Universidad Zaragoza

Trabajo Fin de Grado

Grado en Ingeniería Informática

Desarrollo del sistema base y módulo de gestión de entrenamientos para una aplicación móvil de escalada

Development of the core system and training management module for a climbing mobile application

Autor

Diego Lasheras Blasco

Director

Ricardo Julio Rodríguez Fernández

Co-director

Rafael Galán Rodríguez

ESCUELA DE INGENIERÍA Y ARQUITECTURA
Septiembre de 2026

Resumen

Bulderland es un centro de escalada que utiliza medios de comunicación genéricos para organizar entrenamientos con sus atletas, lo que hace difícil la colaboración entre entrenadores y atletas. Este trabajo describe el diseño, la implementación y el despliegue de la aplicación móvil, la cual automatiza todo el proceso con el fin de mejorar esta colaboración. La solución se implementa siguiendo la arquitectura cliente-servidor. En la parte del servidor se usa Node.js y Express, exponiendo una API REST. El cliente móvil multiplataforma se implementa usando React Native y Expo.

El sistema utiliza un modelo de roles múltiples (atleta, entrenador y administrador) con autenticación usando *tokens* JWT y autorización basada en roles. Los entrenadores crean y distribuyen planes de entrenamiento a los atletas, que se registran y completan sus resultados. Los entrenadores y los atletas pueden comunicarse entre sí en tiempo real, así como recibir notificaciones cada vez que ocurre algo importante. Los planes de entrenamiento se almacenan en MongoDB. El acceso al contenido de los entrenamientos está limitado por un abono, que es activado por el administrador y expira automáticamente después de un periodo de tiempo definido, mediante una tarea programada.

El sistema se despliega en un entorno de producción en el servidor de la universidad usando contenedores Docker. Así, el resultado es una aplicación funcional, la cual automatiza el proceso que antes se realizaba manualmente. Es fácil de manejar y aborda una de las lagunas no cubiertas por las soluciones del mercado analizadas: la autoría de los planes de entrenamiento sigue en manos del personal técnico del centro.

Abstract

Bulderland is a climbing center that uses generic communication tools to organize training with its athletes, which makes collaboration between coaches and athletes difficult. This thesis describes the design, implementation and deployment of the mobile application, which automates the whole process in order to improve this collaboration. The solution is implemented following a client-server architecture. On the server side, Node.js and Express are used, exposing a REST API. The cross-platform mobile client is implemented using React Native and Expo.

The system uses a multi-role model (athlete, coach and administrator) with authentication using JWT tokens and role-based authorization. Coaches create and assign training plans to athletes, who enroll in them and record their results. Coaches and athletes can communicate with each other in real time, as well as receive notifications whenever something important happens. Training plans are stored in MongoDB. Access to training content is restricted by a membership, which is activated by the administrator and automatically expires after a defined period of time, through a scheduled task.

The system is deployed in a production environment on the university server using Docker containers. Thus, the result is a working application that automates the process that was previously carried out manually. It is easy to manage and addresses one of the gaps not covered by the market solutions analyzed: authorship of the training plans remains with the center's own coaching staff.

Índice general

Índice de figuras **V**

Índice de tablas **VI**

1. Introducción **1**

1.1. Contexto y motivación 1

1.2. Objetivos del proyecto 1

1.3. Estructura del documento 2

2. Estado del arte **3**

3. Análisis del sistema **5**

3.1. Análisis de requisitos 5

3.2. Definición de los roles de usuario 8

3.3. Casos de uso 9

4. Diseño e implementación **12**

4.1. Selección de tecnologías 12

4.2. Arquitectura de la solución 13

4.2.1. Comunicación en tiempo real 15

4.2.2. Control de acceso por abono 16

4.3. Integración de servicios externos 17

4.4. Diseño de la base de datos 19

4.5. Diseño de la interfaz de usuario 21

5. Pruebas y despliegue **23**

5.1. Enfoque de pruebas 23

5.2. Entorno de despliegue y producción 25

5.3. Organización del código 26

6. Conclusiones y trabajo futuro **28**

Declaración de uso responsable de IA generativa **29**

Bibliografía **30**

A. Planificación y distribución temporal **32**

Índice de figuras

3.1. Diagrama UML de casos de uso del sistema. 10

4.1. Arquitectura por capas del *backend*: cadena de *middleware*, rutas, controladores y modelos para los nueve recursos de la API, canal de tiempo real y servicios externos. 14
4.2. Diagrama UML de secuencia del chat en tiempo real: apertura de sesión de los dos participantes de una sala y difusión de un mensaje. Las puntas de flecha rellenas representan llamadas síncronas, las abiertas mensajes asíncronos y las líneas discontinuas, respuestas. 16
4.3. Diagrama UML de secuencia del ciclo de vida del abono: inicio de sesión, solicitud del atleta, aprobación del administrador, control de acceso en cada petición y caducidad automática. 17
4.4. Diagrama UML de secuencia de una notificación *push*, desde el evento de dominio hasta su entrega en el dispositivo. 18
4.5. Diagrama entidad–relación de la base de datos (nueve colecciones). . . 20
4.6. Mapa de navegación desde el inicio de sesión hasta una pantalla de segundo nivel, según el rol del usuario. 22

5.1. Cobertura del conjunto de pruebas de Jest en el *backend*, por directorio. 24
5.2. Informe de cobertura del conjunto de pruebas de React Testing Library en el cliente móvil, por directorio. 24
5.3. Diagrama UML de despliegue de la aplicación en producción sobre el servidor `crono.unizar.es` de la Universidad de Zaragoza. 25

A.1. Diagrama de Gantt del cronograma de desarrollo del proyecto (350 horas). 33

Índice de tablas

2.1. Comparativa de las soluciones existentes frente a los requisitos del proyecto. 4

3.1. Requisitos funcionales del módulo de cuentas y autenticación. 6
3.2. Requisitos funcionales del módulo de planes y sesiones. 6
3.3. Requisitos funcionales del módulo de ejecución y estadísticas. 7
3.4. Requisitos funcionales del módulo de comunicación y notificaciones. . . 7
3.5. Requisitos no funcionales del sistema. 8
3.6. Catálogo de casos de uso del sistema. 11

5.1. Comparativa global de las métricas de cobertura entre el *backend* y el cliente móvil. 25

A.1. Distribución estimada del esfuerzo por tipo de actividad. 32

Capítulo 1

Introducción

1.1. Contexto y motivación

En el ámbito deportivo la planificación y el seguimiento de la carga de entrenamiento condicionan tanto el progreso como la prevención de lesiones [1]; ajustar las rutinas a la evolución de cada atleta exige una comunicación constante con su entrenador [2, 3].

El presente trabajo de fin de grado surge de una necesidad planteada por la empresa de escalada Bulderland. Antes de este proyecto, la gestión de los planes de entrenamiento, la comunicación de las rutinas y las anotaciones sobre el rendimiento o las sensaciones de los atletas dependían de métodos no estandarizados, como herramientas genéricas de mensajería. Estas herramientas dificultaban el flujo de trabajo.

Por tanto, la motivación de este proyecto es digitalizar y centralizar este proceso mediante una solución tecnológica a medida. La aplicación reúne a entrenadores y atletas en un único cliente móvil: los entrenadores de Bulderland diseñan los planes de entrenamiento y hacen seguimiento del rendimiento de sus atletas, mientras que los atletas se inscriben en los planes que desean, consultan su rutina diaria y registran sus resultados.

1.2. Objetivos del proyecto

El objetivo principal de este proyecto es el desarrollo de una aplicación móvil que permita la gestión remota del entrenamiento del centro de escalada Bulderland, a través del uso de una arquitectura cliente-servidor. Además, en cuanto a asegurar la calidad y el despliegue, este proyecto está orientado a automatizar el proceso de pruebas con las pruebas unitarias, de integración y de extremo a extremo con *Maestro*, una herramienta que ejecuta los flujos completos de usuario sobre la aplicación ya instalada en un emulador; y el proceso de despliegue usando Docker.

1.3. Estructura del documento

El resto de la memoria se organiza en cinco capítulos. El Capítulo 2 sitúa el proyecto frente a las soluciones de mercado existentes. El Capítulo 3 recoge el análisis de requisitos funcionales y no funcionales. El Capítulo 4 aborda el diseño del sistema y su implementación, detallando su arquitectura, tecnologías, base de datos, API e interfaz, junto con sus funcionalidades principales. El Capítulo 5 describe las pruebas, el despliegue en producción y la organización del código. Por último, el Capítulo 6 cierra con las conclusiones y el trabajo futuro. Además, se incluye el Anexo A, que detalla la planificación y la distribución temporal del desarrollo de este proyecto.

Capítulo 2

Estado del arte

En los últimos años, la digitalización de la planificación deportiva ha continuado su rápida expansión, en un escenario de crecientes plataformas de entrenador remoto que han complementado y en algunos casos superado a las herramientas tradicionales de gestión manual. En este capítulo, se expone el estado del arte del software de seguimiento deportivo, tanto desde el punto de vista de las aplicaciones generalistas como de las plataformas específicas de escalada, y se señalan las aportaciones de la solución diseñada para Bulderland.

El mercado actual ofrece diversos productos para el autoseguimiento individual, el entrenamiento remoto genérico y el seguimiento específico de la escalada, pero ninguno de ellos combina la creación de los planes por el propio personal técnico del centro con un modelo de acceso sin coste recurrente por atleta:

Aplicaciones de actividad física y autoseguimiento. Aplicaciones de consumo masivo como *Strava* [4] o *MyFitnessPal* [5] se encargan del autoseguimiento individual, pero no contemplan la figura de un entrenador que crea y prescribe planes a un grupo de atletas, que es la base del proceso de Bulderland.

Plataformas de relación entrenador-atleta. *TrainingPeaks* [6] y *TrueCoach* [7] sí se ocupan de la relación entrenador-atleta y ofrecen prescripción de planes y seguimiento, pero se alejan del caso planteado en dos aspectos: están pensadas para deportes de resistencia o el gimnasio convencional, no para la escalada, y cobran una cuota de suscripción por atleta que crece con el número de atletas del centro.

Aplicaciones específicas de escalada. Dentro de este nicho reducido se encuentran *Lattice Training* [8], que presenta protocolos de entrenamiento de fuerza y suspensión, y *TopLogger* [9], que permite el registro de vías y bloques. La más cercana es *Lattice Training*: permite el seguimiento y la comunicación con el entrenador, pero ese entrenador es un profesional remoto de terceros, cuyo coste se calcula por atleta y que no forma parte del personal técnico del centro. Además, ninguna de las dos ofrece

la gestión de planes asignados por el propio personal técnico junto con un canal de comunicación directo entrenador-atleta en la misma plataforma.

Herramientas genéricas no especializadas. Las hojas de cálculo y las aplicaciones de mensajería genéricas, como *WhatsApp* [10], permiten improvisar el caso de uso, pero dispersan la información y no ofrecen seguimiento histórico del rendimiento.

Un resumen de esta comparativa se recoge en la Tabla 2.1, que evidencia el hueco que motiva el desarrollo de una solución propia.

Solución	**Entrenador – Atleta**	**Enfoque escalada**	**Comunicación**	**A medida**
Strava / *MyFitnessPal*	No	No	No	No
TrainingPeaks / *TrueCoach*	Sí	No	Parcial	No
Lattice Training	Sí	Sí	Sí	No
TopLogger	No	Sí	No	No
Hojas de cálculo / *WhatsApp*	Parcial	No	Sí	No
Solución propuesta	Sí	Sí	Sí	Sí

Tabla 2.1: Comparativa de las soluciones existentes frente a los requisitos del proyecto.

El carácter «a medida» de la Tabla 2.1 se concreta en dos aspectos que ninguna herramienta del mercado resuelve para este caso. El primero es la autoría del entrenamiento: en la solución propuesta son los propios entrenadores de Bulderland quienes diseñan los planes, los asignan a sus atletas y hacen el seguimiento dentro de la misma plataforma; no se trata de un entrenador remoto externo contratado aparte (como en la modalidad de pago de *Lattice Training* [8]). El segundo es el modelo de pago: las plataformas de *coaching* operan bajo una suscripción con coste por atleta que crece con el tamaño del centro, mientras que en la solución propuesta el cobro de la cuota se gestiona de forma externa a la aplicación y el acceso se controla por abono, sin un coste recurrente por usuario ligado a un proveedor externo.

Capítulo 3

Análisis del sistema

Este capítulo recoge en primer lugar los requisitos funcionales y no funcionales de Bulderland. Después, define los roles de usuario que participan en la plataforma. Por último, modela los casos de uso principales que guían las decisiones de diseño e implementación.

3.1. Análisis de requisitos

La especificación de requisitos se ha elaborado a partir del análisis de la situación de partida y de las necesidades detectadas en el funcionamiento del centro. Los requisitos se han clasificado en funcionales (qué debe hacer el sistema) y no funcionales (cómo debe comportarse en términos de calidad). Para facilitar su trazabilidad, cada requisito se identifica con un código: `RF` para los funcionales y `RNF` para los no funcionales.

Los requisitos funcionales se han agrupado por módulos lógicos y se muestran en las Tablas 3.1, 3.2, 3.3 y 3.4.

ID	Requisito	Descripción
RF-01	Registro de atletas	Un usuario puede crear una cuenta de atleta mediante correo electrónico y contraseña.
RF-02	Verificación de cuenta	El sistema valida el correo del nuevo usuario mediante un código OTP de un solo uso con caducidad temporal.
RF-03	Inicio de sesión	Un usuario registrado puede autenticarse con sus credenciales y obtener un *token* de sesión.
RF-04	Acceso con Google	El usuario puede registrarse e iniciar sesión mediante su cuenta de Google.
RF-05	Recuperar contraseña	El usuario puede solicitar y completar el restablecimiento de su contraseña mediante un código de un solo uso.
RF-06	Gestionar perfil	El usuario puede consultar y editar sus datos personales y subir una foto de perfil.
RF-07	Baja de cuenta	El usuario puede solicitar la baja lógica de su propia cuenta.
RF-08	Alta de privilegiados	El administrador puede dar de alta usuarios con rol entrenador o administrador.
RF-09	Administrar usuarios	El administrador puede listar, eliminar y restaurar cuentas de usuario.
RF-10	Solicitar abono	El atleta puede solicitar la activación de su abono.
RF-11	Revisar abonos	El administrador lista las solicitudes de abono pendientes de aprobación.
RF-12	Gestionar abonos	El administrador aprueba o rechaza las solicitudes.

Tabla 3.1: Requisitos funcionales del módulo de cuentas y autenticación.

ID	Requisito	Descripción
RF-13	Crear planes	El entrenador puede crear planes de entrenamiento definiendo nombre, fechas y atletas asociados.
RF-14	Editar planes	El entrenador puede modificar los datos de un plan o darlo de baja lógica.
RF-15	Duplicar planes	El entrenador puede duplicar un plan completo junto con sus sesiones para reutilizarlo.
RF-16	Explorar planes	El atleta puede consultar el catálogo de planes activos disponibles.
RF-17	Inscribirse en planes	El atleta puede inscribirse en los planes en los que desee participar.
RF-18	Crear sesiones	El entrenador puede definir sesiones de entrenamiento con fecha, duración y una lista de ejercicios.
RF-19	Editar sesiones	El entrenador puede modificar o duplicar sesiones y darlas de baja lógica.
RF-20	Asociar multimedia	El entrenador puede vincular vídeos de técnica y fotos a los ejercicios de una sesión.
RF-21	Consultar calendario	El atleta consulta sus sesiones programadas en una vista de calendario mensual.
RF-22	Apuntarse a sesiones	El atleta puede confirmar o anular su asistencia a una sesión concreta.

Tabla 3.2: Requisitos funcionales del módulo de planes y sesiones.

ID	Requisito	Descripción
RF-23	Registrar ejecución	El atleta registra el rendimiento real de una sesión (series, cargas y anotaciones) y la marca como completada.
RF-24	Consultar historial	El atleta puede consultar su historial de registros de ejecución, y su entrenador el de los atletas inscritos en sus planes.
RF-25	Editar registros	El atleta puede actualizar el registro de una sesión que ya había guardado.
RF-26	Cuadro de mando	El usuario accede a un panel con estadísticas adaptadas a su rol (sesiones, horas y adherencia).
RF-27	Progreso histórico	El sistema muestra la evolución histórica de un ejercicio o métrica a lo largo del tiempo.
RF-28	Seguimiento de equipo	El entrenador visualiza la actividad agregada de su equipo y los atletas que requieren seguimiento.
RF-29	Exportar registros	El entrenador puede descargar una hoja de cálculo con los registros de ejecución y un resumen estadístico de los atletas inscritos en sus planes.

Tabla 3.3: Requisitos funcionales del módulo de ejecución y estadísticas.

ID	Requisito	Descripción
RF-30	Chat en tiempo real	Entrenadores y atletas de un plan intercambian mensajes en un chat grupal con entrega inmediata.
RF-31	Historial de mensajes	El sistema persiste y muestra el historial de mensajes de cada plan.
RF-32	Notificaciones *push*	El sistema envía alertas *push* ante eventos relevantes (nuevos planes, cambios en sesiones, etc.).
RF-33	Bandeja de notificaciones	El usuario consulta sus notificaciones, las marca como leídas y las elimina.
RF-34	Gestión de multimedia	El administrador mantiene las bibliotecas de vídeos de técnica y fotos de ejercicios.

Tabla 3.4: Requisitos funcionales del módulo de comunicación y notificaciones.

Los requisitos no funcionales, mostrados en la Tabla 3.5, establecen los atributos de calidad que el sistema debe satisfacer, condicionando las decisiones de arquitectura y tecnología adoptadas posteriormente.

ID	Descripción
RNF-01	**Seguridad.** Las contraseñas se almacenan cifradas (*hash* bcrypt) y el acceso a los recursos protegidos exige un *token* JWT válido.
RNF-02	**Autorización.** El sistema aplica control de acceso basado en roles y comprobaciones de propiedad sobre los recursos sensibles.
RNF-03	**Robustez.** Toda entrada de escritura se valida contra un esquema antes de procesarse, rechazando datos malformados con errores descriptivos.
RNF-04	**Disponibilidad.** La comunicación en tiempo real se recupera automáticamente ante caídas de red mediante reconexión transparente.
RNF-05	**Multiplataforma.** La aplicación cliente debe ejecutarse desde una única base de código tanto en Android como en iOS.
RNF-06	**Usabilidad.** La interfaz sigue un enfoque *mobile-first*, con retroalimentación visual continua ante las acciones del usuario.
RNF-07	**Rendimiento.** Las consultas habituales deben resolverse con baja latencia, aprovechando la desnormalización del modelo documental.
RNF-08	**Mantenibilidad.** El código se organiza por capas (rutas, controladores y modelos), favoreciendo el desacoplamiento y la legibilidad.
RNF-09	**Integridad.** Las eliminaciones sobre entidades núcleo aplican borrado lógico para preservar el histórico.
RNF-10	**Reproducibilidad.** El entorno de ejecución se conteneriza con Docker para garantizar despliegues predecibles.
RNF-11	**Calidad.** El sistema se valida mediante pruebas unitarias, de integración y de extremo a extremo automatizadas.
RNF-12	**Protección.** Se aplica un limitador de tasa y un límite de tamaño de petición para mitigar abusos.

Tabla 3.5: Requisitos no funcionales del sistema.

3.2. Definición de los roles de usuario

El sistema tiene un modelo de control de acceso basado en roles, por lo que cada usuario solo puede acceder a la información y funciones relacionadas con su perfil. Los roles que se contemplan son:

- **Atleta:** es el usuario final y el rol más numeroso. El atleta consulta sus planes y sesiones, se suscribe a los planes disponibles, registra la realización de las sesiones de entrenamiento, consulta las estadísticas de progreso y se comunica con su entrenador usando el chat.
- **Entrenador:** es un rol del personal técnico de Bulderland. El entrenador tiene la posibilidad de crear y administrar planes, crear y editar sesiones y ejercicios, enlazar recursos multimedia, monitorizar la actividad de los atletas inscritos y comunicarse con ellos. Sin embargo, no puede administrar las cuentas de los usuarios ni la biblioteca global de multimedia.

- **Administrador:** es el nivel de privilegio más alto, que controla el funcionamiento general de la plataforma. Administra el ciclo de vida de las cuentas de usuario (la creación de cuentas con perfil privilegiado y la recuperación de cuentas eliminadas), la aprobación y gestión de los abonos, el mantenimiento de la biblioteca de vídeos de técnica y fotos de ejercicios, y las estadísticas de salud y uso del sistema. Su relación con los planes de entrenamiento está limitada por la función de moderación: puede archivarlos, pero no tiene acceso a la pantalla de creación y edición que está diseñada para el entrenador.

3.3. Casos de uso

A partir de los requisitos funcionales y de los roles definidos, se han identificado los casos de uso que modelan las interacciones más representativas entre los actores y el sistema. CU-12 se modela como una inclusión de CU-11, ya que programar las sesiones de un plan solo tiene sentido dentro de la gestión de un plan ya existente; el resto de variantes opcionales (duplicar un plan o una sesión, restaurar un usuario dado de baja, cambiar la foto de perfil) quedan recogidas dentro de su caso de uso base en lugar de representarse como extensiones aparte, ya que no cambian de actor ni añaden una interacción distinta con el sistema. La Figura 3.1 recoge el diagrama UML del catálogo completo y la Tabla 3.6 describe cada caso de uso junto con sus actores y flujo principal.

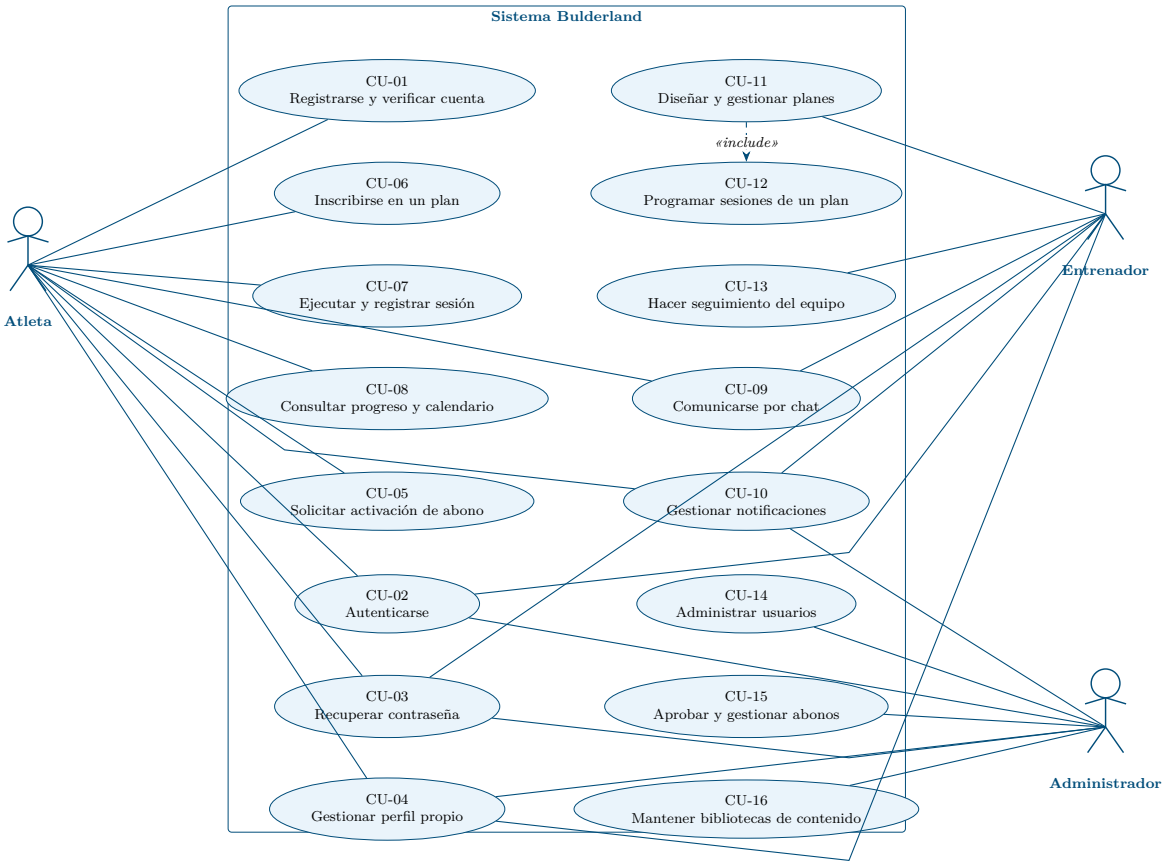

Figura 3.1: Diagrama UML de casos de uso del sistema.

ID	Actor(es)	Descripción
CU-01	Atleta	El usuario introduce sus datos, recibe un código de un solo uso por correo electrónico, lo valida y su cuenta queda activa (RF-01, RF-02).
CU-02	Todos	El usuario introduce sus credenciales o utiliza su cuenta de Google; el sistema verifica la identidad y emite un *token* de sesión (RF-03, RF-04).
CU-03	Todos	El usuario solicita el restablecimiento de contraseña, recibe un código de un solo uso y establece una nueva contraseña (RF-05).
CU-04	Todos	El usuario consulta y edita sus datos personales, sube una foto de perfil o solicita la baja lógica de su cuenta (RF-06, RF-07).
CU-05	Atleta	El atleta solicita la activación de su abono; el sistema registra la petición y queda pendiente de aprobación por el administrador (RF-10).
CU-06	Atleta	El atleta explora el catálogo de planes activos, selecciona uno y confirma su inscripción (RF-16, RF-17).
CU-07	Atleta	El atleta abre una sesión programada, confirma asistencia, completa los ejercicios registrando series, cargas y anotaciones, marca la sesión como completada y puede editar sus registros (RF-22, RF-23, RF-25).
CU-08	Atleta	El atleta consulta el calendario mensual, el historial de ejecución y accede a su panel de estadísticas de progreso (RF-21, RF-24, RF-26, RF-27).
CU-09	Atleta, Entrenador	Entrenador y atleta intercambian mensajes en el chat grupal del plan, con entrega inmediata y persistencia del historial (RF-30, RF-31).
CU-10	Todos	El usuario recibe alertas ante eventos relevantes, consulta su bandeja de notificaciones, las marca como leídas y las elimina (RF-32, RF-33).
CU-11	Entrenador	El entrenador crea un plan definiendo nombre, fechas y atletas; puede editarlo, duplicarlo o darlo de baja lógica (RF-13, RF-14, RF-15). Incluye CU-12.
CU-12	Entrenador	El entrenador define sesiones dentro de un plan, con fecha, duración y lista de ejercicios; puede vincular vídeos y fotos a cada ejercicio (RF-18, RF-19, RF-20).
CU-13	Entrenador	El entrenador visualiza la actividad agregada de su equipo y el detalle de los atletas que requieren seguimiento, y puede descargar sus registros en una hoja de cálculo (RF-24, RF-28, RF-29).
CU-14	Administrador	El administrador lista las cuentas del sistema, da de alta usuarios con rol privilegiado, elimina cuentas y restaura las dadas de baja (RF-08, RF-09).
CU-15	Administrador	El administrador revisa las solicitudes de abono pendientes, las aprueba o rechaza; el sistema activa el abono con caducidad automática (RF-11, RF-12).
CU-16	Administrador	El administrador mantiene la biblioteca global de vídeos de técnica y fotos de ejercicios disponibles para los entrenadores (RF-34).

Tabla 3.6: Catálogo de casos de uso del sistema.

Capítulo 4

Diseño e implementación

Este capítulo traduce los requisitos recogidos en el capítulo anterior en decisiones de diseño. Primero se describen la tecnología elegida para cada capa y la arquitectura general del sistema construida sobre ella. Después, se detallan el modelo de datos y la interfaz que exponen esa arquitectura al usuario.

4.1. Selección de tecnologías

Antes de definir la arquitectura se valoraron distintas alternativas para cada capa del sistema. La elección se hizo principalmente teniendo en cuenta que se trataba de un desarrollo individual y que las tecnologías escogidas debían integrarse bien entre sí.

Cliente móvil multiplataforma. El requisito de desplegar en Android e iOS desde una única base de código descartó el desarrollo nativo independiente, que habría duplicado el esfuerzo. Entre las opciones de desarrollo multiplataforma se compararon React Native [11] y Flutter [12]. Aunque Flutter ofrece un buen rendimiento gráfico, emplea el lenguaje Dart, ajeno al resto del proyecto. Se optó por React Native porque permite emplear JavaScript/TypeScript —el mismo lenguaje que el *backend*—, unificando el ecosistema. Adicionalmente, el uso de Expo [13] simplifica la configuración del entorno, el acceso a las APIs nativas (notificaciones, almacenamiento seguro) y el proceso de compilación.

Servidor y lógica de negocio. Para el *backend* se valoraron entornos como Django (Python) [14], Spring Boot (Java) [15] y Express sobre Node.js. Se escogió Node.js con Express [16, 17] para poder emplear JavaScript/TypeScript tanto en el cliente como en el servidor, lo que simplificó el desarrollo al tratarse de un proyecto individual. Su modelo orientado a eventos encajaba además con las necesidades de comunicación en tiempo real de la aplicación.

Persistencia de datos. La elección de la base de datos NoSQL MongoDB [18] frente al modelo relacional se debe a su flexibilidad polimórfica del modelo documental para

ejercicios de naturaleza variable, mejor rendimiento de lectura por desnormalización y ausencia de desajuste entre objetos y tablas gracias al formato BSON/JSON compartido con Node.js.

Comunicación en tiempo real. Para el chat se consideró el uso directo de la API WebSocket nativa y servicios gestionados como Firebase [19]. Se optó por la librería Socket.io [20] por ofrecer, sobre el protocolo WebSocket, abstracciones de alto nivel (gestión de salas, reconexión automática y mecanismos de fiabilidad) sin la dependencia de un proveedor externo de pago que sí implicaría una solución como Firebase.

Infraestructura y pruebas. Por último, se adoptó Docker [21] para contenerizar el *backend* y la base de datos, garantizando entornos reproducibles y un despliegue predecible frente a la instalación manual de dependencias en el servidor. Para las pruebas de extremo a extremo se eligió Maestro [22] frente a alternativas como Appium o Detox, por su sintaxis declarativa en YAML y su menor complejidad de configuración, idónea para automatizar los flujos de usuario completos de la aplicación.

4.2. Arquitectura de la solución

La arquitectura de la solución se construye basada en un modelo de interacción cliente-servidor utilizando REST. La arquitectura permite el desacoplamiento del cliente móvil y el *backend* del servidor, lo que hace que su evolución sea independiente, junto con la comunicación en tiempo real, que se implementa mediante un canal WebSocket. Las dos partes principales son:

- ***Frontend* móvil (cliente):** Implementado con el marco React Native y el conjunto de herramientas Expo. Esta parte contiene la lógica de presentación, la implementación de la interfaz de usuario y la lógica para almacenar el estado en el almacén local. Específicamente el *token* de sesión se almacena con la ayuda de la API Secure Store para evitar el almacenamiento en texto plano.
- ***Backend* centralizado (servidor):** Implementado con Node.js. Es responsable de centralizar la lógica de negocio, la implementación de la autenticación y la autorización basada en roles, el manejo de las solicitudes del cliente móvil, la gestión de la comunicación en tiempo real y el almacenamiento de datos en la base de datos.

El código del *backend* de la solución está estructurado de forma modular y dividido en tres capas independientes. Esto permite una mejor mantenibilidad y pruebas más aisladas de cada capa. Cada solicitud sigue el siguiente flujo:

- **Capa de ruta:** Es el punto de entrada al sistema. En este punto se aplican los *middleware* transversales: CORS, la autenticación JWT para cada punto final [23], el control de acceso basado en roles y la validación de esquemas con Zod [24], así como un *middleware* de límite de tasa que identifica al cliente ya sea por su identificador de usuario si este último está autenticado a través de JWT o simplemente por su dirección IP para no penalizar a todos los atletas que están conectados a la red del gimnasio de escalada con la misma cuota de solicitud. Luego la solicitud se envía al controlador correspondiente.

- **Capa de controlador:** Recibe la petición validada pasada por la capa de ruta, ejecuta la lógica de negocio y se asegura de que el usuario tenga autorización para acceder al recurso solicitado, como por ejemplo verificar que un entrenador sea el propietario del plan que está editando, y luego envía la respuesta al cliente en formato JSON.

- **Capa de modelo (Mongoose):** Maneja la persistencia de los datos usando objetos de Mongoose [25] que primero validarán el esquema y luego realizarán las operaciones CRUD y las transacciones necesarias en la base de datos MongoDB.

La Figura 4.1 resume esta arquitectura por capas, junto con el canal de tiempo real y los servicios externos que se describen a continuación.

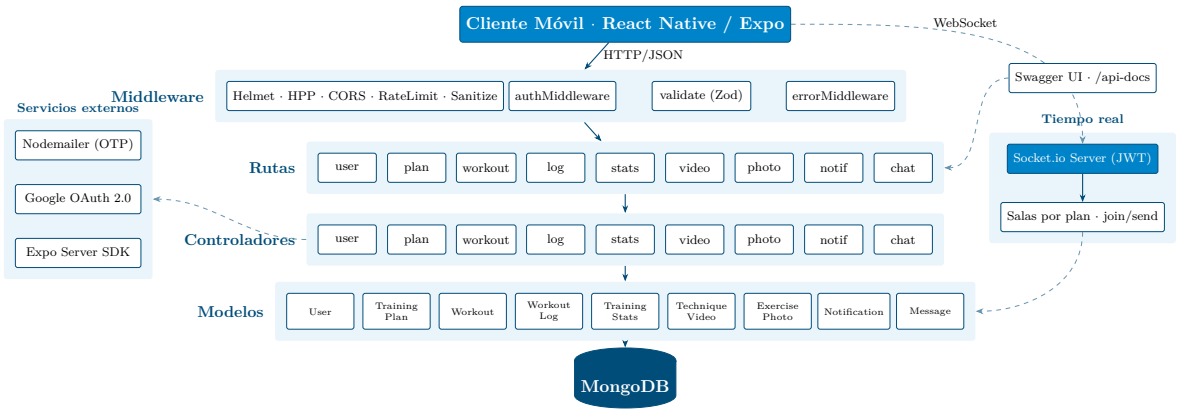

Figura 4.1: Arquitectura por capas del *backend*: cadena de *middleware*, rutas, controladores y modelos para los nueve recursos de la API, canal de tiempo real y servicios externos.

4.2.1. Comunicación en tiempo real

Para facilitar una comunicación fluida e inmediata entre el personal técnico y los atletas dentro de la plataforma, se ha desarrollado una función de mensajería en tiempo real.

La base de la arquitectura de este subsistema es el protocolo WebSocket, que proporciona una comunicación bidireccional y de baja latencia al mantener una sola conexión persistente, en lugar de tener que abrir una nueva por cada petición.

La implementación de esta funcionalidad se logra utilizando la biblioteca Socket.io tanto en el entorno del servidor como en el del cliente. La biblioteca permite al sistema recibir una abstracción de alto nivel de los mecanismos para administrar salas particulares que están mapeadas estáticamente al plan de entrenamiento para separar el flujo de mensajes y proporcionar reconexiones automáticas y mecanismos de fiabilidad en caso de problemas de red.

El chat utiliza el mismo mecanismo de autenticación que el resto del *backend*: el cliente establece la conexión Socket.io usando el mismo *token* JWT en el proceso de *handshake*, y un *middleware* de Socket.io lo verifica antes de establecer la conexión. La Figura 4.2 muestra en un diagrama UML la secuencia desde el establecimiento de la conexión hasta que el mensaje llega al resto de los participantes; el servidor confirma cada solicitud del cliente con un acuse de recibo (`ack`), que este emplea para saber si la operación prosperó.

Existen dos verificaciones de los derechos de acceso a la sala (que están restringidos al propietario del plan y a los atletas que están inscritos en el plan): en el momento de unirse a la sala y en cada mensaje. La segunda no es redundante: el manejador de `send_message` no da por supuesto que el emisor se haya unido antes a la sala, sino que resuelve el permiso otra vez a partir del plan, de modo que cada mensaje se autoriza por separado. Los mensajes se emiten a toda la sala, por lo que la conversación se mantiene sincronizada en todos los clientes conectados.

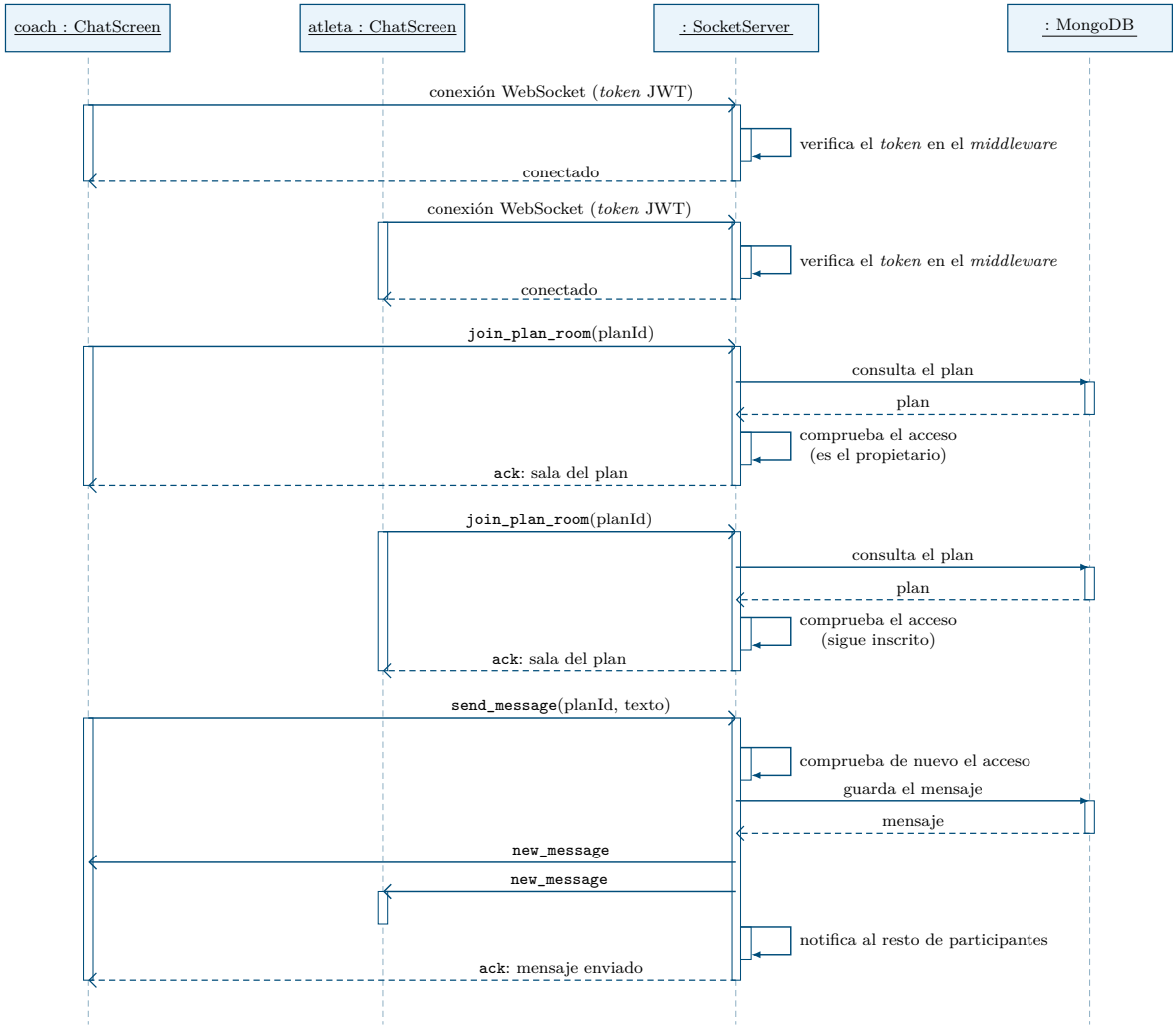

Figura 4.2: Diagrama UML de secuencia del chat en tiempo real: apertura de sesión de los dos participantes de una sala y difusión de un mensaje. Las puntas de flecha rellenas representan llamadas síncronas, las abiertas mensajes asíncronos y las líneas discontinuas, respuestas.

4.2.2. Control de acceso por abono

Debido a que no existe una pasarela de pago unificada, es el atleta quien envía una solicitud para activar su abono introduciendo su número de identificación. Posteriormente, el administrador verifica el pago y activa el abono seleccionando uno de los periodos disponibles (mensual, trimestral o anual). De esta manera, la fecha de vencimiento se calcula automáticamente sin necesidad de introducirla manualmente.

El acceso al material de entrenamiento se controla a través de una capa de *middleware* que excluye a los entrenadores y administradores (no requieren abono). Esta capa rechaza la solicitud de un atleta sin un abono válido y devuelve el código de error correspondiente; por lo tanto, el cliente móvil diferencia esta situación de la denegación de permisos habitual y muestra una notificación específica. La fecha de vencimiento, sin embargo, no está vinculada a ninguna acción del atleta: una vez al

día, una tarea programada verifica los abonos vencidos, los actualiza e inicia el mismo procedimiento de envío de notificaciones descrito en la Sección 4.3. La Figura 4.3 resume cómo estos tres puntos —activación, control de acceso y caducidad— comparten el mismo estado del abono.

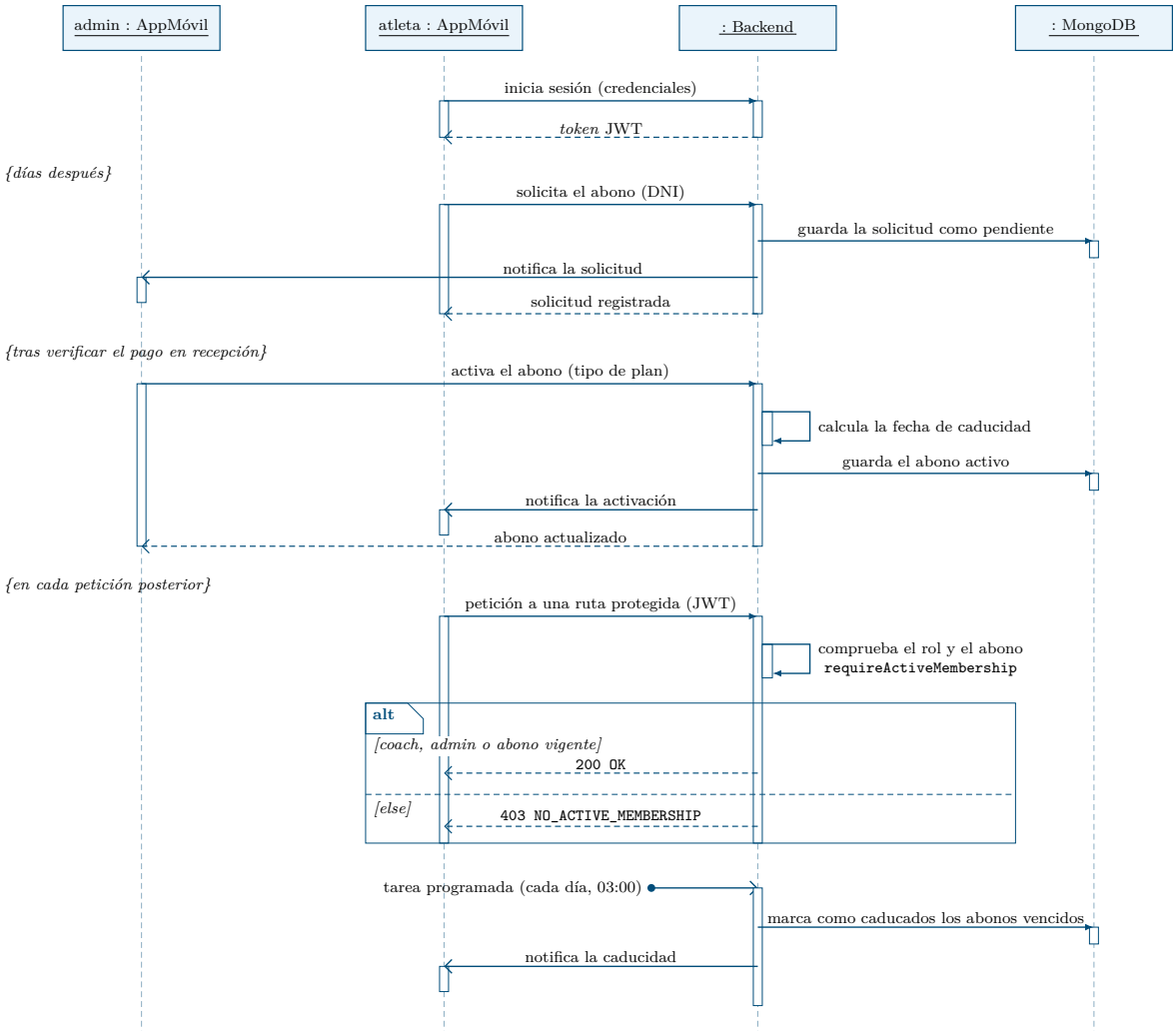

Figura 4.3: Diagrama UML de secuencia del ciclo de vida del abono: inicio de sesión, solicitud del atleta, aprobación del administrador, control de acceso en cada petición y caducidad automática.

4.3. Integración de servicios externos

Con el objetivo de no reimplementar funcionalidades estándar ya resueltas y reducir la carga de mantenimiento, la arquitectura delega ciertas responsabilidades clave a servicios externos de terceros:

- **Autenticación de Google OAuth 2.0 [26]:** Integración del flujo de inicio de sesión único mediante el proveedor de identidad de Google. Los usuarios pueden registrarse y autenticarse con su cuenta de Google. Implementación en el lado del servidor de la decodificación y validación del ID token enviado por el cliente

mediante la biblioteca oficial google-auth-library, lo que evita el almacenamiento de contraseñas y añade una capa de seguridad.

- **Verificación de correo electrónico usando Nodemailer [27]:** Este módulo es responsable del envío automático de correos electrónicos a través del servidor SMTP. El servicio crea contraseñas númericas de un solo uso que son ingresadas por el usuario para verificar su cuenta cuando se registra o solicita un restablecimiento de contraseña; por lo tanto, el proceso se completa con éxito solo con una dirección de correo electrónico válida que sea controlada por el registrante.

- **Notificación Push (Expo Server SDK):** Envío de alertas *push* usando la API de Expo. El uso de Expo permite abstraer las diferencias entre los servicios nativos de Android y iOS en una sola interfaz.

La Figura 4.4 muestra el ciclo de vida de una notificación desde el evento que la desencadena hasta su recepción en el dispositivo. Primero se filtran los destinatarios cuyos avisos están apagados y, para todos los demás destinatarios, el aviso se registra en la base de datos (creando un historial de la aplicación) y se envía como un aviso *push* a los destinatarios cuyos perfiles contienen un *token* de Expo válido. Los *tokens* inválidos son descartados y los mensajes válidos se agrupan, ya que hay un límite de destinatarios por solicitud impuesto por la API de Expo.

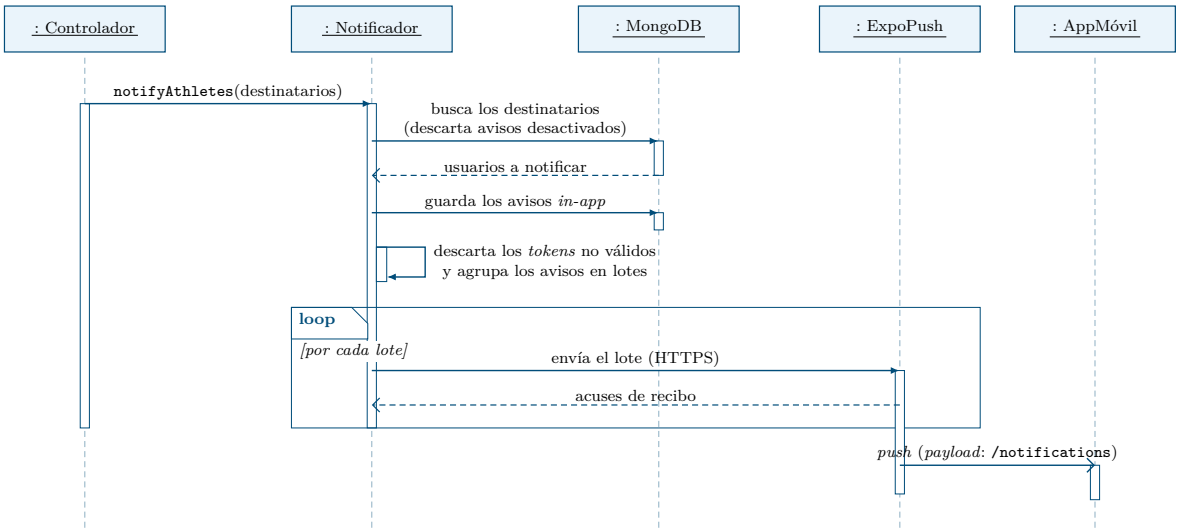

Figura 4.4: Diagrama UML de secuencia de una notificación *push*, desde el evento de dominio hasta su entrega en el dispositivo.

4.4. Diseño de la base de datos

El sistema utiliza MongoDB como base de datos para su capa de persistencia, una base de datos NoSQL orientada a documentos. Existen varias razones técnicas por las cuales se ha elegido este paradigma en lugar del relacional:

1. **Polimorfismo:** El modelo de documentos no requiere la fragmentación de la información en varias tablas. Puede suceder que una sesión de entrenamiento incluya diferentes tipos de métricas como series de cardio, ejercicios de fuerza organizados en series y actividades de escalada en roca medidas en tiempo. Si se utilizara una base de datos relacional se tendrían que crear varias tablas relacionadas o dejar algunas con muchos espacios en blanco. En MongoDB toda la información de una sesión de entrenamiento se almacena en un solo documento que contiene un array de ejercicios cuyo esquema es diferente en cada documento.

2. **Mejor rendimiento en lectura:** La información no tiene que ser unida mediante sentencias complejas para consultarla ya que no existen tablas separadas. Así, la aplicación móvil obtiene toda la información de una sesión obteniendo solo un documento en lugar de combinar la información de varias tablas. Por ejemplo, al abrir el historial de una sesión, la aplicación recupera con una sola consulta el documento `WorkoutLog`, que ya lleva embebido el array de ejercicios realizados con sus series, repeticiones y peso; en un modelo relacional, esa misma pantalla habría requerido combinar una tabla de registros con otras de ejercicios y series.

3. **Ausencia de desajuste entre objetos y tablas (*impedance mismatch*):** La información en las bases de datos relacionales se almacena en filas y columnas mientras que el código de la aplicación se escribe con objetos. Esto crea un desajuste y requiere el uso de una herramienta de Mapeo Objeto-Relacional que realice la transformación. MongoDB, por el contrario, utiliza el modelo de documentos con el formato BSON, muy similar a los objetos de JavaScript y que se adapta perfectamente a las estructuras de datos creadas en el *backend* de Node.js, por lo que no se necesita el paso intermedio de transformar los datos. Mongoose es una herramienta de Mapeo Objeto-Documento utilizada en el proyecto para definir los esquemas y acceder a los documentos.

La Figura 4.5 muestra el diagrama entidad-relación de la aplicación.

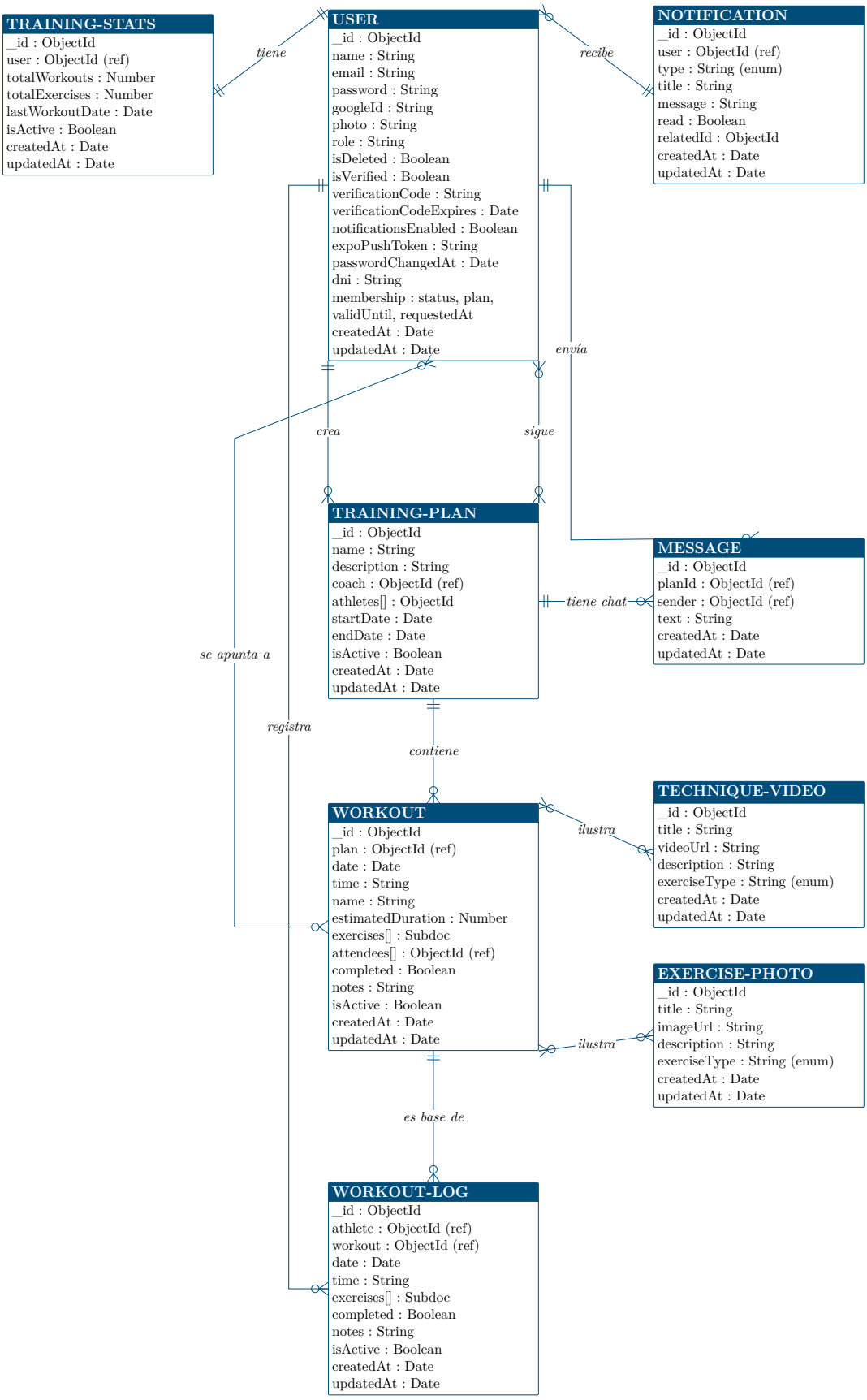

Figura 4.5: Diagrama entidad–relación de la base de datos (nueve colecciones).

Los documentos mostrados en la Figura 4.5 no contienen la información binaria de fotos o vídeos, solo contienen el enlace a ellos. En el caso de subir una foto de perfil o una foto de ejercicio por parte del administrador, el cliente envía en el cuerpo de la petición la foto codificada en Base64; en el lado del servidor se decodifica, se comprueba si no es mayor a 500 KB y se guarda como archivo en un directorio expuesto estáticamente por Express y conservado entre despliegues. En el documento correspondiente en la colección de MongoDB –en el campo `photo` del usuario o `imageUrl` de la foto de ejercicio– solo se guarda la URL, no el contenido binario. La técnica para los vídeos se utiliza de la misma manera usando URL ya que no es necesario subirlos al servidor, ya que el administrador solo hace un enlace a un vídeo externo. Esta separación entre los datos y el contenido multimedia evita guardar binarios en los documentos de MongoDB, lo cual tendría un efecto negativo en el tamaño de los índices y el rendimiento de las consultas; además, el formato BSON también impone un límite máximo de 16 MB por documento, lo cual haría imposible adjuntar imágenes o vídeos de tamaño completo.

4.5. Diseño de la interfaz de usuario

En lo que a la interfaz se refiere, ha sido diseñada teniendo en cuenta el enfoque *mobile-first*, lo que significa tener pantallas sencillas, textos fácilmente legibles incluso desde lejos y elementos lo suficientemente grandes para hacer clic, ya que la aplicación se utiliza principalmente en dispositivos móviles durante las sesiones de entrenamiento.

Dicho esto, se establecieron tres pautas principales:

- **Adaptación de rol:** Diferentes roles (administradores, entrenadores y atletas) pueden utilizar una interfaz y una navegación distintas, presentando las funcionalidades que necesitan y, por lo tanto, sin abrumarlos con información.
- **Consistencia visual:** Se ha definido el lenguaje de diseño y es consistente a lo largo de toda la interfaz (usando la paleta de colores corporativa de *Bulderland*, las fuentes, el espaciado y los componentes).
- **Retroalimentación continua:** Las acciones del usuario generan retroalimentación visual a través de notificaciones emergentes (*toasts*) o confirmaciones antes de realizar las acciones como eliminar datos.

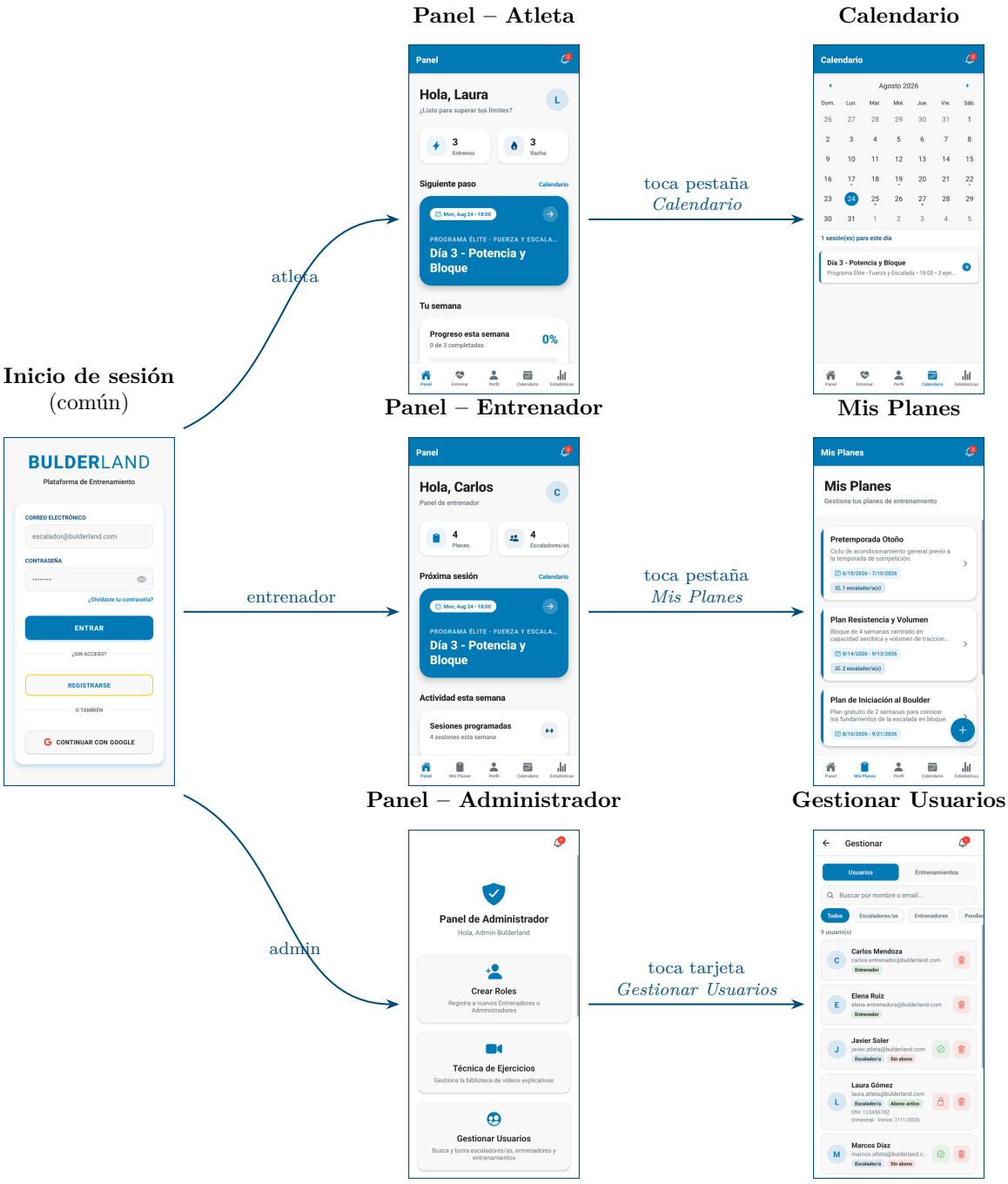

Figura 4.6: Mapa de navegación desde el inicio de sesión hasta una pantalla de segundo nivel, según el rol del usuario.

Capítulo 5

Pruebas y despliegue

Este capítulo marca el final del proceso de implementación de este sistema: se prueba el sistema construido de forma automatizada a distintos niveles, se explica su despliegue en el entorno de producción de la universidad y se detalla la organización del código fuente resultante.

5.1. Enfoque de pruebas

El proceso de validación del sistema se divide en dos partes: pruebas unitarias y de integración, donde los controladores de *backend* y los elementos de la interfaz móvil se validan por separado, y pruebas de extremo a extremo, donde se simula todo el flujo de la aplicación en un dispositivo real.

El *backend* utiliza Jest [28] y Supertest [29] para probar los controladores, modelos, *middlewares* y rutas a través de llamadas HTTP simuladas hechas contra una base de datos de MongoDB de prueba que se ejecuta en Docker; con una cobertura de líneas de código del 91,5 % (Figura 5.1). El envío de correos electrónicos y las notificaciones *push* tienen una cobertura menor porque dependen de servicios de terceros. El cliente móvil utiliza React Testing Library [30] para validar la lógica de negocio y el comportamiento de las pantallas para cada rol, con aproximadamente un 83,7 % de cobertura de líneas de código (Figura 5.2). La Tabla 5.1 resume y compara el resto de métricas de cobertura globales extraídas para ambos entornos.

All files

90.83% Statements 1516/1669 **74.05%** Branches 665/898 **92.55%** Functions 199/215 **91.5%** Lines 1443/1577

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
src	88%	44/50	79.41%	27/34	62.5%	5/8	89.36%	42/47
src/config	95.19%	99/104	68.18%	30/44	100%	13/13	94.68%	89/94
src/constants	92.3%	12/13	66.66%	2/3	100%	1/1	92.3%	12/13
src/controllers	89.79%	889/990	73.68%	462/627	92.5%	111/120	90.64%	853/941
src/middleware	88.5%	77/87	75.75%	50/66	100%	16/16	88%	66/75
src/models	100%	34/34	100%	16/16	100%	5/5	100%	32/32
src/models/logs	100%	5/5	100%	0/0	100%	0/0	100%	5/5
src/models/plugins	100%	9/9	83.33%	5/6	100%	3/3	100%	9/9
src/models/templates	100%	11/11	100%	2/2	100%	1/1	100%	11/11
src/routes	99%	99/100	50%	1/2	50%	1/2	99%	99/100
src/schemas	100%	56/56	100%	0/0	100%	0/0	100%	56/56
src/tests/helpers	94.44%	17/18	62.5%	5/8	100%	2/2	94.44%	17/18
src/utils	85.41%	164/192	72.22%	65/90	93.18%	41/44	86.36%	152/176

Figura 5.1: Cobertura del conjunto de pruebas de Jest en el *backend*, por directorio.

All files

81.76% Statements 1776/2172 **68.7%** Branches 1216/1770 **73.08%** Functions 372/509 **83.67%** Lines 1696/2027

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
app	80.99%	179/221	73.12%	117/160	69.69%	23/33	83.57%	173/207
app/(admin)	86.43%	172/199	73.51%	136/185	83.05%	49/59	87.84%	159/181
app/(admin)/photos	69.52%	73/105	61.11%	44/72	71.42%	15/21	69.9%	72/103
app/(admin)/technique	84.78%	39/46	65%	13/20	76.47%	13/17	86.36%	38/44
app/(tabs)	84.93%	310/365	73.33%	187/255	71.91%	64/89	86.84%	297/342
app/athlete	85.71%	18/21	75%	3/4	87.5%	7/8	85.71%	18/21
app/athlete/plan	88.23%	30/34	58.82%	20/34	80%	8/10	90.32%	28/31
app/athlete/workout	76.31%	116/152	57.28%	118/206	62.16%	23/37	78.72%	111/141
app/coach/athlete	85.71%	36/42	53.84%	28/52	100%	7/7	93.93%	31/33
app/coach/plan	67.71%	151/223	60.1%	110/183	48.83%	21/43	69.66%	147/211
app/coach/workout	74.39%	122/164	62.33%	96/154	39.47%	15/38	78.2%	122/156
constants	100%	3/3	100%	0/0	100%	0/0	100%	3/3
src/components	78.08%	114/146	71.57%	68/95	80%	24/30	79.02%	113/143
src/components/charts	86.95%	20/23	50%	9/18	100%	9/9	85%	17/20

Figura 5.2: Informe de cobertura del conjunto de pruebas de React Testing Library en el cliente móvil, por directorio.

Métrica global	*Backend*	Cliente móvil
Líneas de código	91,5 %	83,7 %
Declaraciones	90,8 %	81,8 %
Funciones	92,6 %	73,1 %
Ramas	74,1 %	68,7 %

Tabla 5.1: Comparativa global de las métricas de cobertura entre el *backend* y el cliente móvil.

Respecto a las pruebas de principio a fin, se han automatizado totalmente tres flujos con Maestro, uno por rol (administrador, entrenador y atleta), replicando de principio a fin todas las acciones esenciales de la lógica de negocio: inicio de sesión, gestión de planes, ejecución de sesiones y comunicación por chat. Al ejecutarlos en un emulador de Android contra el sistema real, se verifica al mismo tiempo el *backend*, la base de datos y la interfaz de usuario.

5.2. Entorno de despliegue y producción

La aplicación se aloja en un servidor de la universidad usando contenedores Docker, un proxy inverso que maneja las conexiones externas y un flujo de integración continua que valida cada *commit* antes de fusionarlo. La Figura 5.3 muestra la topología final mediante un diagrama de despliegue en notación UML.

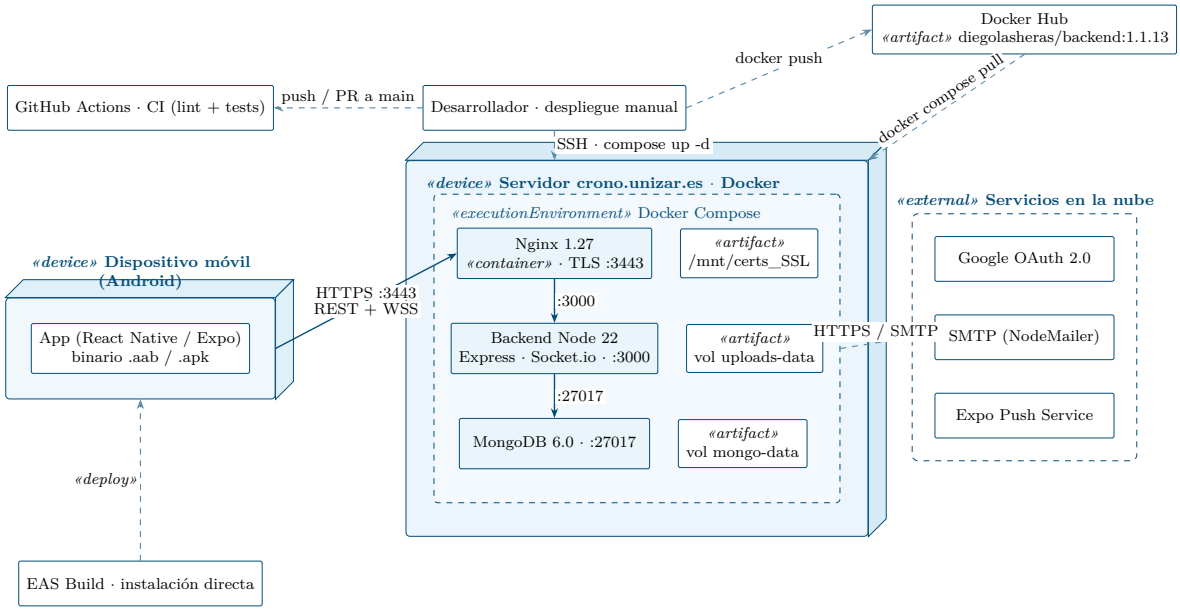

Figura 5.3: Diagrama UML de despliegue de la aplicación en producción sobre el servidor `crono.unizar.es` de la Universidad de Zaragoza.

El proyecto se ha contenedorizado para facilitar su despliegue. Concretamente, se proveen tres contenedores: el *backend* en Node.js, la base de datos MongoDB con su

volumen persistente y un *proxy* inverso Nginx [31], el único servicio expuesto. Nginx maneja las conexiones HTTPS usando certificados SSL/TLS proporcionados por la universidad y redirige las peticiones al *backend*, incluyendo las conexiones WebSocket al chat.

Respecto a la configuración del servidor y otras variables de entorno necesarias, todos los parámetros sensibles (la cadena de conexión para MongoDB, el secreto JWT, las credenciales de correo electrónico y de Google OAuth) se proporcionan a través de un archivo `.env` no incluido en el repositorio. Además, el puerto usado por el servidor en la red pública se proporciona como una variable de entorno para poder coexistir con otros servicios desplegados en el mismo servidor.

El sistema desplegado deja constancia de su propio funcionamiento: los errores no previstos se notifican a un servicio externo de monitorización (Sentry [32]), que conserva la traza completa de cada fallo y avisa por correo electrónico cuando aparece uno nuevo. De este modo, esos fallos del *backend* en producción quedan accesibles desde un único punto sin necesidad de acceder al servidor.

Por último, cada actualización hacia la rama principal activa en GitHub Actions [33] un flujo de trabajo que instala las dependencias, ejecuta el *linter* (ESLint [34]) y el conjunto de pruebas automatizadas, incluyendo un contenedor temporal de MongoDB para el *backend*. De esta forma, se identifican las regresiones antes de fusionar. El despliegue en producción, en cambio, es manual: la imagen del *backend* se construye localmente y se publica en Docker Hub etiquetada con su versión, para que el servidor la descargue después con `docker compose`.

5.3. Organización del código

El código fuente se reparte en dos repositorios independientes, uno por cada componente de la arquitectura cliente-servidor.

El *backend* sigue una organización por capas. El directorio `routes` define los puntos de acceso de la API y asocia a cada uno los *middlewares* de autenticación, autorización y validación. De este modo, las condiciones que debe cumplir quien invoca cada operación quedan declaradas en un único fichero por recurso, y no repartidas por los controladores. El Código 5.1 recoge esta declaración para la biblioteca de fotos de ejercicio.

```
// Lectura: cualquier usuario autenticado
// (el entrenador elige la foto, el atleta la visualiza)
router.get('/', protect, getAllPhotos);

// A partir de aquí, sesión obligatoria y rol de administrador
router.use(protect, restrictTo(ROLES.ADMIN));
router.post('/', validate(createPhotoSchema), createPhoto);
```

Código 5.1: Declaración de rutas de la biblioteca de fotos de ejercicio.

Las peticiones recorren las declaraciones en el orden en que aparecen, de modo que `router.use` aplica la autenticación y la comprobación de rol a todas las operaciones declaradas a partir de esa línea, sin necesidad de repetirlas en cada una. La consulta, declarada antes, queda accesible a cualquier usuario autenticado, mientras que la creación, la modificación y el borrado exigen además el rol de administrador. El nivel de acceso de cada operación depende por tanto de su posición en el fichero, que actúa como declaración explícita de la política de acceso al recurso.

Por su parte, `controllers` gestiona la lógica asociada a las distintas operaciones, mientras que `models` define los esquemas de Mongoose y `schemas` agrupa los esquemas destinados a validar las peticiones entrantes. Además, `tests` y `scripts` contienen, respectivamente, las pruebas y las utilidades destinadas a la carga de datos de prueba. El cliente móvil se estructura en torno al sistema de enrutamiento basado en ficheros de Expo Router. Cada fichero del directorio `app` representa una ruta de la aplicación, y estas se agrupan según el rol que puede acceder a ellas, de manera que la propia jerarquía de directorios refleja el control de acceso por roles del sistema.

El sistema resultante expone 66 puntos de acceso en la API del *backend* y 28 pantallas en el cliente móvil.

Capítulo 6

Conclusiones y trabajo futuro

Este documento recoge el desarrollo completo de una herramienta de gestión deportiva para el centro de escalada Bulderland. Este proceso comprende el análisis y definición de requisitos, su diseño e implementación, y por último las pruebas, validación y despliegue del sistema resultante en un entorno de producción. Partiendo del análisis de los problemas existentes en la comunicación y planificación de los entrenamientos, se ha desarrollado un servidor *backend* y un cliente móvil multiplataforma.

El proyecto actual ha permitido a Bulderland disponer de una plataforma en producción con arquitectura cliente-servidor, con control de acceso basado en roles, comunicación en tiempo real y notificaciones *push*, eliminando así la necesidad de la gestión de mensajería general. La aplicación permite la gestión de planes y sesiones de entrenamiento, un sistema multi-rol con control de acceso basado en roles y por abono, la comunicación en tiempo real con notificaciones *push* y la contenerización con Docker.

En cuanto al trabajo futuro, se plantea la compilación y distribución para iOS. Además, sería necesaria un despliegue en el entorno real de Bulderland con un conjunto de atletas y entrenadores que pongan a prueba las funcionalidades del sistema.

Declaración de uso responsable de IA generativa

El autor declara que se ha utilizado la herramienta de IA generativa Claude (Anthropic), en sus modelos Claude Opus 4.8 y Claude Opus 5, como apoyo en el desarrollo y la depuración de código, la revisión de los diagramas para que cumplan los estándares UML y en la redacción de esta memoria. Toda información o fragmento de código sugerido por la IA ha sido revisado, verificado y validado por el autor antes de su inclusión, siendo el autor plenamente responsable del contenido íntegro del trabajo, incluyendo las secciones en las que se hayan utilizado las herramientas mencionadas como apoyo.

Bibliografía

[1] Tim J. Gabbett. "The training-injury prevention paradox: should athletes be training smarter and harder?" En: *British Journal of Sports Medicine* 50.5 (2016), págs. 273-280. DOI: 10.1136/bjsports-2015-095788.

[2] Anna E. Saw, Luana C. Main y Paul B. Gastin. "Monitoring athletes through self-report: factors influencing implementation". En: *Journal of Sports Science and Medicine* 14.1 (2015), págs. 137-146. URL: https://pubmed.ncbi.nlm.nih.gov/25729301/ (visitado 19-06-2026).

[3] Louise Davis, Sophia Jowett y Susanne Tafvelin. "Communication Strategies: The Fuel for Quality Coach-Athlete Relationships and Athlete Satisfaction". En: *Frontiers in Psychology* 10, 2156 (2019). DOI: 10.3389/fpsyg.2019.02156.

[4] Strava, Inc. *Strava: Aplicación de seguimiento de actividad deportiva*. URL: https://www.strava.com (visitado 19-06-2026).

[5] MyFitnessPal, Inc. *MyFitnessPal: Seguimiento de nutrición y actividad física*. URL: https://www.myfitnesspal.com (visitado 19-06-2026).

[6] Peaksware, LLC. *TrainingPeaks: Plataforma de entrenamiento y preparación deportiva*. URL: https://www.trainingpeaks.com (visitado 19-06-2026).

[7] TrueCoach, Inc. *TrueCoach: Software para entrenadores personales*. URL: https://truecoach.co (visitado 19-06-2026).

[8] Lattice Training Ltd. *Lattice Training: Entrenamiento de escalada basado en datos*. URL: https://latticetraining.com (visitado 19-06-2026).

[9] TopLogger. *TopLogger: Registro de vías y bloques en rocódromos*. URL: https://toplogger.nu (visitado 19-06-2026).

[10] WhatsApp LLC. *WhatsApp: Aplicación de mensajería*. URL: https://www.whatsapp.com (visitado 19-06-2026).

[11] Meta Platforms, Inc. *React Native: Framework para el desarrollo de aplicaciones nativas con React*. URL: https://reactnative.dev (visitado 19-06-2026).

[12] Google LLC. *Flutter: Desarrollo de aplicaciones multiplataforma*. URL: https://flutter.dev (visitado 19-06-2026).

[13] Expo. *Expo: Plataforma de desarrollo para aplicaciones React Native*. URL: https://expo.dev (visitado 19-06-2026).

[14] Django Software Foundation. *Django: Framework de desarrollo web en Python*. URL: https://www.djangoproject.com (visitado 19-06-2026).

[15] Broadcom, Inc. *Spring Boot: Framework de desarrollo de aplicaciones en Java*. URL: https://spring.io/projects/spring-boot (visitado 19-06-2026).

[16] OpenJS Foundation. *Node.js: Entorno de ejecución de JavaScript con E/S no bloqueante orientada a eventos*. URL: https://nodejs.org/en/about (visitado 19-06-2026).

[17] OpenJS Foundation. *Express: Framework web minimalista para Node.js*. URL: https://expressjs.com (visitado 19-06-2026).

[18] MongoDB, Inc. *MongoDB: Base de datos NoSQL orientada a documentos*. URL: https://www.mongodb.com (visitado 19-06-2026).

[19] Google LLC. *Firebase: Plataforma de desarrollo de aplicaciones de Google*. URL: https://firebase.google.com (visitado 19-06-2026).

[20] Socket.IO. *Socket.IO: Comunicación bidireccional en tiempo real basada en eventos*. URL: https://socket.io (visitado 19-06-2026).

[21] Docker, Inc. *Docker: Plataforma de contenedores*. URL: https://www.docker.com (visitado 19-06-2026).

[22] Mobile.dev. *Maestro: Framework de pruebas de extremo a extremo para móviles*. URL: https://maestro.mobile.dev (visitado 19-06-2026).

[23] Michael Jones, John Bradley y Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519. Internet Engineering Task Force, 2015. URL: https://www.rfc-editor.org/rfc/rfc7519 (visitado 19-06-2026).

[24] Colin McDonnell. *Zod: Validación de esquemas para TypeScript con inferencia estática de tipos*. URL: https://zod.dev (visitado 19-06-2026).

[25] Automattic. *Mongoose: Modelado de objetos de MongoDB para Node.js*. URL: https://mongoosejs.com (visitado 30-08-2026).

[26] Google LLC. *Google Identity: Autenticación mediante OAuth 2.0*. URL: https://developers.google.com/identity/protocols/oauth2 (visitado 19-06-2026).

[27] Andris Reinman. *Nodemailer: Envío de correo electrónico desde Node.js*. URL: https://nodemailer.com (visitado 30-08-2026).

[28] OpenJS Foundation. *Jest: Framework de pruebas en JavaScript*. URL: https://jestjs.io (visitado 19-06-2026).

[29] TJ Holowaychuk. *SuperTest: Librería para pruebas de peticiones HTTP en Node.js*. URL: https://github.com/ladjs/supertest (visitado 19-06-2026).

[30] Testing Library. *React Testing Library: Utilidades de pruebas centradas en el comportamiento del usuario*. URL: https://testing-library.com/docs/react-testing-library/intro (visitado 19-06-2026).

[31] F5, Inc. *NGINX: Servidor web y proxy inverso*. URL: https://nginx.org (visitado 19-06-2026).

[32] Functional Software, Inc. *Sentry: Monitorización de errores y rendimiento en aplicaciones*. URL: https://sentry.io (visitado 05-09-2026).

[33] GitHub, Inc. *GitHub Actions: Automatización de flujos de integración y despliegue continuo*. URL: https://docs.github.com/actions (visitado 19-06-2026).

[34] OpenJS Foundation. *ESLint: Analizador estático de código para JavaScript*. URL: https://eslint.org (visitado 01-09-2026).

Anexo A

Planificación y distribución temporal

Este anexo resume la organización temporal del desarrollo y las fases en las que se ha dividido el trabajo.

El desarrollo se ha llevado a cabo entre febrero y agosto de 2026, con una dedicación total aproximada de 350 horas. La Tabla A.1 distribuye esas 350 horas por tipo de actividad: la programación concentra el grueso del esfuerzo, seguida de la redacción de la memoria, las pruebas, el estudio previo y el diseño, con una parte menor dedicada al seguimiento con el tutor.

Categoría	**Horas**	**Porcentaje**
Programación (backend y móvil)	180,0	51,4 %
Redacción de la memoria	60,0	17,1 %
Pruebas	58,0	16,6 %
Estudio previo	21,0	6,0 %
Análisis y diseño	29,0	8,3 %
Reuniones con el tutor	2,0	0,6 %
Total	**350,0**	**100,0 %**

Tabla A.1: Distribución estimada del esfuerzo por tipo de actividad.

El trabajo se ha organizado en siete fases que se solapan parcialmente en el tiempo, tal como recoge la Figura A.1 en un diagrama de Gantt.

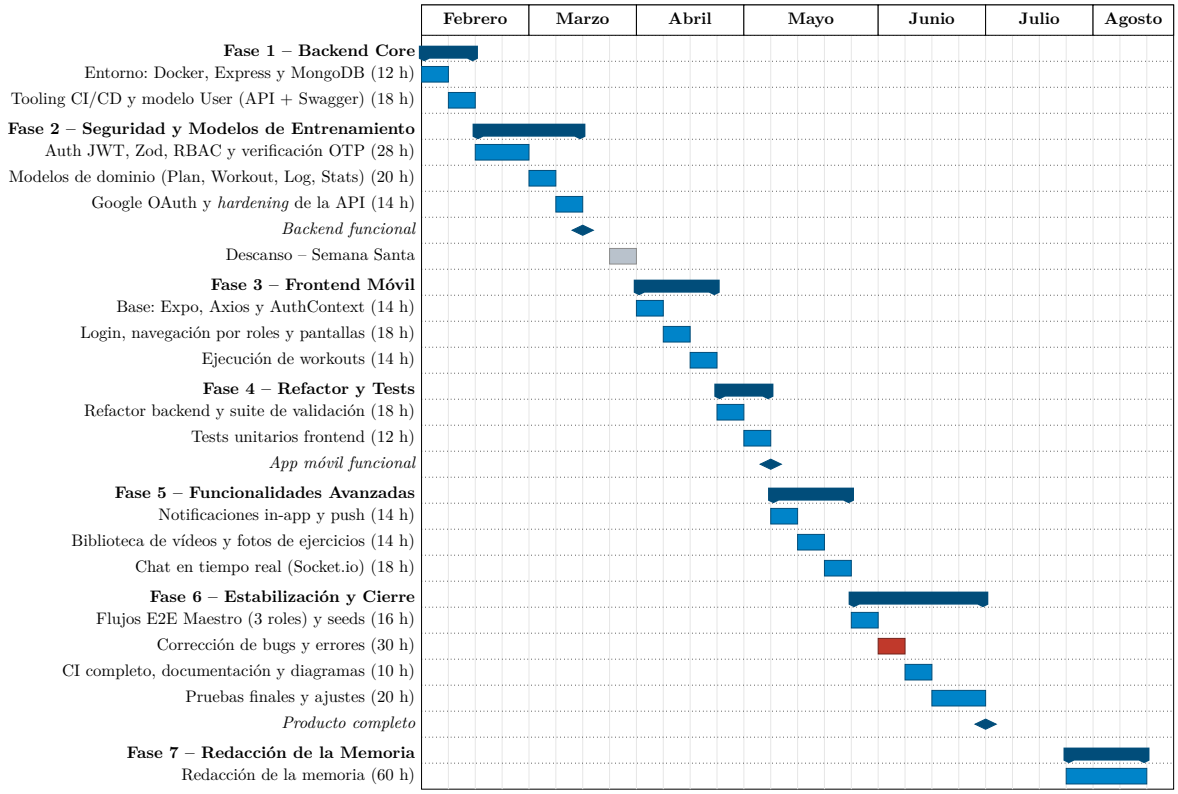

Figura A.1: Diagrama de Gantt del cronograma de desarrollo del proyecto (350 horas).