia durante la introducción del dato. Acciones *Guardar/Cancelar* con validación y cierre automático al completar.

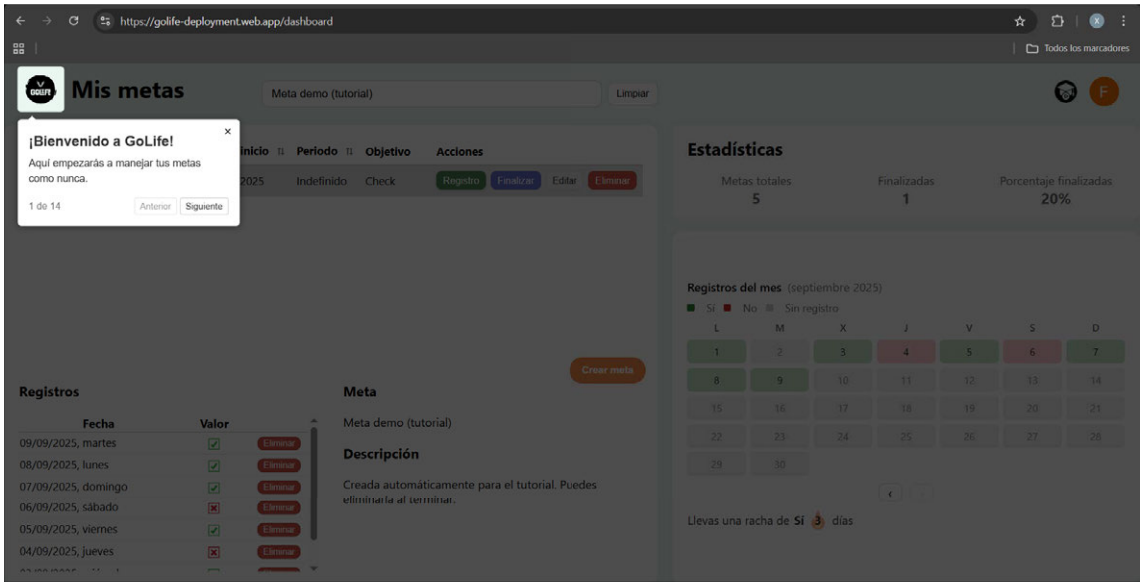


Figura 10.25: Main tutorial de dashboard

Explica dónde encontrar el icono del recorrido y su propósito formativo. Botón principal *Entendido* para continuar sin obligación de seguir el tour. Se muestra a cuentas nuevas o cuando aún no existen metas creadas.

10.1.4. Profile

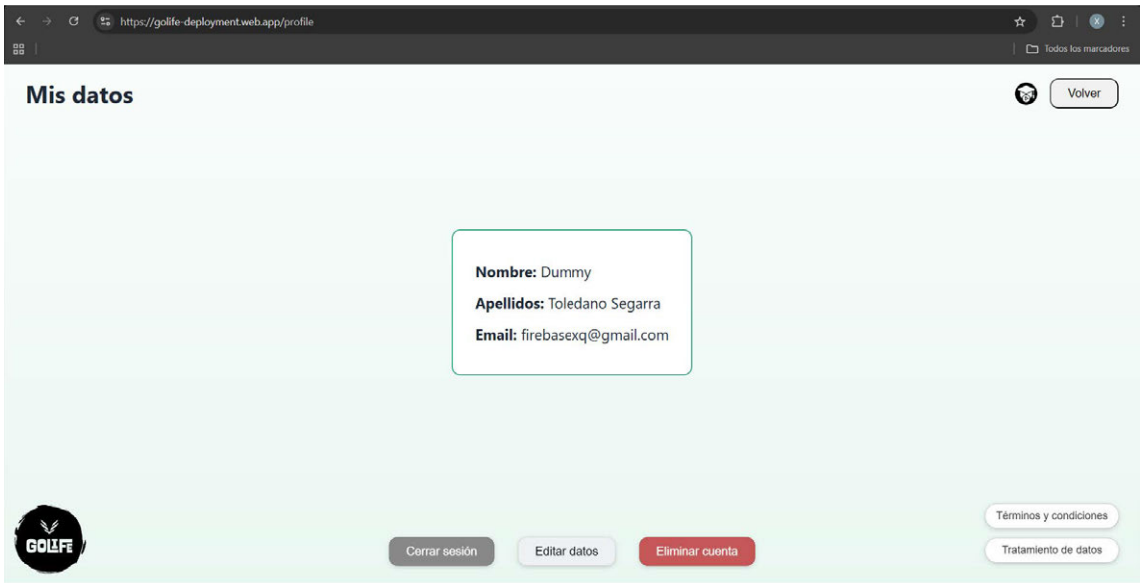


Figura 10.26: Profile vista

Botones disponibles: *Cerrar sesión*, *Editar datos* y *Eliminar cuenta*. En la esquina inferior derecha se accede a *Términos* y *Tratamiento de datos*. Diseño consistente con el resto de la aplicación y jerarquía visual clara.

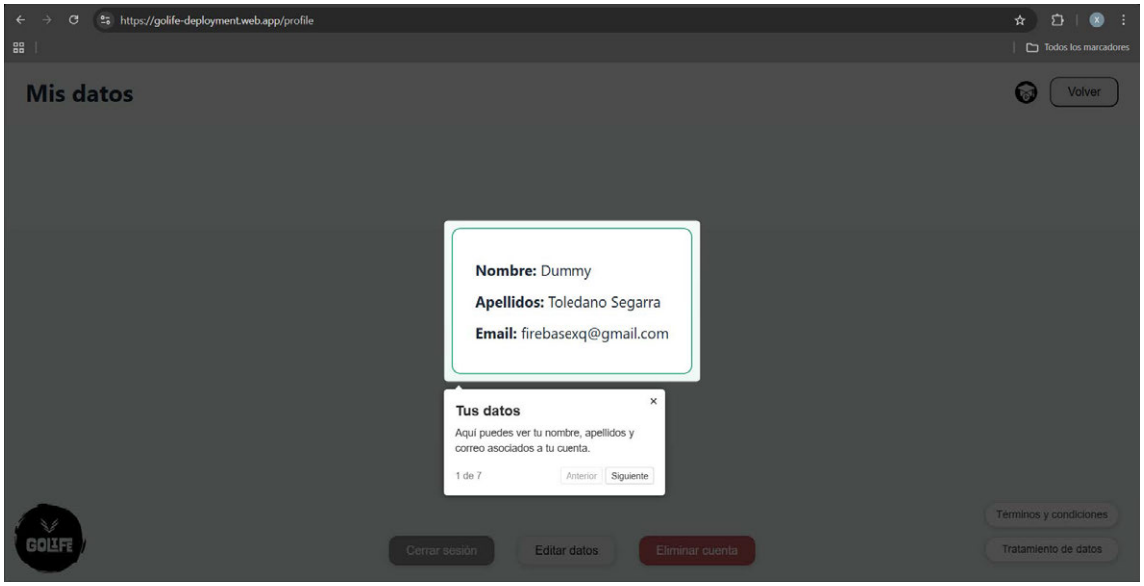


Figura 10.27: Tutorial en Profile

Explica qué información se muestra (nombre, apellidos y email) y cómo navegar. Controles *Anterior/Siguiente* permiten avanzar sin perder el contexto. El fondo oscurecido centra la atención y evita clics accidentales fuera del paso.

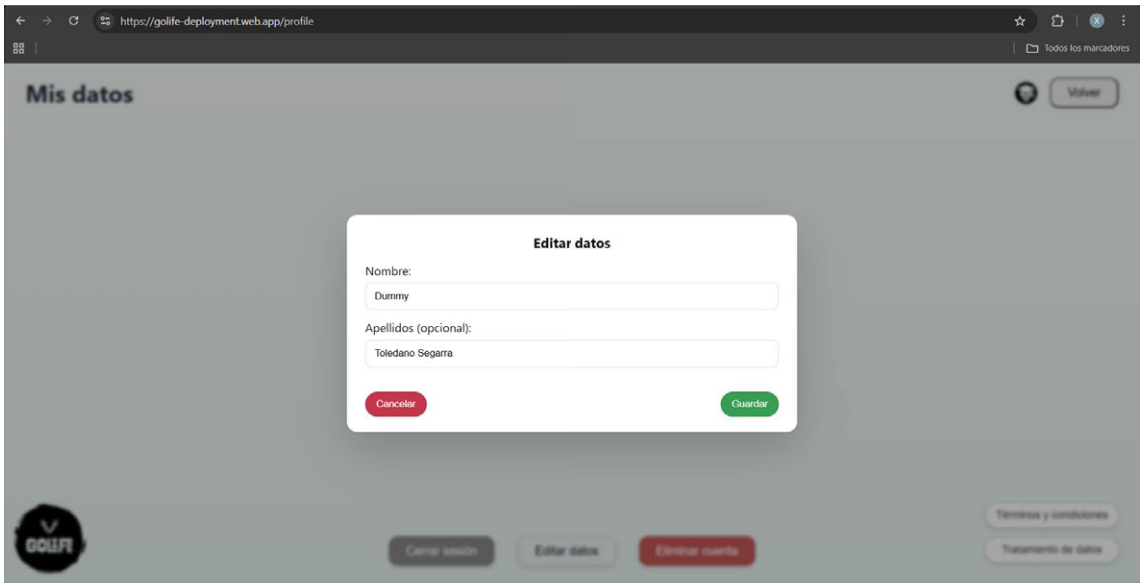


Figura 10.28: Edición de datos personales

Se permite actualizar nombre y apellidos (opcional) manteniendo el email. Acciones *Guardar/Cancelar* con validación previa y cierre elegante. El fondo atenuado preserva el contexto y previene interacciones fuera del diálogo.

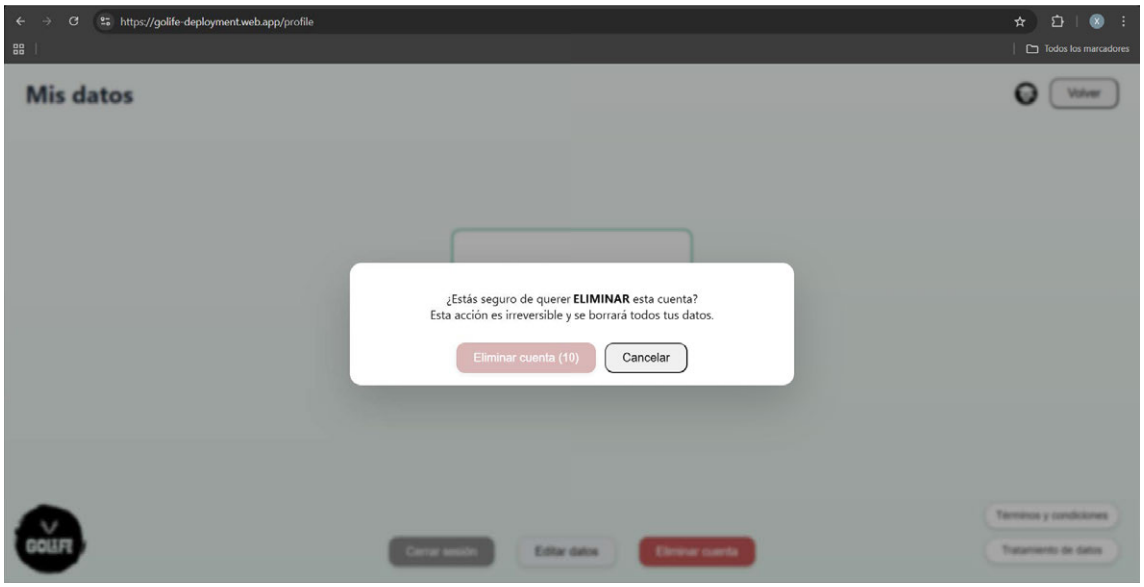


Figura 10.29: Confirmación para eliminar la cuenta

Botón de confirmación con cuenta atrás para evitar pulsaciones impulsivas. La operación elimina credenciales y datos asociados de forma coordinada. Si algo falla, se cancela y se informa sin modificar el estado de la cuenta.

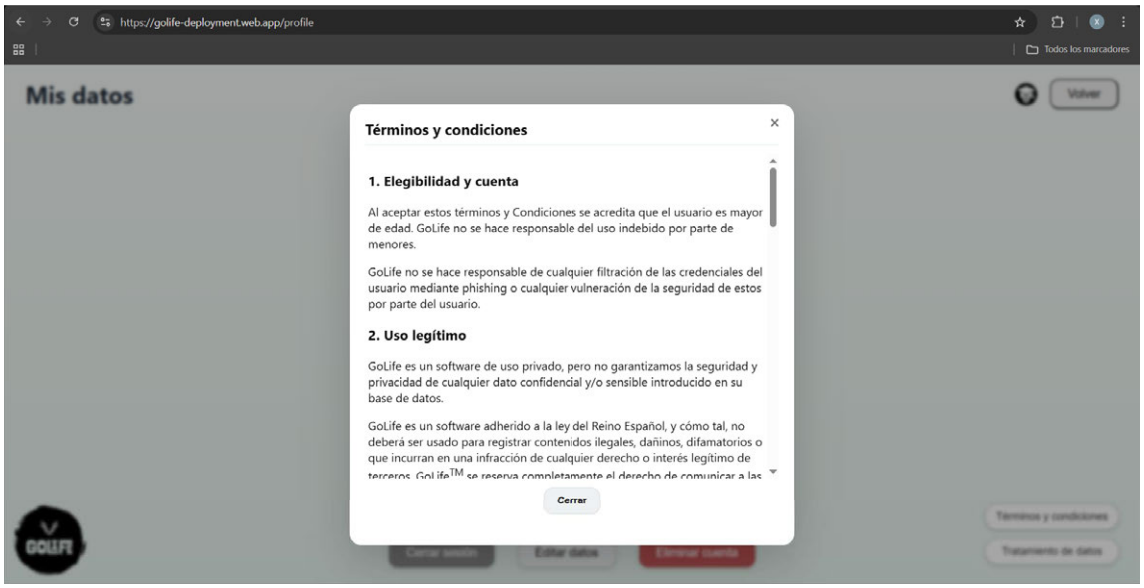


Figura 10.30: Modal de *Términos y condiciones*

Incluye apartados de elegibilidad, uso legítimo y responsabilidades del usuario. Permite consulta rápida desde Profile sin abandonar la vista principal. Cierre seguro: no altera sesión ni datos al salir del diálogo.

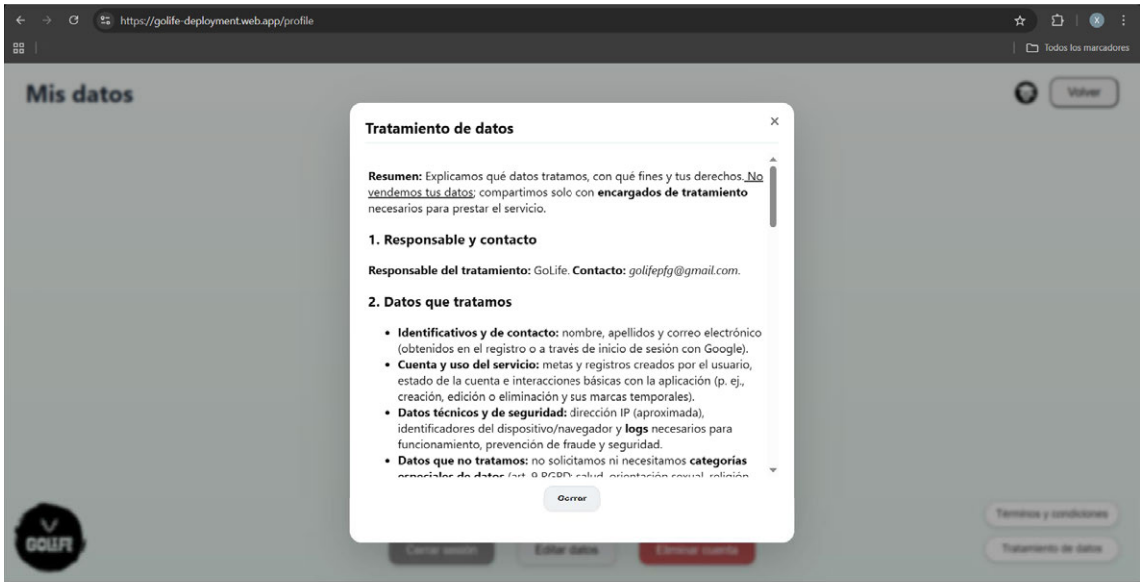


Figura 10.31: Modal de *Tratamiento de datos*

Expone categorías tratadas/no tratadas y la base legal aplicable. Misma estética que en Login/Onboarding para coherencia informativa. Pensado para consulta transparente en cualquier momento.

10.1.5. Vídeos demostrativos

Se han publicado cuatro vídeos breves con el funcionamiento general de la aplicación. **Los siguientes enlaces son clicables** (haz clic para abrirlos en YouTube):

GoLife - Crear nueva cuenta

GoLife - Recuperar contraseña

GoLife - Profile

GoLife - Dashboard

10.2. Ámbito Legal

Tratamos **GoLife** como un producto y, en consecuencia, definimos **Términos y Condiciones de uso** y una **Política de Tratamiento de Datos** alineadas con el servicio y nuestras capacidades técnicas (incluidas dependencias como Firebase Authentication, Cloud KMS y MongoDB Atlas), todo ello en el contexto de un **Proyecto Fin de Grado**. Estos son visibles desde la interfaz de GoLife y es requerido aceptarlos para crear la cuenta en nuestro sistema y poder acceder a la aplicación web la primera vez.

En esta sección se resumen los compromisos y límites del servicio, el tratamiento de los datos personales, y los principios de seguridad y responsabilidad aplicados.

10.2.1. Términos y Condiciones de uso

1. **Elegibilidad y cuenta**
   Al aceptar estos Términos y Condiciones se acredita que el usuario es mayor de edad. GoLife no se hace responsable del uso indebido por parte de menores.

   GoLife no se hace responsable de cualquier filtración de las credenciales del usuario mediante *phishing* o cualquier vulneración de la seguridad de estas por parte del propio usuario.

2. **Uso legítimo**
   GoLife es un software de uso privado, pero no garantizamos la seguridad y privacidad de cualquier dato confidencial y/o sensible introducido en su base de datos.

   GoLife es un software adherido a la ley del Reino de España y, como tal, no deberá ser usado para registrar contenidos ilegales, dañinos, difamatorios o que incurran en una infracción de cualquier derecho o interés legítimo de terceros. GoLife^{TM} se reserva completamente el derecho de comunicar a las autoridades pertinentes cualquier contenido o registro que lesione las leyes nacionales, europeas o internacionales, de conformidad con la legislación vigente.

   El código fuente de GoLife^{TM} es de propiedad única y exclusiva de los desarrolladores (D. Eduardo Segarra Ledesma y D. Xusheng Qiu Huang), y los susodichos se reservan el derecho de tomar las acciones legales pertinentes contra cualquier intento de infringir los derechos de autor de estos, entre las cuales se podría encontrar, pero sin estar limitado a: ingeniería inversa, accesos no autorizados o *phishing*.

3. **Contenido del usuario**
   El usuario conservará su debido derecho sobre todo aquel contenido que, acorde con las leyes vigentes (véase: Ley Orgánica de Protección de Datos), introduzca en las bases de datos de GoLife^{TM}. La empresa se reserva los derechos para el tratamiento de estos datos, de cara a la optimización del software de una manera categóricamente interna.

   GoLife no compartirá los datos a los que acceda con terceros (véase: Política de Tratamiento de Cookies).

   Concedes a GoLife una licencia no exclusiva, mundial e irrevocable para alojar, procesar y mostrar tu contenido con el único fin de prestar el servicio.

4. **Cambios y disponibilidad**
   GoLife es un software en constante desarrollo; como tal, se introducirán periódicamente cambios y/o mejoras en el presente servicio. Se comunicará por vía email al usuario los cambios realizados en un plazo no inferior a **DOS (2)** días laborables. A la hora de introducir estos cambios, el servicio podrá no estar disponible por un periodo no superior a **DIEZ (10)** días laborables.

5. **Incumplimientos y cierre de cuenta**
   GoLife se reserva el derecho, de manera exclusiva e imprescriptible, a suspender o cerrar la cuenta del usuario si observa actividades sospechosas, eliminando la información contenida en la base de datos cuando resulte pertinente.

   El usuario puede en cualquier momento cerrar su cuenta de una manera completamente unilateral.

6. **Propiedad intelectual**
   GoLife y sus elementos (marcas, código fuente, diseño, propiedad intelectual e industrial, y derechos de explotación en exclusiva) son propiedad exclusiva de D. Eduardo Segarra Ledesma y D. Xusheng Qiu Huang, y la utilización del software no confiere ningún tipo de licencia sobre estas propiedades, de conformidad con los artículos 1, 3, 5, 17 y 18 ss del Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia.

7. **Exclusión de garantías**
   El software de GoLife^{TM}, si bien está enfocado de una manera general, no garantiza la adecuación a propósitos específicos que pueda tener el usuario.

8. **Limitación de responsabilidad**
   GoLife no se hace responsable de ninguna manera de cualquier pérdida pecuniaria, moral o relativa a la integridad física por el uso de este software. GoLife recomienda el uso responsable de las herramientas facilitadas.

9. **Modificaciones de los Términos**
   GoLife se reserva el derecho de la modificación de los Términos y Condiciones de la plataforma web si se ve motivada para hacerlo de manera tanto externa como interna. GoLife notificará con una antelación de **5 días laborables** al usuario antes de llevar a

cabo el pretendido cambio.

Para demostrar la aceptación del usuario de la modificación de estos Términos y Condiciones, se impedirá el uso o *log-in* de este hasta la debida aceptación de los mismos.

10. **Contacto**
Soporte y consultas legales: `golifepfg@gmail.com`

10.2.2. Tratamiento de datos

Resumen: explicamos qué datos tratamos, con qué fines y tus derechos. No vendemos tus datos; compartimos solo con *encargados de tratamiento* necesarios para prestar el servicio.

1. **Responsable y contacto**
Responsable del tratamiento: **GoLife**.
Contacto: `golifepfg@gmail.com`.

2. **Datos que tratamos**

   - *Identificativos y de contacto:* nombre, apellidos y correo electrónico (obtenidos en el registro o a través de inicio de sesión con Google).
   - *Cuenta y uso del servicio:* metas y registros creados por el usuario, estado de la cuenta e interacciones básicas con la aplicación (p. ej., creación, edición o eliminación y sus marcas temporales).
   - *Datos técnicos y de seguridad:* dirección IP (aproximada), identificadores del dispositivo/navegador y *logs* necesarios para funcionamiento, prevención de fraude y seguridad.
   - *Datos que no tratamos:* no solicitamos ni necesitamos **categorías especiales** de datos (art. 9 RGPD: salud, orientación sexual, religión, etc.). Te pedimos que **no** incluyas este tipo de información en tus metas o registros. Si detectamos contenidos ilícitos o categorías especiales sin base legal, podremos suprimirlos y suspender o cancelar la cuenta, conforme a los Términos de Uso.

3. **Finalidades y bases jurídicas (RGPD)**

   - *Prestación del servicio* (art. 6.1.b): alta y autenticación (Firebase), gestión de la cuenta, metas y registros, y soporte.
   - *Seguridad y continuidad* (art. 6.1.f; y, cuando proceda, art. 6.1.c): prevención de fraude y abuso, control y diagnóstico mediante *logs*, copias de seguridad y recuperación ante fallos, así como mejora del servicio y estadísticas agregadas no identificadas.
   - *Cumplimiento normativo* (art. 6.1.c): atención de requerimientos de autoridades y demás obligaciones legales aplicables.
   - *Comunicaciones operativas* (arts. 6.1.b y 6.1.f): avisos sobre el servicio, seguridad o cambios relevantes. Las comunicaciones no esenciales solo se envían con **consentimiento** (art. 6.1.a), que puede retirarse en cualquier momento.

4. **Conservación**
Guardamos tus datos mientras tengas cuenta activa y, tras la baja, durante los plazos necesarios para cumplir obligaciones legales o resolver reclamaciones. Al borrar tu cuenta desde *Perfil*, eliminamos los datos operativos de la aplicación y de nuestra base de datos; los *logs* y copias de seguridad se depuran conforme a ciclos técnicos razonables.

5. **Destinatarios y transferencias**
   - *Encargados de tratamiento:* proveedores que actúan como encargados (art. 28 RGPD), principalmente de infraestructura y autenticación (p. ej., Google Cloud / Firebase). Tratan los datos solo conforme a nuestras instrucciones y bajo contratos con obligaciones de confidencialidad, seguridad y limitación de finalidad.
   - *Transferencias internacionales:* GoLife opera principalmente en la UE. No obstante, **Firebase Authentication** se presta desde EE. UU., lo que puede implicar transferencias internacionales. Google LLC afirma cumplir con RGPD aportando garantías adecuadas (p. ej., certificación en el EU–U.S. Data Privacy Framework y, cuando proceda, Cláusulas Contractuales Tipo (Decisión de Ejecución (UE) 2021/914), junto con medidas técnicas y organizativas complementarias.
   - *Cesiones y comunicaciones:* no vendemos datos personales ni los cedemos para marketing de terceros. Solo comunicaremos datos cuando exista obligación legal o sea necesario para la prestación del servicio.
6. **Derecho**
   Puedes ejercer los derechos de **acceso**, **rectificación**, **supresión**, **oposición**, **limitación** y **portabilidad**. También puedes **retirar el consentimiento** cuando aplique y **reclamar** ante la AEPD. Para ejercerlos, escribe a `golifepfg@gmail.com` o usa los mecanismos dentro de la app (Perfil).
7. **Seguridad**
   GoLife aplicará las medidas necesarias, técnicas y requeridas por ley para asegurar la **confidencialidad**, **integridad** y **disponibilidad** de tus datos. Sin embargo, GoLife no se hace responsable de cualquier filtración que se dé por causas de un uso ilegítimo o indebido por parte del usuario.
8. **Cookies y analítica**
   Usamos cookies/tecnologías similares **estrictamente necesarias** para que la app funcione y, en su caso, **analítica agregada** para mejorar el servicio. Puedes gestionar cookies no esenciales desde los ajustes de tu navegador.
9. **Menores**
   GoLife no está dirigido a menores de **18 años**. Si detectamos una cuenta de menor, la eliminaremos inmediatamente.
10. **Cambios en esta política**
    Podemos actualizar esta política para reflejar cambios legales o del servicio. Te avisaremos de **cambios materiales** por medios razonables.

10.2.3. Cumplimiento RGPD (seudonimización)

Esta subsección se centra exclusivamente en la **seudonimización** aplicada en GoLife. Para el resto de aspectos de cumplimiento (bases jurídicas, derechos, conservación, cookies, encargados, transferencias), véanse las subsecciones anteriores.

Objetivo

Reducir el riesgo de identificación directa: el `uid` emitido por Firebase **no** se almacena en claro; se trabaja con un **seudónimo** derivado.

Ámbito de datos

La seudonimización se aplica **exclusivamente al** `uid` emitido por Firebase, por ser el único identificador técnico que podría **vincular directamente** la identidad de autenticación con los datos de uso (metas y registros). En la base de datos **no** guardamos el `uid` en claro, sino su **seudónimo** derivado mediante Cloud KMS.

- `uid` **(identificador técnico):** Seudonimizado en escritura/lectura; nunca se persiste en claro. Las autorizaciones se hacen comparando el valor seudonimizado con el `uid` verificado del token.
- **Metas y registros:** No deben contener información sensible ni identificativa de personas (véanse Términos y Condiciones). Describen objetivos/avances y no incluyen correos ni otros identificadores directos.
- **Nombre y apellidos:** Se tratan como *datos personales*, pero su aportación es **opcional**, en **libre texto**, y se usan como **alias de visualización**. No se verifican, no se indexan ni se emplean como clave de unión ni para decisiones automatizadas. El usuario puede **editarlos o eliminarlos** en cualquier momento. Dado que el correo no se persiste en la base de datos y el `uid` está seudonimizado, estos campos, por sí solos, presentan **bajo riesgo** de identificación en nuestro contexto. Aunque no se cifran campo a campo, aplicamos medidas *adecuadas y proporcionales* (art. 32 RGPD):
  - **TLS en tránsito** entre todos los componentes.
  - **Cifrado en reposo a nivel de plataforma** (proveedores cloud).
  - **Control de accesos (IAM)** y **mínimo privilegio** en los servicios.

En resumen, el **único punto de enlace técnico** entre identidad y datos operativos es el `uid`; por ello es el **único** campo sometido a seudonimización. El resto de atributos se minimizan, son opcionales o carecen de identificadores directos, y se protegen con las salvaguardas anteriores.

Flujo técnico resumido

La API deriva y verifica el seudónimo mediante **Cloud KMS** (operaciones `macSign`/`macVerify`). La **clave no sale** de KMS. Todas las comunicaciones entre componentes se realizan sobre **HTTPS/TLS**.

Gestión de claves

Clave MAC en KMS (ubicación UE), control de acceso por **IAM** con **mínimo privilegio**; únicamente la cuenta de servicio de la API puede firmar/verificar. Rotación y custodia gestionadas por el proveedor.

Persistencia y autorización

En base de datos sólo se guarda el **seudónimo** (resultado MAC). Las consultas y comprobaciones de pertenencia se hacen comparando ese valor seudonimizado contra el `uid` verificado del token.

Transferencias

La autenticación de usuarios se realiza con **Firebase Authentication** (operado en EE. UU.); la aplicación no persiste `uid` en claro en la BD. El cálculo/uso del seudónimo ocurre en infraestructura de la UE.

Proveedor y cumplimiento (Firebase)

Firebase declara **cumplimiento con RGPD**, actúa como *encargado del tratamiento* bajo acuerdos de tratamiento de datos, y documenta ubicaciones/procesos de datos y salvaguardas aplicables (p. ej., DPF/SCC cuando procede) (114). Esta conformidad se complementa en GoLife con seudonimización, control de accesos e **cifrado en tránsito**.

Limitaciones y coherencia

La seudonimización **no equivale** a anonimización: sigue siendo dato personal bajo RGPD. Se complementa con controles de acceso, registro de actividad y cifrado en tránsito. Esta medida es compatible con el ejercicio de derechos (acceso, supresión), al operar sobre el seudónimo.

10.3. Tests con usuarios reales

Como complemento a las pruebas unitarias, de integración y end-to-end, realizamos un **esbozo inicial de validación con personas usuarias**. El objetivo fue obtener una **primera impresión** de la experiencia general con GoLife: invitamos a participantes a usar libremente la aplicación (registro, creación/gestión de metas y consulta del panel) y, al finalizar, completaron un **cuestionario breve** sobre facilidad de uso, claridad de la interfaz, utilidad percibida y problemas encontrados. Se trata de una **sesión exploratoria**, no de una evaluación formal.

La captación se hizo mediante **redes sociales** para alcanzar diversidad básica de perfiles. Las respuestas se recogieron **de forma anónima**, informando del propósito del estudio y del uso **académico** de los resultados. Dado que este trabajo se enmarca en un **Proyecto Fin de Grado**, no se realizaron fases previas de investigación formal (p. ej., definición de público objetivo, muestreos o protocolos completos), por lo que los hallazgos deben interpretarse como **indicativos** y orientados a guiar mejoras iniciales del producto.

10.3.1. Cuestionario

El cuestionario se diseñó y distribuyó con **Google Forms**. Puede consultarse íntegro en el Anexo: Cuestionario de Evaluación de GoLife. Las preguntas se orientan a medir **usabilidad general** y a **validar criterios de aceptación** (aprendizaje, facilidad de uso, rendimiento percibido).

1. **¿Cuál es tu rango de edad?** *(obligatoria)*
   - 18–24 años
   - 25–34 años
   - 35–49 años
   - 50–64 años
   - 65 o más años
2. **¿Pudiste usar las funciones principales de GoLife (crear, hacer registros, editar, actualizar y eliminar metas) sin ayuda?** *(obligatoria)*
   - Sí, todas sin ayuda

- Sí, la mayoría, pero con alguna dificultad
- No, necesité ayuda

3. **En general, manejar las funciones principales de GoLife me resultó...** *(obligatoria)*

   Escala Likert 1–5: *Muy Difícil (1) ... Muy Fácil (5)*

4. **GoLife me resultó intuitivo desde un inicio** *(obligatoria)*

   Escala Likert 1–5: *Muy en Desacuerdo (1) ... Muy de Acuerdo (5)*

5. **¿Viste el tutorial de GoLife?** *(obligatoria)*

   - Sí, lo vi completo antes de crear metas
   - Sí, pero lo vi después de crear metas
   - No, no lo vi en ningún momento

6. **Si viste el tutorial, ¿por qué lo hiciste?**

   - Quería tener una guía antes de empezar a crear metas
   - Lo vi para aclarar dudas después de empezar a crear metas
   - No entendía la aplicación web en un inicio
   - No encontré el tutorial
   - Otra... *(abierta)*

7. **¿Crees que GoLife puede servir para distintos tipos de metas (por ejemplo: estudios, deporte, finanzas, hábitos, etc.)?** *(obligatoria)*

   - Sí, sin duda
   - En parte, pero con limitaciones
   - No, creo que está muy enfocada a un tipo concreto de metas

8. **¿Para qué tipo de metas usaste GoLife?**

   Respuesta abierta

9. **¿Cómo valorarías la velocidad de carga de la aplicación web en general?** *(obligatoria)*

   Escala Likert 1–5: *Muy Lenta (1) ... Muy Rápida (5)*

10. **¿Has experimentado tiempos de espera largos o bloqueos al usar la aplicación?** *(obligatoria)*

    - No
    - Sí, pero muy pocas veces
    - Sí, con frecuencia

10.3.2. Resultados

En esta subsección **revisamos los gráficos de cada pregunta del cuestionario** y **interpretamos las respuestas** para identificar patrones y la **recepción general** de GoLife. Con ello queremos señalar **aciertos**, posibles **fricciones** y **oportunidades de mejora** que orienten versiones futuras del producto. Los gráficos fueron generados por **Google Forms** a partir de **38 personas** que usaron la aplicación y completaron el cuestionario .

P1. Rango de edad (perfil de usuario) (Figura 10.32)

- **Descripción:** Predomina *18–24* (**60,5 %**, 23; estudiante o recién graduado). Le siguen *25–34* (**23,7 %**, 9; joven adulto, laboral temprana), *35–49* (**10,5 %**, 4; adulto en desarrollo personal/familiar) y *50–64* (**5,3 %**, 2; experiencia tecnológica intermedia). *65 o más*: 0 %.
- **Lectura:** Muestra concentrada en perfiles **jóvenes y familiarizados con la tecnología**; la recepción del resto de preguntas refleja principalmente este segmento.

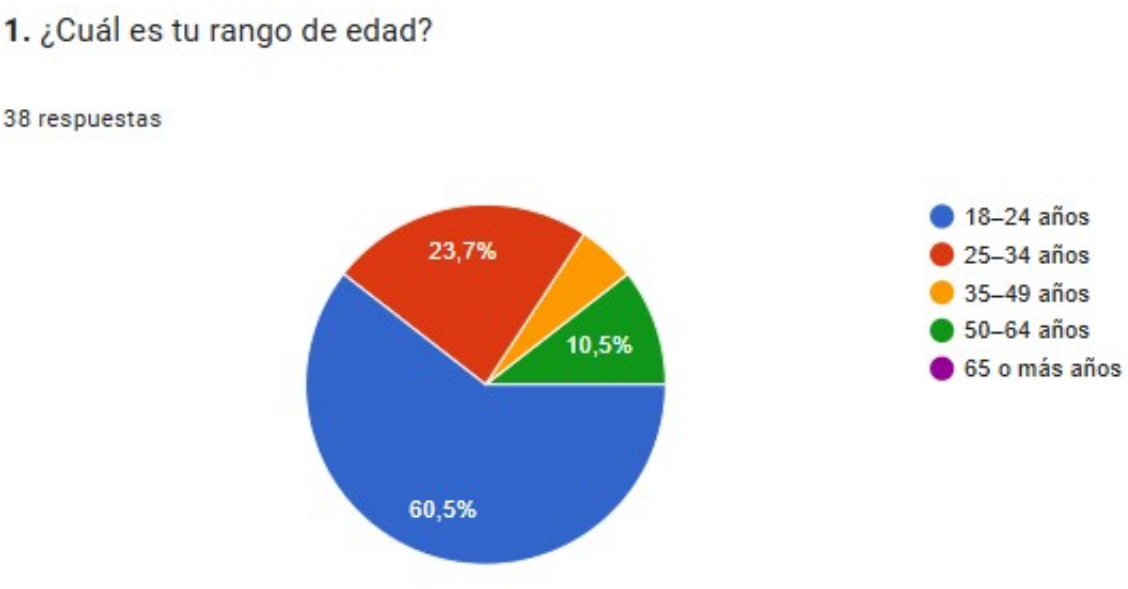


Figura 10.32: Resultados Cuestionario - Pregunta 1

P2. Uso de funciones sin ayuda (Figura 10.33)

- **Descripción:** «Sí, todas sin ayuda»: **84,2 %** (32). «Sí, la mayoría, pero con alguna dificultad»: **10,5 %** (4). «No, necesité ayuda»: **5,3 %** (2).
- **Lectura:** Predomina el uso **autónomo**, lo que respalda la **facilidad de uso** y **aprendizaje** inicial. El **15,8 %** con dificultad o necesidad de ayuda sugiere fricciones puntuales (p. ej., descubribilidad de acciones o nomenclatura) a mejorar en onboarding e interfaz.

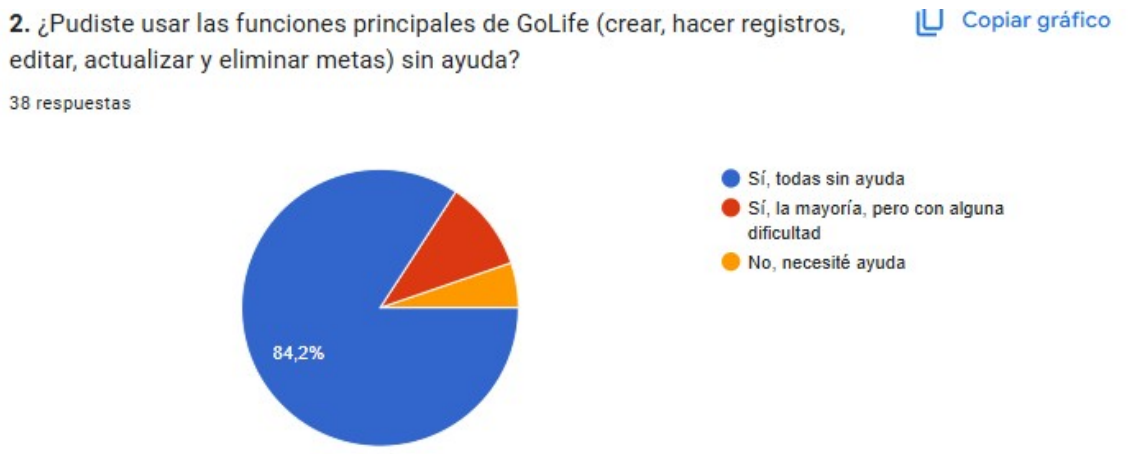


Figura 10.33: Resultados Cuestionario - Pregunta 2

P3. Facilidad para manejar las funciones principales (Figura 10.34)

- **Descripción:** 1: **0 %** (0), 2: **0 %** (0), 3: **5,3 %** (2), 4: **23,7 %** (9), 5: **71,1 %** (27). Agrupado 4–5: **94,7 %** (36). **Media** ≈ 4,66.
- **Lectura:** Valoración **claramente positiva** con fuerte concentración en 5, lo que respalda **facilidad de uso** y **aprendizaje** inicial. Las pocas respuestas en 3 apuntan a mejoras puntuales de claridad o descubribilidad.

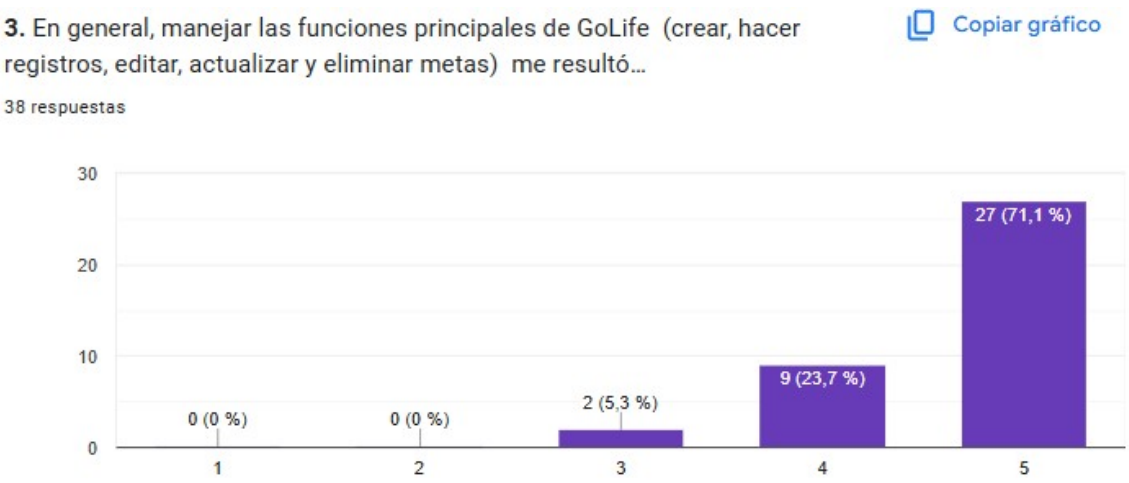


Figura 10.34: Resultados Cuestionario - Pregunta 3

P4. Intuitivo desde un inicio (Figura 10.35)

- **Descripción:** 1: **0 %** (0), 2: **0 %** (0), 3: **10,5 %** (4), 4: **23,7 %** (9), 5: **65,8 %** (25). 4–5: **89,5 %** (34). **Media** ≈ 4,55.
- **Lectura:** Percepción **muy positiva** de la intuición inicial; dos tercios puntúan 5 y casi nueve de cada diez 4–5. El 10,5 % en 3 sugiere mejoras puntuales de **descubribilidad** en el primer uso.

Figura 10.35: Resultados Cuestionario - Pregunta 4

P5. ¿Viste el tutorial de GoLife? (Figura 10.36)

- **Descripción:** «Sí, lo vi completo antes de crear metas»: **57,9 %** (22). «Sí, pero lo vi después de crear metas»: **26,3 %** (10). «No, no lo vi»: **15,8 %** (6). Visto (cualquier momento): **84,2 %** (32).
- **Lectura:** Alta exposición al tutorial, mayoritariamente previa al uso, lo que puede reforzar la buena percepción de facilidad (P2–P4).

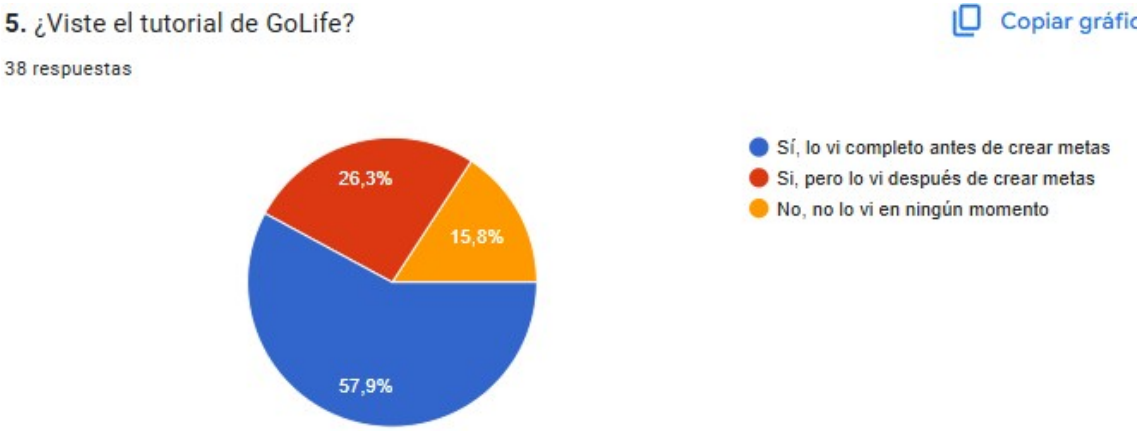


Figura 10.36: Resultados Cuestionario - Pregunta 5

P6. Motivo para ver el tutorial (Figura 10.37)

- **Descripción:** «Quería tener una guía antes de empezar a crear metas»: **52,9 %** (18). «Lo vi para aclarar dudas después de empezar a crear metas»: **32,4 %** (11). «No entendía la aplicación web en un inicio»: **5,9 %** (2). Respuestas en *Otros* (1 cada una; **2,9 %**): «Lo hago siempre», «Para probar las funciones de la app», «Quería asegurar que no me salté nada». «No encontré el tutorial»: **0 %**.
- **Lectura:** Predomina una motivación **proactiva** (guía previa) y, en segundo lugar, la **resolución de dudas** posterior, lo que sugiere que el tutorial aporta valor tanto al inicio, como en el uso continuo. Los pocos casos por falta de comprensión inicial indican

oportunidades de **mejorar el primer uso** (pistas contextuales, tour breve) y mantener el tutorial **accesible e integrado**.

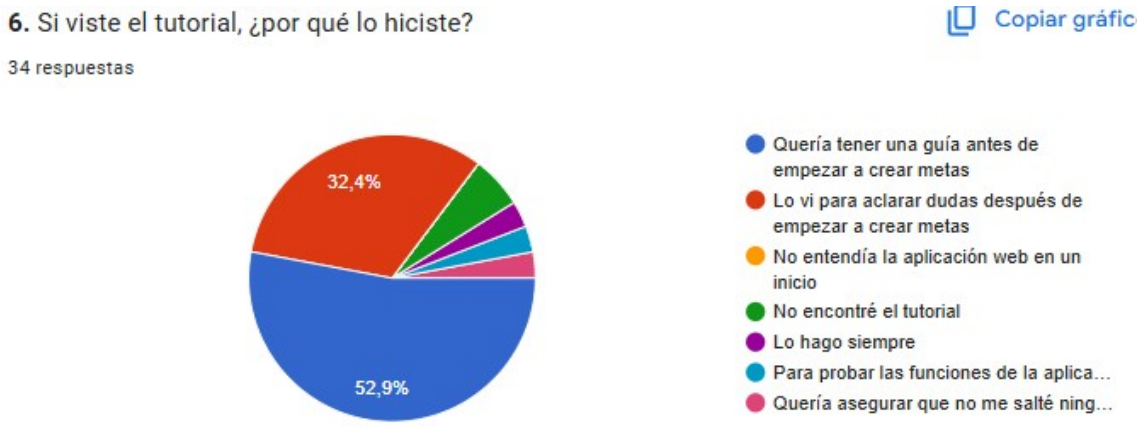


Figura 10.37: Resultados Cuestionario - Pregunta 6

P7. ¿GoLife sirve para distintos tipos de metas? (Figura 10.38)

- **Descripción:** «Sí, sin duda»: **76,3 %** (29). «En parte, pero con limitaciones»: **23,7 %** (9). «No, muy enfocada a un tipo concreto»: **0 %** (0).
- **Lectura:** Percepción **mayoritariamente versátil** del producto, con un cuarto de la muestra señalando **limitaciones**. Lo cual es razonable e intencionado dado el foco genérico de GoLife.

Figura 10.38: Resultados Cuestionario - Pregunta 7

P8. ¿Para qué tipo de metas usaste GoLife? (Figura 10.39)

- **Descripción:** Respuestas abiertas agrupadas en categorías. Distribución sobre válidas (36):
  Deporte / Actividad física **50,0 %** (18);
  Estudios / Académico **16,7 %** (6);
  Ocio / Lectura / Juegos **13,9 %** (5);

Personales / Hábitos **11,1 %** (4);
Proyecto creativo **2,8 %** (1);
Salud / Bienestar **2,8 %** (1);
Trabajo / Negocio **2,8 %** (1).
«*Otros (no tipo de meta)*»: 2 casos (**5,3 %** del total) -p. ej., "solo para probar la aplicación"- no contabilizados en los porcentajes.

- **Lectura:** Predominio claro de **metas deportivas/fitness**, con uso **académico** relevante y presencia de **ocio** y **hábitos personales**. La diversidad sugiere **versatilidad** del producto.

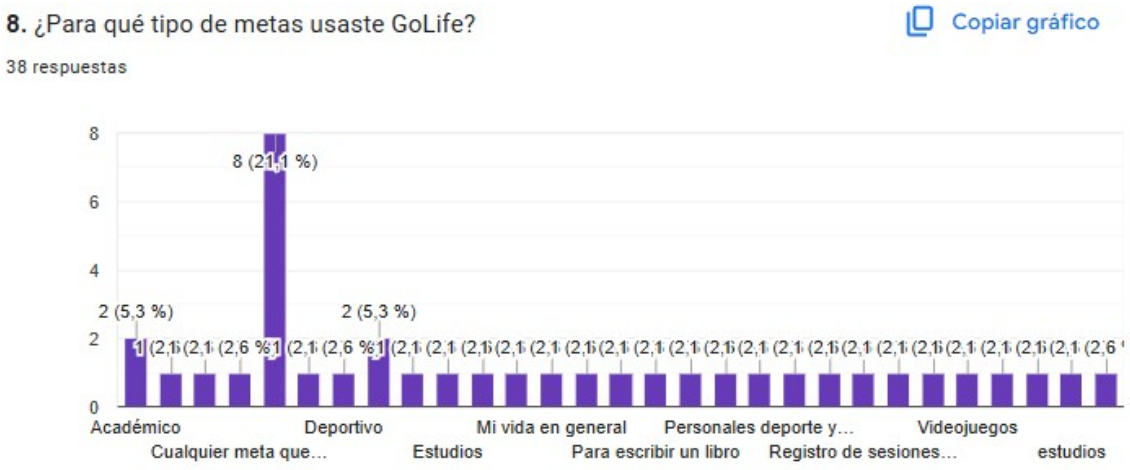


Figura 10.39: Resultados Cuestionario - Pregunta 8

P9. Velocidad de carga (Figura 10.40)

- **Descripción:** 1: **0 %** (0), 2: **0 %** (0), 3: **0 %** (0), 4: **21,1 %** (8), 5: **78,9 %** (30). 4–5: **100 %** (38). **Media** ≈ 4,79.
- **Lectura:** Percepción **muy positiva** del rendimiento percibido; la totalidad puntúa 4–5 y predomina 5. Mantener optimizaciones actuales para sostener esta experiencia.

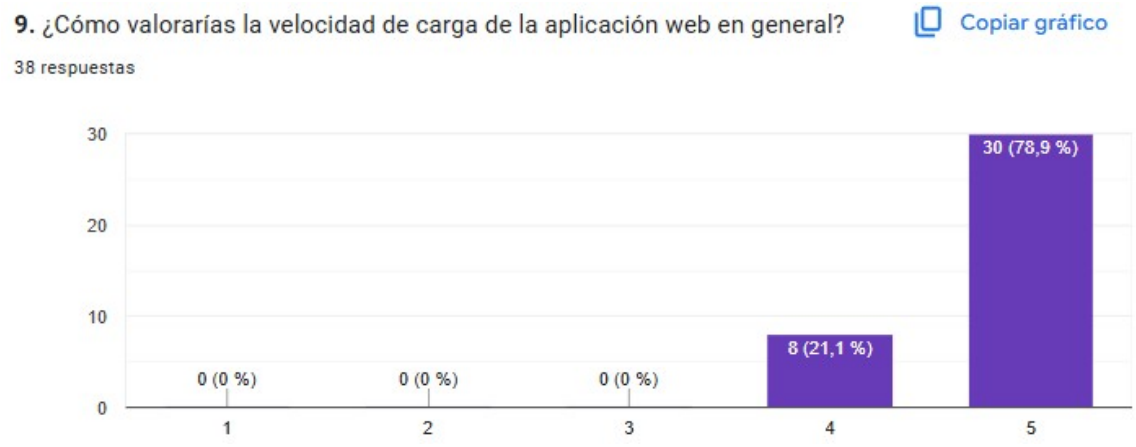


Figura 10.40: Resultados Cuestionario - Pregunta 9

P10. Tiempos de espera o bloqueos (Figura 10.41)

- **Descripción:** «No»: **97,4 %** (37). «Sí, pero muy pocas veces»: **2,6 %** (1). «Sí, con frecuencia»: **0 %** (0).
- **Lectura:** Incidencias **casi inexistentes**; estabilidad y rendimiento percibidos **sólidos**, coherentes con P9. Mantener monitoreo de errores y latencias y revisar el caso aislado para detectar posibles fricciones puntuales.

Figura 10.41: Resultados Cuestionario - Pregunta 10

Conclusiones:

Los resultados muestran una **recepción general positiva** de GoLife.

La muestra está **concentrada en usuarios jóvenes** (P1), por lo que las conclusiones representan sobre todo este perfil y será conveniente ampliar a ≥ 35 años.

El uso fue mayoritariamente **autónomo** (P2) y las valoraciones de *facilidad* e *intuición* fueron altas (P3: media $\approx 4{,}66$; P4: media $\approx 4{,}55$), aunque persisten fricciones puntuales (10,5 % en 3 en P4; 15,8 % reporta alguna dificultad o ayuda en P2).

El **rendimiento percibido** es muy favorable (P9: media $\approx 4{,}79$, 100 % en 4–5; P10: 97,4 % sin esperas/bloqueos). GoLife se percibe **versátil** (P7) y los **usos reales** se concentran en *deporte/actividad física* (50 %), seguidos de *estudios* (16,7 %), *ocio/lectura/juegos* (13,9 %) y *hábitos personales* (11,1 %) (P8).

El **tutorial** se consulta ampliamente (84,2 %, P5), principalmente como guía previa (P6), si bien un 15,8 % no lo usa, por lo que la experiencia debe mantenerse *usable sin depender de él*.

10.4. Verificación de criterios de aceptación

En esta sección verificamos, de forma concisa, los **criterios de aceptación** de los RNF1–RNF6. Para cada RNF se lista el *criterio de aceptación* y, justo debajo, el *cumplimiento* con una breve evidencia o estado (Si / Parcialmente / No). Las evidencias se apoyan en los

resultados de pruebas con usuarios, la revisión de la arquitectura y despliegue en la nube, y los mecanismos de seguridad implementados.

RNF1 - Diseño Genérico

- **Criterio de aceptación:** La interfaz y funcionalidades no deben estar limitadas a un único caso de uso, permitiendo la adaptación a diferentes perfiles de usuarios.
  - **Cumplimiento:** *Sí*, P7 muestra percepción de versatilidad (76,3 % «sí, sin duda»; 23,7 % «en parte»); P8 refleja variedad de usos (deporte, estudios, ocio, hábitos, salud, negocio, proyecto creativo).
- **Criterio de aceptación:** Las pruebas de usuario con un grupo diverso (mínimo 5 perfiles diferentes) deben demostrar que al menos el 80 % de los usuarios pueden utilizar las funciones principales sin asistencia.
  - **Cumplimiento:** *Sí*, P2 indica 84,2 % «todas sin ayuda» ($\geq$ 80 %); la muestra incluye diversidad de perfiles (combinación de rangos de edad, P1, y tipos de metas, P8) ≥ 5.

RNF2 - Interfaz Intuitiva

- **Criterio de aceptación:** Se debe realizar al menos una prueba de usabilidad con usuarios reales y obtener una calificación mínima de 4/5 en facilidad de uso.
  - **Cumplimiento:** *Sí*, prueba con usuarios reales; P3 «facilidad» media $\approx$ 4,66 (94,7 % en 4–5) y P4 «intuición inicial» media $\approx$ 4,55 (89,5 % en 4–5).
- **Criterio de aceptación:** La cantidad de pasos necesarios para completar acciones clave no debe superar los estándares establecidos en aplicaciones similares del mercado.
  - **Cumplimiento:** *Sí*, la UI se diseñó siguiendo la **regla de los 3 clics**; los flujos clave (crear, registrar, editar y eliminar metas) se completan en ≤ 3 clics desde la pantalla principal (dashboard) en la versión evaluada. Apoya esta evidencia P2 (84,2 % «todas sin ayuda», 10,5 % «mayoría con alguna dificultad»).

RNF3 - Escalabilidad

- **Criterio de aceptación:** La arquitectura debe permitir la expansión de almacenamiento y cómputo sin necesidad de una reestructuración completa.
  - **Cumplimiento:** *Sí*, despliegue **100 % cloud** (MongoDB Atlas, Firebase y Google Cloud Platform). La expansión horizontal se consigue **actualizando planes/tiers** de estos servicios (p. ej., mayor tamaño de clúster en Atlas y más cómputo en GCP) sin cambios de arquitectura; únicamente requiere **ajustar el rate limiting** de la API para admitir más usuarios simultáneos. Esto implica **mayor coste operativo**, pero **sin dificultades técnicas** ni reingeniería.

RNF4 - Disponibilidad

- **Criterio de aceptación:** La disponibilidad del sistema debe ser $\geq$ 99 % mensual, medida a través de un sistema de monitoreo automatizado.

  - **Cumplimiento:** *Sí*, los servicios base ofrecen SLA/SLO superiores al 99 %: App Engine (99,95 %) (115), MongoDB Atlas (99,995 %) (116), Firebase Auth (99,95 %) (117), Firebase Hosting (99,95 %) (118) y Cloud KMS (99,95–99,99 %) (119). La disponibilidad se mide con **Cloud Monitoring (GCP)** y métricas nativas de Firebase y Atlas, con alertas automáticas.

- **Criterio de aceptación:** Las interrupciones no planificadas no deben superar las 87,6 horas al año o 7,2 horas al mes.
  - **Cumplimiento:** *Sí*, con 99,95 % mensual el presupuesto teórico de caída es $\approx$ **22 min/mes** (muy por debajo de 7,2 h); Atlas 99,995 % reduce esto a $\approx$ **2,2 min/mes**. Estas cifras se respaldan en los SLA/SLO de los proveedores (115; 116; 117; 118; 119) y quedan registradas mediante Cloud Monitoring e historial de incidentes.

RNF5 - Arquitectura Cloud

- **Criterio de aceptación:** Todos los servicios del back end deben ejecutarse en una infraestructura en la nube.
  - **Cumplimiento:** *Sí*, la API se ejecuta en **Google App Engine**; autenticación con **Firebase Auth**; gestión de claves con **Cloud KMS**; base de datos en **MongoDB Atlas**. No existen componentes fuera de la nube.
- **Criterio de aceptación:** Los datos deben almacenarse en servicios de bases de datos en la nube, sin dependencias de almacenamiento local.
  - **Cumplimiento:** *Sí*, toda la persistencia reside en **MongoDB Atlas** (clúster gestionado en la nube); la aplicación no mantiene almacenamiento local ni volúmenes persistentes propios.

RNF6 - Securización

- **Criterio de aceptación:** El sistema debe implementar un esquema de autenticación basado en tokens JWT con expiración configurable.
  - **Cumplimiento:** *Sí*, la autenticación se delega en **Firebase Auth**, que emite **JWT firmados**. La *expiración* del token se gestiona desde las políticas de sesión de Firebase.
- **Criterio de aceptación:** La API debe validar la autenticidad de los tokens en cada solicitud protegida y rechazar aquellas sin un token válido.
  - **Cumplimiento:** *Sí*. Cada petición a rutas protegidas debe incluir `Authorization: Bearer <JWT>`. Un filtro en la API extrae el token y lo valida. Si falta el token o no es válido, la solicitud se **rechaza** con `401 - No Autorizado`.

10.5. Cambios post-tests

Esta sección recoge los **ajustes realizados tras el test con usuarios** y el **análisis del cuestionario** (Google Forms), complementados con **comentarios directos** de varios participantes. El objetivo es **cerrar el ciclo de mejora**: identificar fricciones, priorizarlas y aplicar cambios concretos que mejoren la descubribilidad, la facilidad de uso.

1. Título contextual en el alta de registros

- **Motivación:** algunos usuarios dudaban sobre a qué meta estaban añadiendo el registro.
- **Implementación:** el diálogo de registro muestra ahora un encabezado dinámico del tipo Nuevo registro de: *<Nombre de la meta>*.
- **Impacto:** reduce errores de contexto y mejora la trazabilidad de los datos.

Figura 10.42: Cambio post-test: Alta de registros

2. Mini–tutorial al cerrar el mensaje de bienvenida

- **Motivación:** parte de los usuarios tenían dificultades al encontrar el tutorial.
- **Implementación:** al descartar el mensaje de bienvenida se dispara un mini–tutorial de un paso que resalta exclusivamente el botón de *Tutorial* en la barra superior.
- **Impacto:** aumenta el descubrimiento de la ayuda interactiva sin interrumpir el flujo.

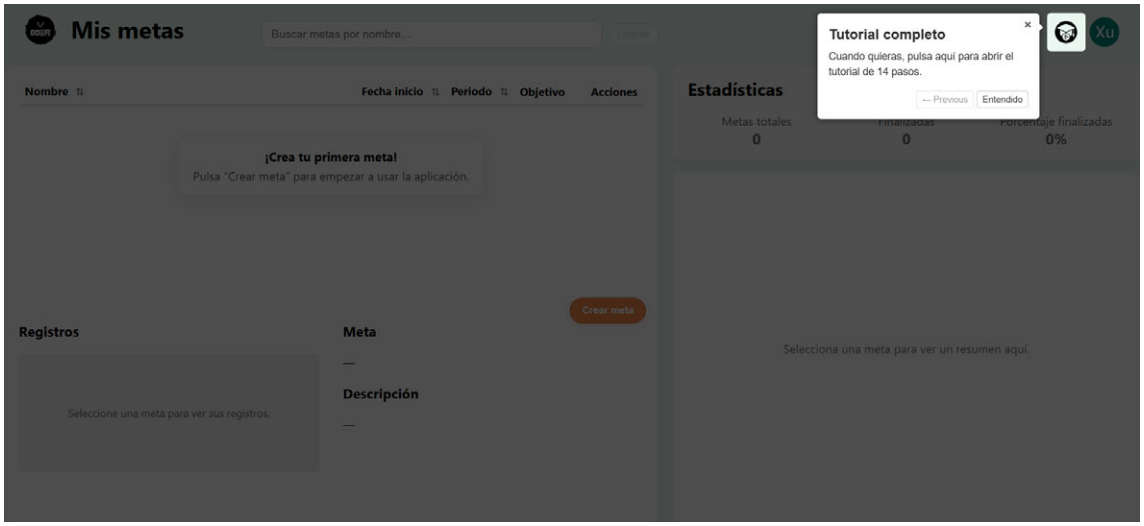


Figura 10.43: Cambio post-test: Guía a Tutorial

3. Limpieza del buscador de metas al finalizar el tutorial

- **Motivación:** el término de búsqueda quedaba persistente y ocultaba metas, generando confusión.
- **Implementación:** al completar el tutorial se vacía el campo de búsqueda y se restablece la lista completa.
- **Impacto:** evita estados "vacíos" inesperados y mejora la percepción de control.

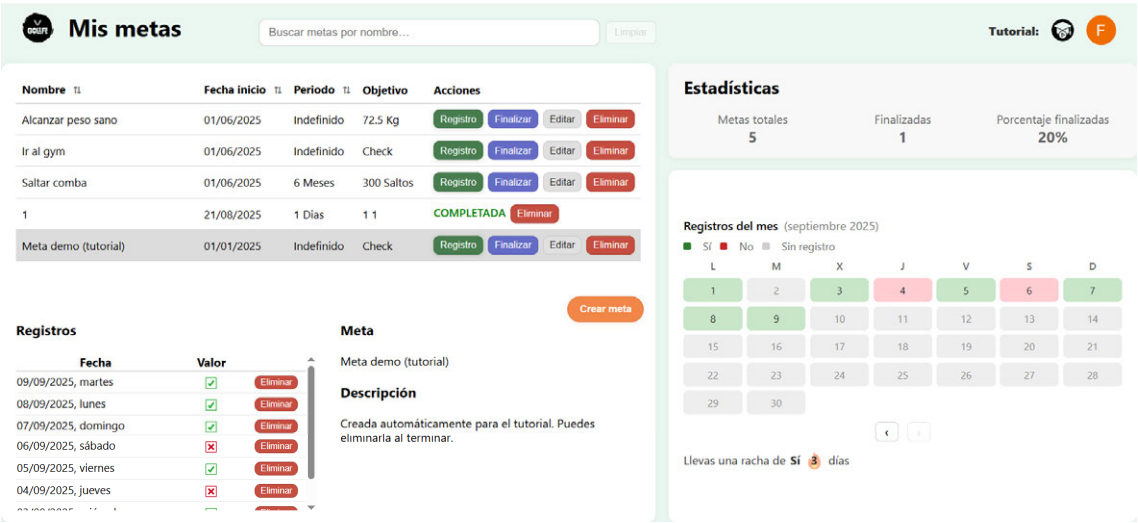


Figura 10.44: Cambio post-test: Limpieza buscador tras Tutorial

4. Mejora del contenido del tutorial en *Crear meta* (objetivo y unidad)

- **Motivación:** hubo dudas entre metas cuantitativas (con unidad) y metas tipo *check*.
- **Implementación:** se reescribió el paso del tutorial con ejemplos concretos (p. ej., «Correr 5 km = objetivo: 5 + unidad: km», «asistencia al gym: *Sí/No*»), aclarando la relación *valor + unidad* y la alternativa *check*.
- **Impacto:** reduce ambigüedad y errores de configuración inicial.

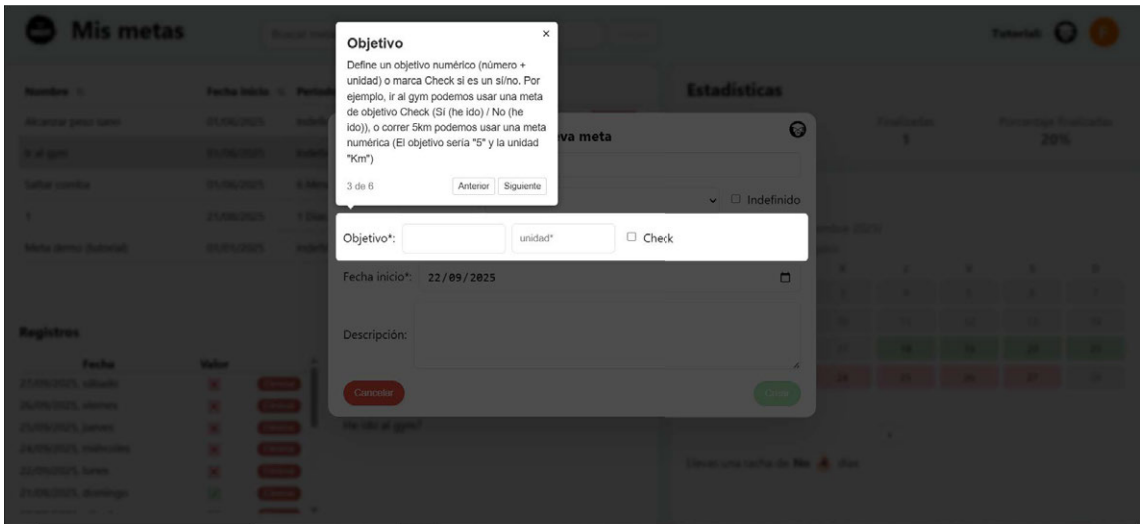


Figura 10.45: Cambio post-test: Tutorial en Crear meta

5. Feedback explícito ante contraseñas inválidas

- **Motivación:** cuando la contraseña no cumplía los mínimos, el error pasaba desapercibido.
- **Implementación:** al detectar una contraseña no válida se abre un modal informativo con los requisitos (longitud, mayúsculas/minúsculas, número y carácter especial) y un enlace a la guía de «cómo crear una contraseña segura».
- **Impacto:** aumenta la tasa de éxito en autenticación y estandariza la seguridad percibida.

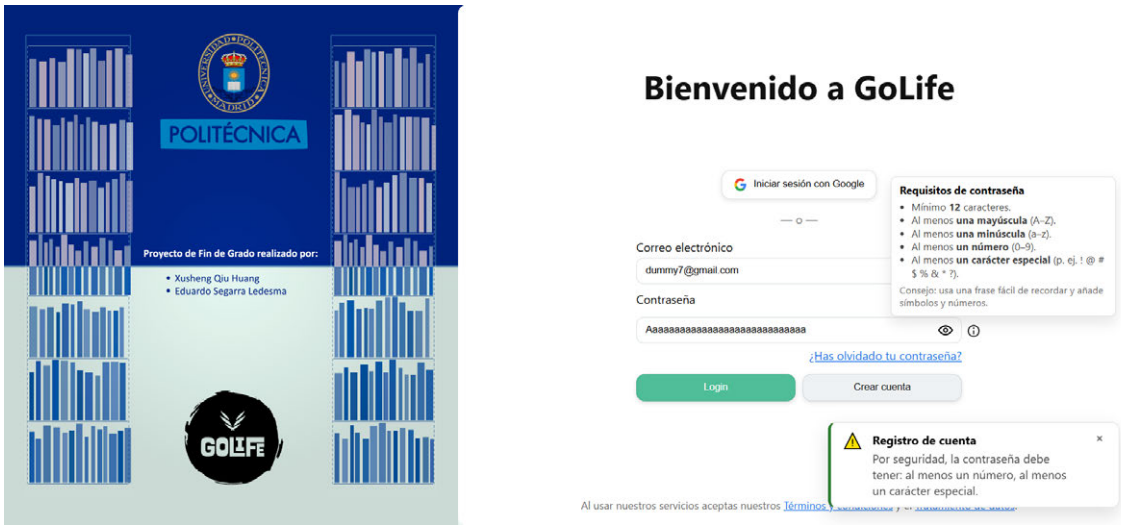


Figura 10.46: Cambio post-test: Feedback ante contraseña invalida

Capítulo 11

Conclusiones y trabajos futuros

11.1. Conclusiones

Este proyecto partía de una premisa sencilla y exigente: servir de ejercicio práctico de desarrollo cloud y demostrar que una aplicación web 100 % cloud, con foco en la sencillez y en el seguimiento de metas, puede ofrecer una experiencia usable, rápida y válida para distintos contextos de uso.

A la luz del sistema construido y de las evidencias recogidas, la respuesta es **sí**. Por un lado, los objetivos planteados en la introducción (plataforma de seguimiento de metas esencial, desplegada íntegramente en la nube y validada con usuarios reales) se han materializado: la solución permite crear, editar, actualizar y eliminar metas desde cualquier navegador, y se apoya en servicios gestionados (Firebase Hosting/Auth, App Engine, Cloud KMS y MongoDB Atlas) que reducen la carga operativa y habilitan crecimiento ordenado sin reingeniería. Esto cumple con el propósito de combinar **experiencia clara** y **base tecnológica robusta** definido al inicio.

Los resultados de la sesión exploratoria con usuarios refuerzan esa conclusión: la mayoría pudo usar las funciones principales sin ayuda y valoró positivamente la **facilidad** y la **intuición** de la interfaz, así como la **rapidez** percibida. Además, la herramienta se percibe **versátil**, con usos reportados en deporte, estudios, hábitos y ocio, lo que respalda la premisa de un diseño genérico. En paralelo, el despliegue **100 % cloud** (API, autenticación y almacenamiento gestionados) demostró que es viable sostener la sencillez del producto con una arquitectura preparada para crecer sin reingeniería, apoyándose en escalado de servicios y monitorización nativa.

Como reflexión final, el proyecto deja dos aprendizajes clave. Primero, que **menos es más**: limitarse a lo esencial reduce la curva de aprendizaje y mejora la adopción sin sacrificar valor. Segundo, que la **nube** no es solo un entorno de despliegue, sino un acelerador de producto: permite concentrarse en la experiencia mientras mantiene abiertas las puertas a la escalabilidad y libertad de diseño.

Con todo, el balance es nítido: GoLife **cumple el propósito fundacional** y sienta una base sólida (técnica y de experiencia) sobre la que seguir creciendo con control.

11.2. Impacto ODS

Los **Objetivos de Desarrollo Sostenible (ODS)** son un marco de 17 metas impulsadas por Naciones Unidas dentro de la Agenda 2030 para abordar desafíos globales, entre ellos salud, educación, innovación, igualdad y clima, mediante acciones coordinadas de gobiernos, empresas, instituciones y ciudadanía (120). Este marco sirve como referencia común para orientar y evaluar iniciativas que buscan generar valor social y ambiental, además de resultados técnicos o de negocio.

En este contexto, por **impacto ODS** entendemos la contribución que un proyecto realiza de forma **directa** (por el uso y las funcionalidades que habilita) o **indirecta** (por su forma de operar, gobernanza y buenas prácticas) al avance de uno o varios ODS.

Figura 11.1: Objetivos de Desarrollo Sostenible

En el caso de GoLife, esta contribución se argumenta a partir de evidencias observables (usos reales, resultados del cuestionario, decisiones de diseño y despliegue) y se presenta como una *evaluación razonada*, no como una certificación formal de cumplimiento.

11.2.1.  ODS 3: Salud y Bienestar

Cómo aplica (directa): GoLife facilita **definir y seguir** hábitos saludables (p. ej., ejercicio físico) con una experiencia simple y accesible, alineada con el ODS 3 de **garantizar una vida sana y promover el bienestar para todos** (121).

Por qué y cómo: En las pruebas, el uso más frecuente fue **deporte/actividad física**; el registro rápido y el progreso visual fomentan la adherencia diaria. Así, el ODS 3 se traduce en acciones concretas: fijar objetivos, anotar avances y mantener rutinas de salud de forma sostenida (121).

11.2.2.  ODS 4: Educación de Calidad

Cómo aplica (directa): GoLife apoya el **aprendizaje autónomo** al permitir fijar objetivos de estudio, planificar sesiones y seguir el progreso, en línea con el ODS 4 de **garantizar una educación inclusiva y de calidad y promover oportunidades de aprendizaje durante toda la vida** (122).

Por qué y cómo: La interfaz sencilla y accesible desde cualquier navegador reduce barreras de entrada y facilita rutinas de estudio sostenidas: definir metas, registrar sesiones y visualizar avances, reforzando la organización y continuidad del aprendizaje (122).

11.2.3.  ODS 9: Industria, Innovación e Infraestructuras

Cómo aplica (indirecta): GoLife impulsa la **innovación digital** al construirse sobre infraestructura **100 % cloud** escalable y de alta disponibilidad, alineada con el ODS 9 de **desarrollar infraestructuras fiables, sostenibles y resilientes y fomentar la innovación** (123).

Por qué y cómo: El uso de servicios gestionados (hosting, autenticación, cómputo y base de datos) reduce la carga operativa y favorece la escalabilidad sin reingeniería; esto facilita iterar en producto y hacer llegar la solución a más usuarios con menor fricción técnica, en sintonía con la infraestructura e innovación promovidas por el ODS 9 (123).

11.3. Lineas futuras

Por último, se presentan varias **posibles líneas de trabajo futuras** sobre las que iterar y mejorar GoLife y sus posibles nuevas versiones a partir de lo aprendido en este proyecto.

1. Web responsive (uso en navegadores móviles)

- **Objetivo:** Hacer la interfaz completamente *responsive* para teléfonos y tabletas.
- **Abordaje propuesto:** Adaptar la interfaz para que se vea y funcione bien en pantallas pequeñas: diseño que se ajusta solo, botones grandes y fáciles de pulsar, textos legibles y espacios adecuados; además, mejorar los tiempos de carga en móvil.
- **Impacto esperado:** Aumenta el alcance y la accesibilidad; reduce fricción en registros "sobre la marcha".

2. Widgets de escritorio (interacción rápida desde PC)

- **Objetivo:** Permitir *quick actions* sin abrir el navegador: añadir registro, marcar meta, ver progreso.
- **Abordaje propuesto:** Mini-aplicaciones o *widgets* para Windows (bandeja del sistema, atajo global), notificaciones nativas, autenticación con token y límites de tasa alineados con la API.

- **Impacto esperado:** Incrementa la frecuencia de uso y el cumplimiento de metas al reducir el tiempo de acceso.

3. Aplicación móvil

- **Objetivo:** Ofrecer una experiencia nativa con notificaciones y registro offline.
- **Abordaje propuesto:** App móvil (Android como prioridad inicial) con paridad funcional básica (metas y registros), *offline-first* y *push notifications* y sincronización con la API existente.
- **Impacto esperado:** Mejora la retención y el compromiso diario; facilita el registro contextual de hábitos.

Bibliografía

[1] American Psychological Association. (2023) Self-management. [Online]. Available: https://dictionary.apa.org/self-management

[2] OECD, *OECD Skills Outlook 2021: Learning for Life*. Paris: OECD Publishing, 2021. [Online]. Available: https://doi.org/10.1787/0ae365b4-en

[3] B. Aeon, A. Faber, and A. Panaccio, "Does time management work? a meta-analysis," *PLOS ONE*, vol. 16, no. 1, p. e0245066, 2021. [Online]. Available: https://doi.org/10.1371/journal.pone.0245066

[4] B. Gardner, "What is habit and how can it be used to change real-world behaviour? narrowing the theory–reality gap," *Social and Personality Psychology Compass*, vol. 18, no. 7, p. e12975, 2024. [Online]. Available: https://doi.org/10.1111/spc3.12975

[5] OECD, "Recommendation of the council on financial literacy," OECD Legal Instruments, 2020. [Online]. Available: https://legalinstruments.oecd.org/en/instruments/OECD-LEGAL-0461

[6] E. A. Locke and G. P. Latham, "The development of goal setting theory: A half century retrospective," *Motivation Science*, vol. 5, no. 2, pp. 93–105, 2019. [Online]. Available: https://doi.org/10.1037/mot0000127

[7] I. Notion Labs. (2025) Notion — one workspace. [Online]. Available: https://www.notion.com/

[8] Doist. (2025) Todoist — página principal (es). [Online]. Available: https://www.todoist.com/es

[9] I. Habitica. (2025) Habitica — home. [Online]. Available: https://habitica.com/static/home

[10] Any.do. (2025) Any.do — to do list, planner & calendar. [Online]. Available: https://www.any.do/

[11] (n.d.) ¿cuál es la diferencia entre paas, iaas, saas y caas? Google Cloud. [Online]. Available: https://cloud.google.com/learn/paas-vs-iaas-vs-saas?hl=es_419

[12] (2023) Modelos de servicio cloud: Iaas, paas y saas. Stackscale. [Online]. Available: https://www.stackscale.com/es/blog/modelos-de-servicio-cloud/

[13] (n.d.) ¿qué es baas? | backend como servicio vs. sin servidor. Cloudflare. [Online]. Available: https://www.cloudflare.com/es-es/learning/serverless/glossary/backend-as-a-service-baas/

[14] (2023) ¿qué es backend as a service (baas)? IONOS. [Online]. Available: https://www.ionos.es/digitalguide/servidores/know-how/backend-as-a-service-baas/

[15] (n.d.) ¿qué es la base de datos como {servicio} (dbaas)? IBM. [Online]. Available: https://www.ibm.com/es-es/think/topics/dbaas

[16] (n.d.) Database as a service (dbaas) explained. MongoDB. [Online]. Available: https://www.mongodb.com/resources/basics/databases/database-as-a-service

[17] Google Cloud. (2025) App engine documentation. [Online]. Available: https://cloud.google.com/appengine#documentation

[18] (n.d.) Key management service (kms): Documentación. Google Cloud. [Online]. Available: https://cloud.google.com/kms/docs/key-management-service?hl=es-419

[19] Firebase. (2025) Firebase hosting documentation. [Online]. Available: https://firebase.google.com/docs/hosting?hl=es-419

[20] ——. (2025) Guía de autenticación de firebase. [Online]. Available: https://firebase.google.com/docs/auth?hl=es-419

[21] MongoDB, Inc. (2025) Mongodb atlas. [Online]. Available: https://www.mongodb.com/es/atlas

[22] (n.d.) Kanbanflow. KanbanFlow. [Online]. Available: https://kanbanflow.com/

[23] (n.d.) Intellij idea. JetBrains. [Online]. Available: https://www.jetbrains.com/idea/

[24] (n.d.) Github. GitHub, Inc. [Online]. Available: https://github.com/

[25] GitHub, Inc. (2025) Github actions. [Online]. Available: https://github.com/features/actions

[26] (n.d.) Mongodb compass documentation. MongoDB. [Online]. Available: https://www.mongodb.com/docs/compass/

[27] (n.d.) Docker desktop. Docker, Inc. [Online]. Available: https://www.docker.com/products/docker-desktop/

[28] E. International. (2015) Ecmascript 2015 language specification (ecma-262, 6th edition). [Online]. Available: https://ecma-international.org/wp-content/uploads/ECMA-262_6th_edition_june_2015.pdf

[29] R. Team. (2025) Writing markup with jsx. [Online]. Available: https://react.dev/learn/writing-markup-with-jsx

[30] ——. (2025) React versions and docs policy. [Online]. Available: https://react.dev/versions

[31] WHATWG. (2025) Html living standard. [Online]. Available: https://html.spec.whatwg.org/multipage/

[32] W. C. W. Group. (2025) Css snapshot 2024. [Online]. Available: https://www.w3.org/TR/css-2024/

[33] M. O. Source. (2022) react@18.2.0 on npm. [Online]. Available: https://www.npmjs.com/package/react/v/18.2.0

[34] T. R. Team. (2022) React v18.0. [Online]. Available: https://react.dev/blog/2022/03/29/react-v18

[35] R. Software. (2025) React router — official documentation. [Online]. Available: https://reactrouter.com/

[36] ——. (2025) remix-run/react-router (github). [Online]. Available: https://github.com/remix-run/react-router

[37] R. Team. (2024) React router v7. [Online]. Available: https://remix.run/blog/react-router-v7

[38] V. Team. (2025) Vitest — getting started. [Online]. Available: https://vitest.dev/guide/

[39] ——. (2025) Vitest — test api reference. [Online]. Available: https://vitest.dev/api/

[40] M. O. Source. (2022) Create react app — getting started. [Online]. Available: https://create-react-app.dev/docs/getting-started/

[41] ——. (2022) react-scripts — npm package. [Online]. Available: https://www.npmjs.com/package/react-scripts

[42] Y. mirror. (2022) Create react app 5.0.1 release notes. [Online]. Available: https://classic.yarnpkg.com/en/package/react-scripts

[43] M. Carroll and R. Hanlon. (2025) Sunsetting create react app. [Online]. Available: https://react.dev/blog/2025/02/14/sunsetting-create-react-app

[44] (n.d.) Más información sobre firebase para web (versión modular). Firebase. [Online]. Available: https://firebase.google.com/docs/web/learn-more?hl=es-419#modular-version

[45] A. Project. (2025) Axios — getting started. [Online]. Available: https://axios-http.com/docs/intro

[46] ——. (2025) Axios api reference. [Online]. Available: https://axios-http.com/docs/api_intro

[47] K. Ahmed. (2025) Driver.js — documentation. [Online]. Available: https://driverjs.com/docs/

[48] ——. (2025) driver.js@1.3.6 — npm. [Online]. Available: https://www.npmjs.com/package/driver.js/v/1.3.6/

[49] G. C. Team. (2025) Googlechrome/web-vitals (github). [Online]. Available: https://github.com/GoogleChrome/web-vitals

[50] web.dev by Google. (2020) Web vitals — overview. [Online]. Available: https://web.dev/articles/vitals

[51] T. Library. (2024) React testing library — introduction. [Online]. Available: https://testing-library.com/docs/react-testing-library/intro/

[52] ——. (2022) @testing-library/jest-dom — docs. [Online]. Available: https://testing-library.com/docs/ecosystem-jest-dom/

[53] ——. (2023) @testing-library/user-event v13.5.0 — docs. [Online]. Available: https://testing-library.com/docs/user-event/v13/

[54] jsdom contributors. (2025) jsdom — a javascript implementation of web standards. [Online]. Available: https://github.com/jsdom/jsdom

[55] V. Team. (2025) Vitest — coverage (v8 provider). [Online]. Available: https://vitest.dev/guide/coverage

[56] ——. (2025) @vitest/coverage-v8 — npm. [Online]. Available: https://www.npmjs.com/package/%40vitest/coverage-v8

[57] (n.d.) Documentación de java se 17. Oracle. [Online]. Available: https://docs.oracle.com/en/java/javase/17/

[58] (n.d.) Openjdk project jdk 17. OpenJDK. [Online]. Available: https://openjdk.org/projects/jdk/17/

[59] (n.d.) Spring boot. Spring. [Online]. Available: https://spring.io/projects/spring-boot

[60] (2025) google-cloud-kms 2.74.0 — maven central. Sonatype Central Repository. [Online]. Available: https://central.sonatype.com/artifact/com.google.cloud/google-cloud-kms/2.74.0

[61] (n.d.) Configurar el sdk de firebase admin. Firebase. [Online]. Available: https://firebase.google.com/docs/admin/setup

[62] (2025) firebase-admin 9.5.0 — maven central. Sonatype Central Repository. [Online]. Available: https://central.sonatype.com/artifact/com.google.firebase/firebase-admin/9.5.0

[63] (n.d.) Bucket4j — token bucket rate limiting. Bucket4j. [Online]. Available: https://github.com/bucket4j/bucket4j

[64] B. Manes. (n.d.) Caffeine: A high performance caching library for java. [Online]. Available: https://caffeine.github.io/

[65] (n.d.) Mongodb java driver (sync) documentation. MongoDB. [Online]. Available: https://www.mongodb.com/docs/drivers/java/sync/current/

[66] (n.d.) springdoc-openapi. springdoc.org. [Online]. Available: https://springdoc.org/

[67] (n.d.) Testing features in spring boot. Spring. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#features.testing

[68] (n.d.) Testing — spring security reference. Spring. [Online]. Available: https://docs.spring.io/spring-security/reference/servlet/test/

[69] (n.d.) Testcontainers for java. Testcontainers. [Online]. Available: https://java.testcontainers.org/

[70] S. Birkner. (n.d.) system-lambda. [Online]. Available: https://github.com/stefanbirkner/system-lambda

[71] (n.d.) Spring boot maven plugin — reference documentation. Spring. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/

[72] (n.d.) Maven surefire plugin. Apache Maven. [Online]. Available: https://maven.apache.org/surefire/maven-surefire-plugin/

[73] (n.d.) Maven failsafe plugin. Apache Maven. [Online]. Available: https://maven.apache.org/surefire/maven-failsafe-plugin/

[74] (n.d.) Jacoco maven plugin. JaCoCo. Consultado el 23 de agosto de 2025. [Online]. Available: https://www.jacoco.org/jacoco/trunk/doc/maven.html

[75] (n.d.) Maven resources plugin. Apache Maven. [Online]. Available: https://maven.apache.org/plugins/maven-resources-plugin/

[76] (n.d.) Overleaf. Overleaf. [Online]. Available: https://es.overleaf.com/

[77] (n.d.) diagrams.net (draw.io). diagrams.net. [Online]. Available: https://www.drawio.com/

[78] (n.d.) Staruml. StarUML. [Online]. Available: https://staruml.io/

[79] Online Gantt. (2025) Online gantt (gantt chart tool). [Online]. Available: https://www.onlinegantt.com/#/gantt

[80] (2025) Figma. Figma, Inc. [Online]. Available: https://www.figma.com/es-es/

[81] (n.d.) Swagger ui. SmartBear Software. [Online]. Available: https://swagger.io/tools/swagger-ui/

[82] (n.d.) Openapi initiative. Linux Foundation. [Online]. Available: https://www.openapis.org/

[83] (2025) Páginas de github. GitHub, Inc. [Online]. Available: https://docs.github.com/es/pages

[84] Glassdoor. (2025) Ingeniero de software — salarios en españa. [Online]. Available: https://www.glassdoor.es/Sueldos/ingeniero-de-software-sueldo-SRCH_KO0,21.htm

[85] Google Cloud. (2025) Funciones gratuitas de google cloud. [Online]. Available: https://cloud.google.com/free/docs/free-cloud-features?hl=es-419

[86] ——. (2025) App engine pricing. [Online]. Available: https://cloud.google.com/appengine/pricing?hl=es-419

[87] Firebase. (2025) Planes y precios de firebase. [Online]. Available: https://firebase.google.com/pricing?hl=es-419

[88] MongoDB, Inc. (2025) Planes y precios de mongodb atlas. [Online]. Available: https://www.mongodb.com/pricing

[89] Kanban University, "La guía oficial del método kanban," 2021, versión V.1 (Feb '21). Copyright © Mauvius Group Inc. [Online]. Available: https://resources.kanban.university/wp-content/uploads/2021/08/The-Official-Kanban-Guide_Spanish_A4.pdf

[90] (n.d.) Acid compliance in mongodb: Can nosql databases be acid compliant? MongoDB. [Online]. Available: https://www.mongodb.com/resources/products/capabilities/acid-compliance

[91] D. Coupal and K. W. Alger. (2019) Building with patterns: The extended reference pattern. MongoDB. Actualizado 2019-03-19. [Online]. Available: https://www.mongodb.com/company/blog/building-with-patterns-the-extended-reference-pattern

[92] ——. (2019) Building with patterns: The computed pattern. MongoDB. Actualizado 2020-03-30. [Online]. Available: https://www.mongodb.com/company/blog/building-with-patterns-the-computed-pattern

[93] Polymorphic schema pattern. MongoDB. [Online]. Available: https://www.mongodb.com/docs/manual/data-modeling/design-patterns/polymorphic-data/polymorphic-schema-pattern/

[94] Mongodb limits and thresholds. MongoDB Documentation. [Online]. Available: https://www.mongodb.com/docs/manual/reference/limits/

[95] Get schema advice (cloud manager api) — performance advisor. MongoDB Documentation. Incluye el trigger `DOCS_CONTAIN_UNBOUNDED_ARRAY` con descripción Arrays with over 10,000 entries. [Online]. Available: https://www.mongodb.com/docs/cloud-manager/reference/api/performance-advisor/get-schema-advice/

[96] Art. 4 gdpr — definitions. GDPR.eu / GDPR-Info. [Online]. Available: https://gdpr-info.eu/art-4-gdpr/

[97] Art. 32 gdpr — security of processing. GDPR.eu / GDPR-Info. [Online]. Available: https://gdpr-info.eu/art-32-gdpr/

[98] S. Brown. The c4 model for visualising software architecture — introduction. [Online]. Available: https://c4model.com/introduction

[99] ——. C4 model — deployment diagrams. [Online]. Available: https://c4model.com/diagrams/deployment

[100] Gcp regions and zones map. Economize. [Online]. Available: https://www.economize.cloud/resources/gcp/regions-zones-map

[101] Ubicaciones de almacenamiento y procesamiento de datos | firebase. Google Firebase. [Online]. Available: https://firebase.google.com/support/privacy?hl=es-419#data_storage_and_processing_locations

[102] Descripción general de identity and access management (iam). Google Cloud. [Online]. Available: https://cloud.google.com/iam/docs/overview?hl=es

[103] Descripción general de las cuentas de servicio. Google Cloud. [Online]. Available: https://cloud.google.com/iam/docs/service-account-overview?hl=es-419

[104] Verify id tokens | firebase authentication. Google Firebase. [Online]. Available: https://firebase.google.com/docs/auth/admin/verify-id-tokens

[105] Cloud kms grpc. Google Cloud. [Online]. Available: https://cloud.google.com/kms/docs/grpc?hl=es-419

[106] Descripción general de grpc | api gateway. Google Cloud. [Online]. Available: https://cloud.google.com/api-gateway/docs/grpc-overview?hl=es-419

[107] Mongodb wire protocol. MongoDB. [Online]. Available: https://www.mongodb.com/docs/manual/reference/mongodb-wire-protocol/

[108] Create and use strong passwords. Microsoft. [Online]. Available: https://support.microsoft.com/en-us/windows/create-and-use-strong-passwords-c5cebb49-8c53-4f5e-2bc4-fe357ca048eb

[109] Algoritmos compatibles | cloud kms — firma mac. Google Cloud. [Online]. Available: https://cloud.google.com/kms/docs/algorithms?hl=es-419#mac_signing_algorithms

[110] Guidelines 01/2025 on pseudonymisation. European Data Protection Board (EDPB). [Online]. Available: https://www.edpb.europa.eu/system/files/2025-01/edpb_guidelines_202501_pseudonymisation_en.pdf

[111] Clústeres de mongodb: conceptos fundamentales. MongoDB. [Online]. Available: https://www.mongodb.com/es/resources/products/fundamentals/clusters

[112] Back up and restore your cluster — mongodb atlas. MongoDB. [Online]. Available: https://www.mongodb.com/docs/atlas/backup-restore-cluster/

[113] APIDog. (2025) Implementing rate limiting in apis. [Online]. Available: https://apidog.com/blog/implementing-rate-limiting-in-apis/

[114] (2025) Privacidad y seguridad en firebase. Firebase / Google. [Online]. Available: https://firebase.google.com/support/privacy?hl=es-419

[115] Google Cloud. (2020) App engine service level agreement (sla). [Online]. Available: https://cloud.google.com/appengine/sla

[116] MongoDB, Inc. (2025) Reliability — mongodb atlas. [Online]. Available: https://www.mongodb.com/cloud/atlas/reliability

[117] Google Cloud. (2019) Identity platform service level agreement (sla). [Online]. Available: https://cloud.google.com/identity-platform/sla

[118] Google Firebase. (2020) Service level agreement for hosting and realtime database. [Online]. Available: https://firebase.google.com/terms/service-level-agreement

[119] Google Cloud. (2021) Cloud key management service and cloud hsm service level agreement (sla). [Online]. Available: https://cloud.google.com/kms/sla

[120] Naciones Unidas. (2018) ¿sabes cuáles son los 17 objetivos de desarrollo sostenible? [Online]. Available: https://www.un.org/sustainabledevelopment/es/2018/08/sabes-cuales-son-los-17-objetivos-de-desarrollo-sostenible/

[121] ——. Salud y bienestar — objetivos de desarrollo sostenible. [Online]. Available: https://www.un.org/sustainabledevelopment/es/health/

[122] ——. Educación de calidad — objetivos de desarrollo sostenible. [Online]. Available: https://www.un.org/sustainabledevelopment/es/education/

[123] ——. Industria, innovación e infraestructura — objetivos de desarrollo sostenible. [Online]. Available: https://www.un.org/sustainabledevelopment/es/infrastructure/

[124] V. M. Dominguez Rivas, *Plantilla TFG ETSISI UPM*. ETSISI, 2020.

[125] G. Firebase. (2025) Upgrade from the namespaced api to the modular web sdk. [Online]. Available: https://firebase.google.com/docs/web/modular-upgrade

[126] F. Team. (2021) Introducing the new firebase js sdk (modular). [Online]. Available: https://firebase.blog/posts/2021/07/introducing-the-new-firebase-js-sdk/

Anexos

Casos de Uso Extendidos de GoLife

ID:	CU-1
Nombre:	Crear Cuenta de usuario
Actor primario:	Usuario
Actores secundarios:	Firebase
Descripción:	Los usuarios pueden crear su cuenta en el sistema
Evento activador:	El usuario pulsa el botón de “Crear Cuenta” en la pantalla de Login
Precondiciones:	El usuario no debe tener una cuenta existente con el mismo correo en el sistema
Postcondiciones:	• La cuenta ha sido creada exitosamente • El usuario puede loguearse con sus credenciales
Flujo Normal:	1. El usuario accede a la pantalla de Login 2. El usuario introduce sus credenciales en la pantalla de Login (correo y contraseña) 3. El usuario pulsa el botón de “Crear Cuenta” 4. El sistema envía los datos ingresados a Firebase 5. <<include>> CU-2 Registrar Usuario 6. El sistema recibe la respuesta de Firebase 7. El sistema crea el perfil del usuario en su base de datos 8. El sistema carga los datos del usuario 9. El sistema carga la pantalla de Dashboard
Flujo Alternativo:	6.a.1. El sistema recibe una respuesta de error de Firebase 6.a.2. El sistema muestra el error en la pantalla de Login
Prioridad:	Alta
Otra información:	
Suposiciones:	

ID:	CU-2
Nombre:	Registrar Usuario
Actor primario:	Firebase
Actores secundarios:	
Descripción:	Firebase crea un nuevo registro de usuario en su base de datos
Evento activador:	CU-1 Crear Cuenta de Usuario
Precondiciones:	El sistema envía los datos requeridos a Firebase (correo y contraseña)
Postcondiciones:	Firebase ha enviado una respuesta al sistema
Flujo Normal:	1. Firebase recibe los datos de usuario (correo, contraseña) 2. Firebase valida los datos de usuario (correo, contraseña) 3. Firebase registra los datos de usuario 4. Firebase envía una respuesta al sistema conteniendo una clase User de Firebase
Flujo Alternativo:	2.a.1. La validación falla al ya existir un usuario previamente registrado con el correo recibido 2.a.2. Firebase envía una respuesta al sistema conteniendo el error 2.b.1. La validación falla por la contraseña no cumplir con los requisitos mínimos 2.b.2. Firebase envía una respuesta al sistema conteniendo el error
Prioridad:	Alta
Otra información:	Requisitos de Contraseña: • Mínimo 12 caractéres • Al menos una mayúscula • Al menos una minúscula • Al menos un número • Al menos un carácter especial
Suposiciones:	

ID:	CU-3
Nombre:	Login
Actor primario:	Usuario
Actores secundarios:	Firebase
Descripción:	Los usuarios pueden acceder a su cuenta con sus credenciales
Evento activador:	El usuario hace click en el botón de “Login”
Precondiciones:	El usuario debe haber creado la cuenta previamente y esta seguir existiendo
Postcondiciones:	El usuario se ha logueado en el sistema
Flujo Normal:	1. El usuario accede a la pantalla de Login 2. El usuario ingresa sus credenciales (correo y contraseña) 3. El usuario pulsa el botón de “Login” 4. El sistema envía los datos ingresados a Firebase 5. <<Include>> CU-4 Autenticar 6. El sistema recibe la respuesta de Firebase 7. El sistema carga los datos del usuario 8. El sistema carga la pantalla de Dashboard
Flujo Alternativo:	6.a.1. El sistema recibe una respuesta de error de Firebase 6.a.2. El sistema muestra el error en la pantalla de Login
Prioridad:	Alta
Otra información:	
Suposiciones:	

ID:	CU-4
Nombre:	Autenticar
Actor primario:	Firebase
Actores secundarios:	
Descripción:	Firebase verifica las credenciales que recibe de un intento de login en el sistema
Evento activador:	CU-3 Login
Precondiciones:	El sistema ha enviado los datos requeridos a Firebase (correo y contraseña)
Postcondiciones:	Firebase ha enviado una respuesta a el sistema
Flujo Normal:	1. Firebase recibe los datos de usuario (correo, contraseña) 2. Firebase valida los datos de usuario (correo, contraseña) 3. Firebase envía una respuesta al sistema conteniendo el ID Token JWT de la sesión y el Refresh Token
Flujo Alternativo:	2.a.1. La validación falla al no existir un usuario previamente registrado el correo recibido 2.a.2. Firebase envía una respuesta al sistema conteniendo el error 2.b.1. La validación falla al no coincidir la contraseña recibida con el correo recibido 2.b.2. Firebase envía una respuesta al sistema conteniendo el error
Prioridad:	Alta
Otra información:	
Suposiciones:	

ID:	CU-5
Nombre:	Recuperar Contraseña
Actor primario:	Usuario
Actores secundarios:	Firebase
Descripción:	Los usuarios pueden restablecer su contraseña en caso de no recordarla
Evento activador:	El usuario pulsa el botón de "¿Has olvidado tu contraseña?" en la pantalla de Login
Precondiciones:	El usuario debe estar en la pantalla de Login
Postcondiciones:	La contraseña se ha actualizado a la nueva introducida por el usuario
Flujo Normal:	1. El usuario pulsa el botón de "¿Has olvidado tu contraseña?" en la pantalla de Login 2. El sistema muestra la pantalla de restablecer contraseña 3. El usuario ingresa su correo electrónico 4. El usuario pulsa el botón de “Enviar enlace” 5. El sistema envía la petición a Firebase 6. <<Include>> CU-6 Enviar Correo de Recuperación
Flujo Alternativo:	4.a.1. El usuario pulsa el botón de “Cancelar” 4.a.2. El sistema vuelve a la pantalla de Login
Prioridad:	Media
Otra información:	
Suposiciones:	

ID:	CU-6
Nombre:	Enviar Correo de Recuperación
Actor primario:	Firebase
Actores secundarios:	
Descripción:	Firebase envía un email al correo recibido con un enlace para restablecer la contraseña asociada a dicho correo
Evento activador:	CU-5 Recuperar Contraseña
Precondiciones:	El sistema ha enviado los datos requeridos a Firebase (correo)
Postcondiciones:	Firebase ha enviado un email al correo recibido con un enlace de restablecimiento de contraseña
Flujo Normal:	1. Firebase recibe el correo de usuario 2. Firebase valida el correo de usuario 3. Firebase envía un email al correo recibido con un enlace para actualizar la contraseña asociada a dicho correo
Flujo Alternativo:	2.a.1 La validación falla debido a que el correo recibido no está previamente registrado 2.a.2 Firebase no envía un email
Prioridad:	Media
Otra información:	Correo: cuenta de correo electrónico (aaa@gmail.com) Email: mensaje electrónico
Suposiciones:	Se asume que tras Firebase haber enviado el email de recuperación, y el usuario haber recibido dicho email, el usuario usará el enlace de recuperación para restablecer la contraseña asociada al correo

ID:	CU-7
Nombre:	Ver Datos de Usuario
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden acceder a sus datos de usuario
Evento activador:	Pulsar icono de Perfil
Precondiciones:	El usuario está logueado y en el Dashboard
Postcondiciones:	El usuario se encuentra en la pantalla de Perfil
Flujo Normal:	1. El usuario pulsa el icono de Perfil 2. El sistema carga la pantalla de Perfil
Flujo Alternativo:	
Prioridad:	Media
Otra información:	
Suposiciones:	

ID:	CU-8
Nombre:	Eliminar Cuenta de Usuario
Actor primario:	Usuario
Actores secundarios:	Firebase
Descripción:	Los usuarios pueden eliminar su cuenta y con ella todos sus datos correspondientes
Evento activador:	El usuario pulsa el botón de “Eliminar cuenta”
Precondiciones:	El usuario se encuentra en la pantalla de Perfil
Postcondiciones:	La cuenta del usuario y todos sus datos han sido eliminados y el usuario se encuentra en la pantalla de Login
Flujo Normal:	1. El usuario pulsa el botón de “Eliminar Cuenta” 2. El sistema una pantalla de confirmación 3. El usuario pulsa el botón de “Eliminar cuenta” 4. El sistema solicita a Firebase eliminar la cuenta del correo del usuario 5. <<Include>> CU-9 Eliminar Registro de Usuario 6. El sistema recibe la respuesta de Firebase 7. El sistema borra los datos del usuario 8. El sistema desloguea al usuario 9. El sistema carga la pantalla de Login
Flujo Alternativo:	3.a.1. El usuario pulsa el botón “Cancelar” 3.a.2. El sistema cierra la pantalla de confirmación 5.a.1. El sistema recibe un error de Firebase 5.a.2. El sistema muestra el error en la pantalla de Perfil
Prioridad:	Media
Otra información:	
Suposiciones:	

ID:	CU-9
Nombre:	Eliminar Registro de Usuario
Actor primario:	Firebase
Actores secundarios:	
Descripción:	Firebase elimina el registro del usuario
Evento activador:	CU-8 Eliminar Cuenta de Usuario
Precondiciones:	El sistema le ha enviado a Firebase
Postcondiciones:	Firebase ha enviado una respuesta al sistema
Flujo Normal:	1. Firebase recibe la petición 2. Firebase valida los datos de usuario 3. Firebase elimina el registro del usuario 4. Firebase envía una respuesta al sistema
Flujo Alternativo:	
Prioridad:	Media
Otra información:	
Suposiciones:	

ID:	CU-10
Nombre:	Editar Datos de Usuario
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden editar sus datos del sistema
Evento activador:	El usuario pulsa el botón de “Editar datos”
Precondiciones:	El usuario se encuentra en la pantalla de Perfil
Postcondiciones:	Los datos del usuario han sido modificados en la base de datos del sistema
Flujo Normal:	1. El usuario pulsa el botón de “Editar datos” 2. El sistema carga la pantalla de Editar datos 3. El usuario modifica los campos que quiera editar 4. El usuario le da al botón “Guardar” 5. El sistema guarda las modificaciones en su base de datos 6. El sistema carga los datos del usuario 7. El sistema carga la pantalla de Perfil
Flujo Alternativo:	4.a.1. El usuario pulsa el botón “Cancelar” 4.a.2. El sistema vuelve a la pantalla de Perfil 5.a.1. El sistema no puede guardar las modificaciones debido a un error con la base de datos 5.a.2. El sistema muestra el error por pantalla
Prioridad:	Media
Otra información:	
Suposiciones:	

ID:	CU-11
Nombre:	Crear Meta
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden crear metas en el sistema
Evento activador:	El usuario pulsa el botón de “Crear meta”
Precondiciones:	El usuario está en la pantalla de Dashboard
Postcondiciones:	La meta ha sido creada
Flujo Normal:	1. El usuario pulsa el botón de “Crear Meta” 2. El sistema muestra la pantalla de creación de metas 3. El usuario rellena los datos requeridos para la creación de su meta 4. El usuario pulsa el botón de “Crear” 5. El sistema almacena la meta en su base de datos 6. El sistema carga los datos del usuario 7. El usuario carga la pantalla de Dashboard
Flujo Alternativo:	5.a.1. El sistema detecta algún campo vacío que no puede estar vacío 5.a.2. El sistema muestra un mensaje por pantalla recordándole al usuario que debe rellenar todos los datos requeridos 5.a.3. Vuelve al punto 2 5.b.1. El sistema no puede almacenar la meta en su base de datos debido a un error 5.b.2. El sistema muestra el error por pantalla
Prioridad:	Alta
Otra información:	
Suposiciones:	

ID:	CU-12
Nombre:	Ver Meta
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden ver en detalle sus metas
Evento activador:	El usuario pulsa en la vista previa de una meta
Precondiciones:	El usuario está en la pantalla de Dashboard y tiene metas previamente creadas
Postcondiciones:	El sistema muestra los datos de una meta en detalle
Flujo Normal:	1. El usuario pulsa en la meta 2. El sistema carga todos los datos de la meta, incluidos registros 3. El sistema actualiza la pantalla de Dashboard con todos los datos de la meta, incluyendo registros y cargando grafico/calendario
Flujo Alternativo:	2.a.1. El sistema no puede cargar las metas del usuario debido a un error con la base de datos 2.a.2. El sistema muestra el error por pantalla
Prioridad:	Alta
Otra información:	
Suposiciones:	

ID:	CU-13
Nombre:	Eliminar Registro
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden eliminar registros de metas
Evento activador:	El usuario pulsa el botón “Eliminar” de un registro
Precondiciones:	CU-11 Ver Meta
Postcondiciones:	El registro ha sido eliminado
Flujo Normal:	1. El usuario pulsa el botón “Eliminar” de un registro 2. El sistema elimina el registro de la meta de su base de datos 3. El sistema actualiza la pantalla de Dashboard
Flujo Alternativo:	2.a.1. El sistema no puede eliminar el registro debido a un error con la base de datos 2.a.2. El sistema muestra el error por pantalla
Prioridad:	Media
Otra información:	
Suposiciones:	

ID:	CU-14
Nombre:	Crear Registro
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden crear registros de sus metas
Evento activador:	El usuario pulsa el botón de “Registro” de una meta
Precondiciones:	El usuario está en la pantalla de Dashboard y tiene metas previamente creadas
Postcondiciones:	El registro ha sido creado
Flujo Normal:	1. El usuario pulsa el botón de “Registro” de una meta 2. El sistema carga la pantalla de “Registro” 3. El usuario introduce los datos requeridos para la creación del registro 4. El usuario pulsa el botón de “Guardar” 5. El sistema crea el registro para la meta en su base de datos 6. El sistema carga los datos del usuario 7. El sistema carga la pantalla de Dashboard
Flujo Alternativo:	5.a.1. El sistema detecta algún campo vacío que no puede estar vacío 5.a.2. El sistema muestra un mensaje por pantalla recordándole al usuario que debe rellenar todos los datos requeridos 5.a.3. Vuelve al punto 3 5.b.1. El sistema no puede almacenar el registro en su base de datos debido a un error 5.b.2. El sistema muestra el error en la pantalla de Dashboard
Prioridad:	Alta
Otra información:	
Suposiciones:	

ID:	CU-15
Nombre:	Finalizar Meta
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden finalizar sus metas
Evento activador:	El usuario pulsa el botón “Finalizar” de una meta
Precondiciones:	El usuario está en la pantalla de Dashboard y tiene metas previamente creadas
Postcondiciones:	El estado de la meta es finalizada
Flujo Normal:	1. El usuario pulsa el botón “Finalizar” de una meta 2. El sistema carga la pantalla de confirmación 3. El usuario pulsa el botón “Aceptar” 4. El sistema modifica el estado de la meta a finalizada en su base de datos 5. El sistema carga los datos del usuario 6. El sistema muestra la pantalla de Dashboard
Flujo Alternativo:	3.a.1. El usuario pulsa el botón “Cancelar” 3.a.2. El sistema cierra la ventana de confirmación 4.a.1. El sistema no puede cambiar el estado de la meta a finalizada en su base de datos debido a un error 4.a.2. El sistema muestra el error por pantalla
Prioridad:	Media
Otra información:	Cambiar el estado de una meta a finalizada hará que se considere como acabada/abandonada en el sistema y se cierre la posibilidad de crear registros de la misma y editarla
Suposiciones:	

ID:	CU-16
Nombre:	Editar Meta
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden editar sus metas
Evento activador:	El usuario pulsa el botón “Editar” de una de sus metas
Precondiciones:	El usuario está en la pantalla de Dashboard y tiene metas previamente creadas
Postcondiciones:	Los cambios sobre la meta han sido almacenados en la base de datos
Flujo Normal:	1. El usuario pulsa el botón “Editar” en una meta 2. El sistema carga la pantalla de “Editar Meta” 3. El usuario modifica los campos que desea editar 4. El usuario pulsa el botón “Guardar” 5. El sistema actualiza los datos de la meta en su base de datos 6. El sistema carga los datos de usuario 7. El sistema carga la pantalla de Dashboard
Flujo Alternativo:	4.a.1. El usuario pulsa el botón “Cancelar” 4.a.2. El sistema vuelve a la pantalla de Dashboard 5.a.1. El sistema detecta algún campo vacío que no puede estar vacío 5.a.2. El sistema muestra un mensaje por pantalla recordándole al usuario que debe rellenar todos los datos requeridos 5.a.3. Vuelve al punto 3 5.b.1. El sistema no puede actualizar la meta en su base de datos debido a un error 5.b .2. El sistema muestra el error por pantalla
Prioridad:	Media
Otra información:	
Suposiciones:	

ID:	CU-17
Nombre:	Eliminar Meta
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden eliminar sus metas
Evento activador:	El usuario pulsa el botón “Eliminar” de una de sus metas
Precondiciones:	El usuario está en la pantalla de Dashboard y tiene metas previamente creadas
Postcondiciones:	La meta ha sido eliminada
Flujo Normal:	1. El usuario pulsa el botón “Eliminar” de una meta 2. El sistema carga la pantalla de confirmación 3. El usuario pulsa el botón “Eliminar” 4. El sistema elimina la meta y todos sus registros de su base de datos 5. El sistema carga los datos del usuario 6. El sistema carga la pantalla de Dashboard
Flujo Alternativo:	3.a.1. El usuario pulsa el botón de “Cancelar” 3.a.2. El sistema cierra la ventana de confirmación 4.a.1. El sistema no puede eliminar la meta en su base de datos debido a un error 4.a.2. El sistema muestra el error por pantalla
Prioridad:	Media
Otra información:	
Suposiciones:	

ID:	CU-18
Nombre:	Ver Tutorial
Actor primario:	Usuario
Actores secundarios:	
Descripción:	Los usuarios pueden ver un tutorial sobre las funcionalidades del dashboard
Evento activador:	El usuario pulsa el icono del tutorial
Precondiciones:	El usuario está en la pantalla de Dashboard
Postcondiciones:	El usuario ha visto el tutorial y la meta de ejemplo ha sido creada en la base de datos del sistema
Flujo Normal:	1. El usuario pulsa el icono del tutorial 2. El sistema abre una pantalla de confirmación 3. El usuario pulsa el botón “Empezar” 4. El sistema crea una meta y sus registros de ejemplo en la base de datos 5. El sistema muestra una ventana de tutorial sobre una funcionalidad del dashboard 6. El usuario pulsa el botón “Siguiente” 7. Vuelta al 5 (hasta que no queden funcionalidades) 8. El sistema cierra el tutorial
Flujo Alternativo:	3.a.1 El usuario pulsa el botón “Cancelar” 3.a.2 El sistema cierra la pantalla de confirmación 5.a.1. El sistema no puede crear la meta y sus registros en su base de datos debido a un error 5.a.2. El sistema muestra el error por pantalla 6.a.1. El usuario pulsa en el botón “Anterior” 6.a.2. Vuelta al 5 de la funcionalidad anterior a la actual 6.b.1. El usuario pulsa fuera de la ventana del tutorial 6.b.2. El sistema cierra el tutorial
Prioridad:	Media
Otra información:	
Suposiciones:	

Diseño de Endpoints Inicial de GoLife

Endpoints de Gestión de Usuarios:

1. Crear Usuario

- **Método HTTP:** POST
- **Ruta:** /api/usuarios
- **Descripción:** Permite registrar un nuevo usuario en nuestra base de datos. El frontend envia el token de sesión al backend para el registro del usuario.
- **Cuerpo de la solicitud:**

```
{
  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6Ijk3ZTkwZ...<token>"
}
```

- **Respuesta de éxito:**

```
{
  "status": "success",
  "message": "Cuenta creada exitosamente",
}
```

- **Respuestas de error:**
  - **400 Bad Request**

    ```
    {
      "status": "error",
      "message": "Datos inválidos o faltantes"
    }
    ```

    Se devuelve cuando la solicitud no contiene la información mínima requerida o el token proporcionado es inválido/formato incorrecto.

  - **401 Unathorized**

    ```
    {
      "status": "error",
      "message": "Token de sesión inválido o expirado"
    }
    ```

    Se devuelve cuando el token de sesión (idToken) no puede ser validado correctamente por el backend.

  - **500 Internal Server Error**

    ```
    {
      "status": "error",
      "message": "Error interno en el servidor"
    }
    ```

    Se devuelve cuando ocurre un error inesperado al procesar la solicitud, por ejemplo, un fallo al conectar con la base de datos.

1. Eliminar Usuario

- **Método HTTP:** DELETE

- **Ruta:** /api/usuarios
- **Descripción:** Permite eliminar la cuenta de un usuario específico. Para garantizar la seguridad, se requiere el token JWT de la sesión.
- **Cuerpo de la solicitud:**

```
{
  "idToken": "eyJhbGciOiJSUzI1NiIsImtpZCI6Ijk3ZTkwZ...<token>"
}
```

- **Respuesta de éxito:**

```
{
  "status": "success",
  "message": "Cuenta creada exitosamente",
}
```

- **Respuestas de error:**
  - **400 Bad Request**

```
{
  "status": "error",
  "message": "Datos inválidos o faltantes"
}
```

Se devuelve cuando la solicitud no contiene la información mínima requerida o el token proporcionado es inválido/formato incorrecto.

  - **401 Unathorized**

```
{
  "status": "error",
  "message": "Token de sesión inválido o expirado"
}
```

Se devuelve cuando el token de sesión (idToken) no puede ser validado correctamente por el backend.

  - **500 Internal Server Error**

```
{
  "status": "error",
  "message": "Error interno en el servidor"
}
```

Se devuelve cuando ocurre un error inesperado al procesar la solicitud, por ejemplo, un fallo al conectar con la base de datos.

Endpoints de Gestión de Metas:

1. Crear Meta

- **Método HTTP:** POST
- **Ruta:** /api/metas
- **Descripción:** Permite al usuario autenticado crear una nueva meta.
- **Cabeceras necesarias:**

- Authorization: Bearer <idToken>

- **Cuerpo de la solicitud:**

```
{
 "nombre": "Correr una maratón",
 "descripcion": "Entrenar para la maratón local",
 "tipoRegistro": "RegistroBool",  // o "RegistroNum"
 "duracion": {
  "valor": 6,
  "unidad": "meses"
 }
“valorObjetivo”: 50,
“unidad”: “Km”
}
```

- **Respuesta de éxito:**

```
{
 "status": "success",
 "message": "Meta creada exitosamente",
}
```

- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

2. Editar Meta

- **Método HTTP:** PUT
- **Ruta:** /api/metas/{metaId}
- **Descripción:** Permite al usuario autenticado editar los detalles de una meta existente.
- **Cabeceras necesarias:**
  - Authorization: Bearer <idToken>
- **Cuerpo de la solicitud:**

```
{
 "nombre": "Correr una maratón actualizada",
 "descripcion": "Nuevo plan de entrenamiento",
 "duracion": {
  "valor": 3,
  "unidad": "semanas"
 },
“valorObjetivo”: 40,
“unidad”: “Km”
}
```

- **Respuesta de éxito:**

```
{
```

```
  "status": "success",
  "message": "Meta actualizada exitosamente"
}
```

- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

3. Eliminar Meta

- **Método HTTP:** DELETE
- **Ruta:** /api/metas/{metaId}
- **Descripción:** Elimina la meta identificada para el usuario autenticado.
- **Cabeceras necesarias:**
  - Authorization: Bearer <idToken>
- **Respuesta de éxito:**

```
{
  "status": "success",
  "message": "Meta eliminada exitosamente"
}
```

- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

4. Finalizar Meta

- **Método HTTP:** POST
- **Ruta:** /api/metas/{metaId}/finalizar
- **Descripción:** Finaliza la meta identificada para el usuario identificado.
- **Cabeceras necesarias:**
  - Authorization: Bearer <idToken>
- **Respuesta de éxito:**

```
{
  "status": "success",
  "message": "Meta finalizada exitosamente"
}
```

- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

5. Crear Registro de Meta

- **Método HTTP:** POST
- **Ruta:** /api/metas/{metaId}/registros

- **Descripción:** Permite registrar un avance en una meta.
- **Cabeceras necesarias:**
    - Authorization: Bearer <idToken>
- **Cuerpo de la solicitud:**

  Ejemplo para registro numérico:

```
{
 "valor": 5.5,
 "fecha": "2025-03-14"
}
```

  Ejemplo para un registro booleano:

```
{
 "completado": true,
 "fecha": "2025-03-14"
}
```

- **Respuesta de éxito:**

```
{
 "status": "success",
 "message": "Registro de meta creado exitosamente"
}
```

- **Respuestas de error:**
    - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

6. Obtener Datos de una Meta Específica

- **Método HTTP:** GET
- **Ruta:** /api/metas/{metaId}
- **Descripción:** Devuelve la información completa de la meta identificada por {metaId} para el usuario autenticado.
- **Cabeceras necesarias:**
    - Authorization: Bearer <idToken>
- **Respuesta de éxito:**

```
{
 "status": "success",
 "data": {
  "metaId": "12345",
  "nombre": "Correr una maratón",
  "fecha": "2025-03-14T10:00:00Z",
  "activa": true,
```

```
  "descripcion": "Entrenar para la maratón local",
  "tipoRegistro": " RegistroNum ",
  "duracion": {
   "valor": 6,
   "unidad": "meses"
  },
  "valorObjetivo": 40,
  "unidad": "Km"
 }
}
```


- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

7. Obtener Registros de una Meta Específica

- **Método HTTP:** GET
- **Ruta:** /api/metas/{metaId}/registros
- **Descripción:** Devuelve la lista de registros asociados a la meta identificada por {metaId} para el usuario autenticado.
- **Cabeceras necesarias:**
  - Authorization: Bearer <idToken>
- **Respuesta de éxito:**
  Ejemplo para registro numérico


```
{
 "status": "success",
 "data": [
  {
   "valor": 3.0,
   "fecha": "2025-03-01T09:00:00Z"
  },
  {
   "valor": 5.5,
   "fecha": "2025-03-14T10:00:00Z"
  }
 ]
}
```


Ejemplo para registro booleano


```
{
 "status": "success",
 "data": [
  {
   "completado": true,
   "fecha": "2025-02-28T15:00:00Z"
  }
 ]
}
```

- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

8. Obtener Todas las Metas del Usuario

- **Método HTTP:** GET
- **Ruta:** /api/metas
- **Descripción:** Devuelve la lista de metas creadas por el usuario autenticado.
- **Cabeceras necesarias:**
  - Authorization: Bearer <idToken>
- **Respuesta de éxito:**

```
{
 "status": "success",
 "data": [
  {
   "metaId": "12345",
   "nombre": "Correr una maratón",
   "fecha": "2025-03-14T10:00:00Z",
   "activa": true,
   "descripcion": "Entrenar para la maratón local",
   "tipoRegistro": ""RegistroNum",
   "duracion": {
    "valor": 6,
    "unidad": "meses"
   },
   “valorObjetivo”: 40,
   “unidad”: “Km”
  },
  {
   "metaId": "12346",
   "nombre": "Leer 12 libros",
   "fecha": "2025-03-14T10:00:00Z",
   "activa": false,
   "descripcion": "Leer un libro por mes",
   "tipoRegistro": ""RegistroBool",
   "duracion": null
   “valorObjetivo”: null,
   “unidad”: “Libros”
  }
 ]
}
```

- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

9. Obtener Todas las Metas y sus Registros del Usuario

- **Método HTTP:** GET
- **Ruta:** /api/metas/full
- **Descripción:** Devuelve la lista completa de metas junto con sus registros asociados para el usuario autenticado.
- **Cabeceras necesarias:**
    - Authorization: Bearer <idToken>
- **Respuesta de éxito:**


```
{
 "status": "success",
 "data": [
  {
   "metaId": "12345",
   "nombre": "Correr una maratón",
   "fecha": "2025-03-14T10:00:00Z",
   "activa": true,
   "descripcion": "Entrenar para la maratón local",
   "tipoRegistro": " RegistroNum",
   "duracion": {
    "valor": 6,
    "unidad": "meses"
   },
   “valorObjetivo”: 40,
   “unidad”: “Km”,
   "registros": [
    {
     "valor": 3.0,
     "fecha": "2025-03-01T09:00:00Z"
    },
    {
     "valor": 5.5,
     "fecha": "2025-03-14T10:00:00Z"
    }
   ]
  },
  {
   "metaId": "12346",
   "nombre": "Leer 12 libros",
   "fecha": "2025-03-14T10:00:00Z",
   "activa": false,
   "descripcion": "Leer un libro por mes",
   "tipoRegistro": " RegistroBool ",
   "duracion": null,
   “valorObjetivo”: null,
   “unidad”: “Libros”,
   "registros": [
    {
     "completado": true,
```

```
      "fecha": "2025-02-28T15:00:00Z"
     }
    ]
   }
  ]
}
```


- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

10. Obtener Todas las Metas Activas del Usuario

- **Método HTTP:** GET
- **Ruta:** /api/metas/activas
- **Descripción:** Devuelve la lista de metas activas (no finalizadas) creadas por el usuario autenticado.
- **Cabeceras necesarias:**
  - Authorization: Bearer <idToken>
- **Respuesta de éxito:**


```
{
 "status": "success",
 "data": [
  {
   "metaId": "12345",
   "nombre": "Correr una maratón",
   "fecha": "2025-03-14T10:00:00Z",
   "activa": true,
   "descripcion": "Entrenar para la maratón local",
   "tipoRegistro": ""RegistroNum",
   "duracion": {
    "valor": 6,
    "unidad": "meses"
   },
   “valorObjetivo”: 40,
   “unidad”: “Km”
  }
 ]
}
```


- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

11. Obtener Todas las Metas Activas y sus Registros del Usuario

- **Método HTTP:** GET
- **Ruta:** /api/metas/activas/full

- **Descripción:** Devuelve la lista completa de metas activas (no finalizadas) junto con sus registros asociados para el usuario autenticado.
- **Cabeceras necesarias:**
    - Authorization: Bearer <idToken>
- **Respuesta de éxito:**


```
{
 "status": "success",
 "data": [
  {
   "metaId": "12345",
   "nombre": "Correr una maratón",
   "fecha": "2025-03-14T10:00:00Z",
   "activa": true,
   "descripcion": "Entrenar para la maratón local",
   "tipoRegistro": " RegistroNum",
   "duracion": {
    "valor": 6,
    "unidad": "meses"
   },
   “valorObjetivo”: 40,
   “unidad”: “Km”,
   "registros": [
    {
     "valor": 3.0,
     "fecha": "2025-03-01T09:00:00Z"
    },
    {
     "valor": 5.5,
     "fecha": "2025-03-14T10:00:00Z"
    }
   ]
  }
 ]
}
```


- **Respuestas de error:**
    - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

12. Obtener Todas las Metas Inactivas del Usuario

- **Método HTTP:** GET
- **Ruta:** /api/metas/inactivas
- **Descripción:** Devuelve la lista de metas inactivas (finalizadas) creadas por el usuario autenticado.
- **Cabeceras necesarias:**
    - Authorization: Bearer <idToken>

- **Respuesta de éxito:**


```
{
 "status": "success",
 "data": [
  {
   "metaId": "12346",
   "nombre": "Leer 12 libros",
   "fecha": "2025-03-14T10:00:00Z",
   "activa": false,
   "descripcion": "Leer un libro por mes",
   "tipoRegistro": ""RegistroBool",
   "duracion": null
   “valorObjetivo”: null,
   “unidad”: “Libros”
  }
 ]
}
```


- **Respuestas de error:**
  - Mismas respuestas de error que en Endpoints de Gestión de Usuarios

Cuestionario de Evaluación de GoLife

¡Hola! 👋

Antes que nada, **muchas gracias por dedicar tu tiempo a probar GoLife y ayudarnos en nuestro Trabajo de Fin de Grado**. Tu opinión es fundamental para validar si nuestra aplicación cumple con los objetivos planteados y para identificar cómo podemos mejorarla.

GoLife es una aplicación web para la gestión personal. Diseñada para ser genérica y poder seguir cualquiera de tus metas personales, registrar actualizaciones periódicas y visualizarlas en un panel de control (dashboard).

👉 **Si todavía no has probado GoLife, puedes hacerlo aquí:**

- https://golife-deployment.web.app/

Te pedimos que respondas con total honestidad:

- Si algo no te ha gustado, tu evaluación sigue siendo igual de valiosa.
- Si consideras que la aplicación no es para ti, también es un **feedback útil** para nuestro proyecto

El cuestionario es **anónimo**, toma aproximadamente **2-3 minutos** y consta de preguntas rápidas junto a algunas abiertas para que compartas tus impresiones. Estará abierto desde el **lunes 8 de septiembre** hasta el **viernes 12 de septiembre.**

¡Gracias de nuevo por tu ayuda! 🙌

* Indica que la pregunta es obligatoria

1. ¿Cuál es tu rango de edad? *

- ◯ 18–24 años
- ◯ 25–34 años
- ◯ 35–49 años
- ◯ 50–64 años
- ◯ 65 o más años

2. ¿Pudiste usar las funciones principales de GoLife (crear, hacer registros, editar, actualizar y eliminar metas) sin ayuda? *

- ◯ Sí, todas sin ayuda
- ◯ Sí, la mayoría, pero con alguna dificultad
- ◯ No, necesité ayuda

3. En general, manejar las funciones principales de GoLife (crear, hacer registros, editar, actualizar y eliminar metas) me resultó... *

	1	2	3	4	5	
Muy Difícil	◯	◯	◯	◯	◯	Muy Fácil

4. GoLife me resultó intuitivo desde un inicio *

	1	2	3	4	5	
Muy en Desacuerdo	◯	◯	◯	◯	◯	Muy de Acuerdo

5. ¿Viste el tutorial de GoLife? *

- ◯ Sí, lo vi completo antes de crear metas
- ◯ Si, pero lo vi después de crear metas
- ◯ No, no lo vi en ningún momento

6. Si viste el tutorial, ¿por qué lo hiciste?

- ◯ Quería tener una guía antes de empezar a crear metas
- ◯ Lo vi para aclarar dudas después de empezar a crear metas
- ◯ No entendía la aplicación web en un inicio
- ◯ No encontré el tutorial
- ◯ Otro:

7. ¿Crees que GoLife puede servir para distintos tipos de metas (por ejemplo: estudios, deporte, finanzas, hábitos, etc.)? *

- ◯ Sí, sin duda
- ◯ En parte, pero con limitaciones
- ◯ No, creo que está muy enfocada a un tipo concreto de metas

8. ¿Para qué tipo de metas usaste GoLife? *

Tu respuesta

9. ¿Cómo valorarías la velocidad de carga de la aplicación web en general? *

	1	2	3	4	5	
Muy Lenta	◯	◯	◯	◯	◯	Muy Rápida

10. ¿Has experimentado tiempos de espera largos o bloqueos al usar la aplicación? *

- ○ No
- ○ Sí, pero muy pocas veces
- ○ Sí, con frecuencia

Enviar

Borrar formulario