FIT4U: Conectando Fitness y Entrenamiento Personalizado

Samuel Vitales Ponce

1603354

Resumen — Este trabajo de final de grado (TFG) presenta FIT4U, una aplicación integral para el entrenamiento físico y mental que conecta usuarios con entrenadores especializados. La plataforma incluye planificación de entrenamientos, seguimiento personalizado y chat en tiempo real. Utilizando metodologías Scrum y DevOps, con NodeJS y PostgreSQL, el proyecto ha resultado altamente satisfactorio para el cliente. FIT4U inspira un estilo de vida activo y saludable, proporcionando las herramientas para alcanzar los objetivos de manera cómoda. Este proyecto representa una oportunidad de crecimiento en desarrollo de software en el sector del fitness.

Palabras clave — FIT4U, entrenamiento físico, entrenamiento mental, entrenadores especializados, planificación de entrenamientos, seguimiento personalizado, gamificación, chat en tiempo real, Scrum, DevOps, NodeJS, PostgreSQL, desarrollo de software, fitness.

Abstract — This Final Degree Project (TFG) presents FIT4U, a comprehensive application for physical and mental training that connects users with specialised trainers. The platform includes training planning, personalised monitoring and real-time chat. Using Scrum and DevOps methodologies, with NodeJS and PostgreSQL, the project has been highly satisfactory for the client. FIT4U inspires an active and healthy lifestyle, providing the tools to achieve goals in a comfortable way. This project represents an opportunity for growth in software development in the fitness industry.

Index Terms — FIT4U, physical training, mental training, specialized trainers, workout planning, personalized tracking, gamification, real-time chat, Scrum, DevOps, NodeJS, PostgreSQL, software development, fitness.

1 Introducción – Contexto del trabajo

Hoy en día, la salud y el bienestar ocupan un lugar muy importante en la vida de las personas, el ejercicio físico y mental se ha convertido en una prioridad para muchos [1].

FIT4U emerge como una respuesta a esta necesidad creciente, ofreciendo una aplicación integral diseñada para ayudar y potenciar la experiencia de entrenamiento de cada usuario, conectando entrenadores con ellos.

No sé trata solamente de una aplicación de ejercicio; es un sistema completo que abarca desde la planificación de entrenamientos hasta el seguimiento individual de cada usuario. Además, tiene una amplia gama de características innovadoras; reservas en tiempo real, integración con redes sociales e incluso un chat en tiempo real con un entrenador especializado en ese ámbito.

La motivación detrás de este proyecto surge de ofrecer una plataforma que inspire a las personas a llevar un estilo de vida más activo y saludable, además de brindar las herramientas para poder alcanzar sus objetivos de forma cómoda.

Para mí, FIT4U representa más que un simple proyecto; es una oportunidad para aprender y crecer como desarrollador. La idea de poder trabajar estrechamente con un cliente me parece emocionante y se acerca a la realidad diaria de muchos trabajos. Además, el hecho de que este proyecto se enfoque en un ámbito que me apasiona, el *fitness*, significa que no solo ganaré experiencia técnica, sino que también disfrutaré del proceso.

2 Objetivos

El objetivo general del proyecto es el siguiente:

- Desarrollar una aplicación integral de ejercicio que brinde a los usuarios una experiencia completa y personalizada para mejorar su salud y bienestar a través del ejercicio físico.

- *E-mail de contacte: svitalesponce@gmail.com*
- *Menció realitzada: Enginyeria del Software*
- *Treball tutoritzat per: Ana Oropesa Física – Departamento de Ingeniería de la Información y de las Comunicaciones*
- *Curs 2023/24*

El proyecto está dividido en dos partes; front-end y back-end. En mi caso, me hago cargo del back-end y por ende, los objetivos específicos son los siguientes:

- Reducir al 90% los accesos no autorizados en un plazo de 1 mes mediante la implementación sistema de autenticación.
- Implementar un sistema de gestión de perfiles que almacene información relevante del usuario, como objetivos de *fitness*, historial de entrenamientos y preferencias de contenido, durante la fase de desarrollo.
- Diseñar un algoritmo de planificación de entrenamientos que genere programas personalizados según los objetivos y el nivel de condición física del usuario, dentro de un plazo de 2 semanas desde el inicio del diseño y desarrollo del algoritmo.
- Crear un sistema de registro de actividades que permita a los usuarios registrar sus sesiones de entrenamiento y otras actividades físicas de manera fácil y rápida, dentro de un plazo de 1 semana durante la fase de desarrollo.
- Implementar un mecanismo para almacenar y analizar los datos de actividad del usuario, permitiendo un seguimiento detallado del progreso a lo largo del tiempo, dentro de un plazo de 6 semanas desde el inicio del diseño e implementación del mecanismo.
- Implementar un sistema de reservas que permita a los usuarios programar y gestionar citas para clases y eventos dentro de la aplicación en un plazo de 3 semanas desde el inicio del desarrollo del sistema, asegurando una interfaz intuitiva y fácil de usar para los usuarios.
- Desarrollar e implementar un sistema de notificaciones y recordatorios que mantenga a los usuarios informados sobre sus actividades programadas dentro de la aplicación, garantizando una entrega oportuna y confiable de las notificaciones en un plazo de 4 días desde el inicio del desarrollo.
- Integrar un sistema de chat en tiempo real dentro de la aplicación que permita a los usuarios comunicarse con entrenadores personales de manera efectiva y segura, asegurando una experiencia de usuario fluida y sin interrupciones en un plazo de 2 semanas desde el inicio del desarrollo del sistema.
- Diseñar, desarrollar e implementar desafíos y sistemas de recompensas dentro de la aplicación que motiven a los usuarios a alcanzar sus objetivos de fitness, garantizando una variedad de desafíos y recompensas significativas en un plazo de 7 meses desde el inicio del diseño y desarrollo.
- Integrar sistemas de puntuación y tablas de clasificación en la aplicación para fomentar una competencia amistosa entre los usuarios, asegurando una presentación clara y actualizada de las puntuaciones y clasificaciones en un plazo de 5 meses desde el inicio de la integración.
- Desarrollar e implementar un motor de búsqueda robusto dentro de la aplicación que permita a los usuarios encontrar fácilmente rutinas de entrenamiento basadas en diversos criterios, asegurando resultados precisos y relevantes en un plazo de 6 días desde el inicio del desarrollo.
- Diseñar, desarrollar e implementar filtros y etiquetas dentro del motor de búsqueda de la aplicación para mejorar la precisión de los resultados de búsqueda, asegurando una navegación y selección de rutinas de entrenamiento más eficiente durante la fase de desarrollo.
- Desarrollar e implementar funciones de interacción social dentro de la aplicación que permitan a los usuarios compartir su progreso, logros y experiencias en plataformas de redes sociales, garantizando una integración fluida y segura en un plazo de 3 semanas desde el inicio del desarrollo.
- Integrar API de redes sociales dentro de la aplicación para facilitar la conexión y la integración con plataformas populares como Facebook, Instagram y Twitter, asegurando una funcionalidad estable y segura antes de acabar la fase de desarrollo.

3 PLANIFICACIÓN

Para realizar la planificación del proyecto, hemos utilizado un diagrama de Gantt [2] que detalla las fases de nuestro proyecto y las tareas a cumplir en cada una de las fases. Las fases son las siguientes:

Planificación: Esta fase comenzó el 22 de febrero de 2024 y finalizará el 1 de abril de 2024. Durante esta fase, se definirán los requisitos del proyecto y se desarrollará un plan de proyecto.

Desarrollo: Esta fase comenzará el 7 de abril de 2024 y finalizará el 1 de mayo de 2024. Durante esta fase, se desarrollará la aplicación.

Pruebas: Esta fase comenzará el 21 de abril de 2024 y finalizará el 10 de mayo de 2024. Durante esta fase, se

probará la aplicación para asegurarse de que cumple con los requisitos.

Conclusiones: Esta fase comenzará el 31 de mayo de 2024 y finalizará el 1 de junio de 2024. Durante esta fase, se documentarán las lecciones aprendidas del proyecto y se finalizará el proyecto.

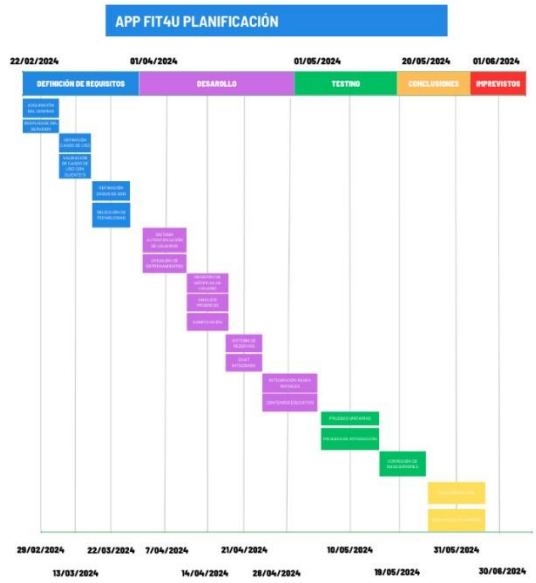


Figura 1: Diagrama de Gantt

4 Metodología

Para el desarrollo de FIT4U, se va a utilizar metodologías ágiles y prácticas de desarrollo modernas para garantizar un desarrollo eficiente, organizado y de calidad. Para ello, se va a estar utilizando dos enfoques claves, Scrum [3] y DevOps [4].

Scrum permitirá dividir el proyecto en iteraciones cortas y manejables, también conocidas como sprints. Cada sprint comienza con una reunión de planificación en la que se establecen las tareas a llevar a cabo. Durante el sprint, se llevan a cabo reuniones de seguimiento. Al final de cada sprint, se revisa el trabajo completado y se realiza una retrospectiva.

Por otro lado,se utilizará DevOps. DevOps es una cultura y conjunto de prácticas que integran el desarrollo de software (Dev) y las operaciones (Ops) con el objetivo de acortar el ciclo de vida de desarrollo, mejorar la colaboración y garantizar la entrega continua de software de alta calidad.

Para llevar a cabo este despliegue automático,se utilizará las siguientes herramientas; Docker [5] y Amazon Web Services [6]. Al utilizar Docker y AWS en conjunto, podemos crear un entorno de desarrollo y despliegue altamente automatizado y escalable. Esto permite iterar rápidamente en nuestra aplicación, garantizando una entrega continua de nuevas características y actualizaciones a nuestros usuarios finales.

5 Estado Del Arte

En esta sección, se detalla el mercado actual de aplicaciones de fitness y salud y se explica por qué FIT4U destaca sobre el resto. Además, se habla de las tecnologías utilizadas, la base de datos y los casos de uso.

5.1 MOTIVACIÓN

Hoy en día, hay muchas aplicaciones web de fitness y salud, desde plataformas de seguimiento de actividad física hasta aplicaciones de entrenamiento personalizado. Strava es un claro ejemplo, muy popular entre los entusiastas del deporte, que permite a los usuarios registrar actividades, establecer metas y conectarse con otros atletas.

Otro claro ejemplo es MyFitnessPal, una aplicación para el seguimiento de la ingesta de alimentos y el ejercicio. MyFitnessPal ofrece herramientas para establecer objetivos de pérdida de peso, seguimiento de calorías y macronutrientes, y registro de entrenamientos.

Por otro lado, también existe Runtastic, la cual ofrece seguimiento de actividad física, entrenamientos y ejercicios.

Las aplicaciones mencionadas anteriormente y similares no ofrecen una experiencia completamente personalizada. Por eso, este trabajo se centra en brindar una personalización más específica para los usuarios.

La aplicación permite realizar reservas en diferentes sesiones, seleccionar diversos entrenamientos y mantener un contacto cercano con los entrenadores a través de un chat integrado.

Esto proporciona una plataforma inspiradora y funcional para promover un estilo de vida activo y saludable. Se han aplicado diversas tecnologías y enfoques para el desarrollo del back-end de FIT4U, con el objetivo de crear una plataforma robusta y eficiente.

5.2 TECNOLOGÍAS

En primer lugar, se ha seleccionado NodeJS con el framework Express debido a su amplia adopción en la industria y su capacidad para construir aplicaciones de manera rápida y escalable.

Además, se ha adoptado una arquitectura hexagonal [7] de tres capas (dominio, aplicación e infraestructura) y se ha aplicado el principio de vertical slicing [8] para organizar el código alrededor de las características y funcionalidades de la aplicación.

A diferencia de otras arquitecturas, por ejemplo, Modelo-Vista-Controlador (MVC) [9], Modelo-Vista-VistaModelo (MVVM) [10] o flujo de datos unidireccional (Flux) [11], esta arquitectura permite modularizar, por tanto, mejora la

comprensión y el mantenimiento del código, así como la integración de nuevas características en un futuro.

El enfoque de *Domain Driven Design* (*DDD*) [12] ha sido fundamental para modelar el dominio de la aplicación y para activar servicios/acciones en segundo plano mediante el lanzamiento de eventos específicos.

Se ha optado por *Domain Driven Design* en lugar de otros enfoques como *CQRS*, *ES* o *Event Storming*, debido a su focalización en el dominio. Además, promueve el uso de un lenguaje común que ayuda a cerrar la brecha entre el negocio y la tecnología. Por último, fomenta un enfoque práctico y empírico basado en la retroalimentación continua y el aprendizaje mediante la implementación y la iteración.

Por último, para habilitar interacción en tiempo real entre los usuarios de la aplicación, se ha integrado el uso de sockets [13], habilitando una comunicación bidireccional entre el cliente y el servidor.

En términos de seguridad, se han implementado diversas medidas, incluyendo el cifrado de datos, la limitación de peticiones y el uso de tokens para autenticación y autorización. Además, se ha adoptado el principio de minimizar las respuestas a los datos justos y necesarios, con el fin de reducir la exposición de información sensible y proteger la privacidad de los usuarios.

5.3 BASE DE DATOS

En cuanto a la persistencia de datos, se evaluaron varias alternativas para el sistema de gestión de bases de datos. Entre las opciones consideradas estaban MongoDB, Aurora SQL y otras más. Sin embargo, tras un análisis detallado, se optó por utilizar PostgreSQL debido a su robustez, escalabilidad y capacidades avanzadas. Además, se destacó su excepcional rendimiento en la manipulación de grandes volúmenes de datos, tanto en operaciones de escritura como de lectura. Antes de iniciar el desarrollo, se realizó un diseño detallado de la base de datos, el cual puede visualizarse en el "Apéndice C".

5.4 CASOS DE USO

Antes de iniciar el desarrollo, se diseñaron casos de uso específicos de la aplicación FIT4U en colaboración con el cliente. Estos casos de uso fueron cuidadosamente validados con el cliente para garantizar que capturaran con precisión las necesidades y expectativas del usuario final.

Los casos de uso son los siguientes:

- Caso de uso chat
- Caso de uso entrenamientos
- Caso de uso gamificación
- Caso de uso inicio de sesión y registro
- Caso de uso perfiles de usuarios
- Caso de uso progreso
- Caso de uso reservas

Se pueden visualizar en el "Apéndice A".

6 DESARROLLO

En esta sección, se detalla el proceso de la aplicación de FIT4U y, además, se describe la implementación de sus funcionalidades.

6.1 CONFIGURACIÓN DEL SERVIDOR

Para el entorno de producción, el equipo ha optado por alojar el servidor en una instancia EC2 de AWS (Amazon Web Services). Esta elección se debe a la flexibilidad y confiabilidad que ofrece EC2 para ejecutar la aplicación en línea.

Para desplegar el servidor en la instancia EC2, se ha configurado un flujo de trabajo automatizado, haciendo uso de GitHub Actions. Esto permite realizar despliegues continuos de la aplicación cada vez que se realicen cambios en el repositorio de GitHub, garantizando que siempre se ejecute la versión más reciente del código en el entorno de producción.

Durante el desarrollo, se ha empleado el entorno local para ejecutar el servidor en localhost:3000, lo cual ha proporcionado un entorno controlado para realizar pruebas exhaustivas y desarrollar de manera eficiente. Para validar las diversas funcionalidades y *endpoints* de la API, se ha utilizado Postman, una herramienta que facilita la ejecución de solicitudes HTTP y la revisión de respuestas de forma rápida y sencilla. Utilizando este enfoque con localhost y Postman, se asegura que todas las APIs funcionen correctamente antes de implementarlas en el entorno de producción.

6.2 CONFIGURACIÓN DE LA BASE DE DATOS

Para iniciar el desarrollo de las funcionalidades de la aplicación, se ha realizado la configuración de la base de datos. Esta base de datos es una instancia de PostgreSQL alojada en AWS (Amazon Web Services). La decisión de alojarla en AWS permite un mayor control sobre las peticiones entrantes y salientes.

La instancia de la base de datos en AWS tiene configuradas las reglas de entrada de manera que están limitadas únicamente a la IP de la instancia EC2 donde se encuentra alojado el servidor, así como a la IP del *tester* del servidor. Esto proporciona una capa adicional de seguridad al restringir el acceso a la base de datos solo a las direcciones IP autorizadas.

Esta configuración asegura que la base de datos esté protegida contra accesos no autorizados y garantiza un

entorno seguro para el almacenamiento y acceso a los datos de la aplicación.

El diseño de la base de datos se ha realizado de esta forma para asegurar un almacenamiento eficiente y seguro de los datos de la aplicación, teniendo en cuenta las relaciones entre las entidades del servidor. Este modelado se ha basado en casos de uso relevantes para garantizar que la estructura de la base de datos se adapte a las necesidades específicas de la aplicación.

6.3 GESTIÓN DE USUARIOS

La gestión de usuarios abarca aspectos como la autenticación, autorización y actualización de datos personales.

Para la autenticación y autorización, se ha empleado *bcryptjs* y *jsonwebtoken*. Al darse de alta, se reciben los datos del usuario, se realizan comprobaciones como la existencia del correo electrónico y se encripta la contraseña antes de insertarla en la base de datos. Se puede visualizar la función de encriptación en el "Apéndice B".

Después de insertar al usuario en la base de datos, se genera un token de acceso seguro para identificar al usuario en las peticiones y verificar su validez y permisos. Para realizar la generación del token, se ha utilizado una función específica. Dicha función, se puede visualizar en el "Apéndice B".

La respuesta al cliente oculta ciertos valores sensibles del usuario. Al iniciar sesión nuevamente, se reciben solo el correo electrónico y la contraseña en la solicitud. Para verificar el inicio de sesión, se comprueba la existencia del correo electrónico y se compara la contraseña recibida con la almacenada en la base de datos.

Se han creado dos middlewares para gestionar la autenticación y autorización de las peticiones. Uno verifica la validez del token y el otro los permisos del usuario (entrenador o deportista), asegurando el acceso solo a usuarios autorizados. Se utiliza *jsonwebtoken* para verificar el token.

Si es válido, se verifica el tipo de cuenta y sus permisos. Cuando se recibe el *payload* del middleware anterior, se comprueba si el tipo de cuenta es adecuado. Si es correcto, se procede con la petición; de lo contrario, se genera un error. Las funciones se detallan en el "Apéndice B".

6.4 GESTIÓN DE SESIONES Y RESERVAS

El sistema debe permitir crear, ver, editar y eliminar sesiones y sus reservas asociadas. Al crear o editar una sesión, se reciben los datos necesarios en el cuerpo de la solicitud. Una vez recibidos, se lleva a cabo la consulta correspondiente en la base de datos.

Para las solicitudes de tipo *GET*, que implican operaciones de *JOIN* para recuperar datos más complejos, como todas las sesiones, se ejecuta una consulta específica. Esta consulta se puede visualizar en el "Apéndice B".

Se ha completado el desarrollo de la entidad de sesiones y se ha avanzado con la entidad de reservas. Para crear una reserva, en la solicitud se requieren ciertos datos, siendo el campo "status" opcional en este caso. Antes de crear una reserva, se realizan ciertas comprobaciones, la existencia de la sesión y además, la capacidad de la sesión.

Una vez completadas las comprobaciones, se actualizan los datos de la sesión y se inserta la nueva reserva en la base de datos.

La entidad de reserva también aborda casos de uso específicos, como la cancelación de una reserva por parte de un usuario. Durante esta petición, se recibe el ID de la reserva. Posteriormente, al recuperar la reserva de la base de datos, se actualiza su estado a "cancelada" y se ajustan los valores de la sesión correspondiente.

6.5 GESTIÓN DE ENTRENAMIENTOS

Se reconoció la gestión de entrenamientos como una parte fundamental de la aplicación, permitiendo a los entrenadores crear entrenamientos personalizados para los deportistas.

Para iniciar este proceso, se desarrolló la entidad de "Exercise", la cual solo podía ser creada por entrenadores. Los datos necesarios para crear un ejercicio son los siguientes: nombre, descripción y la URL de un vídeo demostrativo del ejercicio.

Por otro lado, para el desarrollo de la entidad de "Workout", se tuvo en cuenta que estos dependen de los ejercicios. Por ende, se definió en la base de datos la relación "workouts_exercises", donde por cada entrenamiento se guarda el ejercicio asociado junto con otras características, como el número de repeticiones, el tiempo de descanso y el número de series.

Además, se definieron casos de uso específicos para los deportistas. Por ejemplo, tenían la capacidad de apuntarse/asignarse a un entrenamiento específico o de añadir una calificación al mismo.

Para el primer caso de uso, se creó una relación llamada "users_workouts" en la base de datos, donde se guardaban el ID del usuario y el ID del entrenamiento al que se habían apuntado/asignado. Se puede visualizar la función y la consulta asociada en el "Apéndice B".

Por otro lado, al añadir una calificación, se realizaron más acciones. En primer lugar, se recuperó el entrenamiento para obtener la calificación actual y el número total de calificaciones. Una vez hecho esto, se calculó la nueva calificación y se actualizaron los datos en la base de datos.

Se puede visualizar la función y la consulta asociada en el "Apéndice B".

Por último, se ha implementado la creación de entrenamientos privados para usuarios. Se ha implementado un nuevo atributo dentro de la entidad "Workout". Este atributo es "assigned". Refleja si un entrenamiento ha sido asignado a un deportista en concreto; si es el caso, este solo es visible para él.

6.6 SISTEMA DE MENSAJERÍA

El sistema de mensajería se destacó como la parte más compleja de la aplicación. Requería ser en tiempo real y lo más rápido posible. Por esa razón, se optó por implementar la tecnología de sockets junto con la arquitectura *DDD*.

Para comenzar se modeló todo el sistema *DDD*. Primero se detalló la interfaz de un evento dentro de la aplicación. Seguidamente, se desarrolló el servicio de eventos, para poder escucharlos y emitirlos. Se puede visualizar esta información en el "Apéndice B".

Posteriormente, se hizo uso de la librería *Socket.IO* [14] para convertir la aplicación en un servidor y aprovechar las conexiones bidireccionales. La configuración de *Socket.IO* se puede encontrar en el "Apéndice B".

Las configuraciones anteriores solo se han utilizado para la entidad de mensaje. Se consideró que, para crear o listar conversaciones, no era necesario utilizar *sockets*.

Para crear o enviar un mensaje, se debe recibir el evento "messageCreated" desde el front-end. Una vez recibido, se procesan los datos del evento, que deben incluir *user_id*, *conversation_id* y *content* del mensaje.

Esta información se procesa mediante el servicio de la entidad. Una vez que se ha insertado en la base de datos, se genera un evento "messageSended" desde el back-end para enviar el mensaje al destinatario. Este evento contiene información sobre el mensaje, la conversación, el emisor y el destinatario.

Por último, se han realizado modificaciones en la raíz del proyecto para el uso del servidor *http* y no de la propia aplicación de *Express*.

6.7 SISTEMA DE BÚSQUEDA Y FILTRADO

Durante el desarrollo del sistema, se ha decidido crear un mecanismo de filtrado de los entrenamientos. Se han considerado dos opciones para el filtrado:

1. Recuperar todos los entrenamientos y filtrar en memoria.
2. Recuperar los entrenamientos mediante consultas complejas de *SQL*.

Para ambos casos se utiliza el patrón de diseño Criteria, también conocido como *Filter*, pero cada opción tiene un enfoque distinto. En el primer caso, actúa como un filtro, mientras que en el segundo caso, actúa como un *DTO* (*Data Transfer Object*).

Para el desarrollo del sistema, se ha optado por la segunda opción, ya que se adapta y escala mejor dentro de la arquitectura.

Para comenzar, se ha definido la estructura de nuestro *Criteria* y todos sus componentes, como filtros, operadores, etc. La estructura se puede ver en el "Apéndice B".

Se ha desarrollado un servicio llamado Criteria. Este servicio tiene la función de construir el *Criteria* a partir de una consulta recibida desde el cliente.

La importancia de este servicio radica en que permite manejar los datos para el filtrado de manera general, sin necesidad de conocer específicamente los datos que se reciben del lado del cliente. Esto significa que el servicio es capaz de interpretar la consulta y construir el *Criteria* correspondiente, sin depender de la estructura interna de los datos.

Esta transformación proporciona flexibilidad al sistema, ya que, si en el futuro se necesitan añadir nuevos filtros o modificar los existentes, simplemente se deben actualizar las reglas de construcción del *Criteria* en el servicio. De esta manera, el sistema puede escalar eficientemente para adaptarse a nuevas necesidades de filtrado sin requerir cambios extensos en el código existente.

Utilizando los datos de la estructura generada anteriormente (el *Criteria*), el servicio genera las consultas necesarias para filtrar los datos en la base de datos. Esto implica traducir las condiciones de filtrado definidas en el *Criteria* en cláusulas *SQL* que la base de datos pueda entender.

En resumen, el servicio Criteria simplifica el proceso de filtrado al abstraer la lógica de construcción de consultas y proporcionar una forma genérica de agregar funcionalidades de filtrado a diversas entidades del sistema. Esto promueve la reutilización del código y facilita el mantenimiento y la escalabilidad del sistema en términos de capacidades de filtrado. Ambas funciones se pueden ver en el "Apéndice B".

7 RESULTADOS

Para la valoración de los resultados, debemos tener en cuenta dos aspectos: la valoración final del cliente y el cumplimiento de los objetivos propuestos durante la planificación del proyecto. A continuación, se enumerarán los objetivos para evaluar si se han cumplido y qué resultados han tenido.

1. Reducir al 90% los accesos no autorizados en un plazo de 1 mes mediante la implementación de un sistema de autenticación.

El resultado es que se ha implementado un sistema de autorización y autentificación que ha reducido el 95% de los accesos no deseados.

El cliente comentó que la funcionalidad era correcta y que el diseño del sistema, simple y minimalista pero con sus colores identificativos, era de su agrado. Además, valoró muy positivamente el proceso de registro, destacando la verificación en tiempo real de la contraseña y los mensajes informativos ante posibles errores.

Sin embargo, el único punto negativo fue que cualquier persona podía crear una cuenta de entrenador y obtener todos los permisos y funcionalidades sin ninguna comprobación.

2. Implementar un sistema de gestión de perfiles que almacene información relevante del usuario, como objetivos de fitness, historial de entrenamientos y preferencias de contenido, durante la fase de desarrollo.

El resultado ha sido satisfactorio. Se ha desarrollado un sistema capaz de gestionar usuarios, almacenando la información necesaria y relevante de estos.

El cliente valoró positivamente la funcionalidad, destacando la rapidez y eficiencia en la visualización de reservas y entrenamientos dentro del perfil. También apreció el sistema de validación de contraseñas durante la modificación de datos.

3. Crear un sistema de registro de actividades que permita a los usuarios registrar sus sesiones de entrenamiento y otras actividades físicas de manera fácil y rápida, dentro de un plazo de 7 días durante la fase de desarrollo.

El sistema de registro de actividades fue implementado en 7 días. Los usuarios pueden registrar sus sesiones de entrenamiento de manera sencilla y rápida.

4. Implementar un sistema de reservas que permita a los usuarios programar y gestionar citas para clases y eventos dentro de la aplicación en un plazo de 3 semanas desde el inicio del desarrollo del sistema, asegurando una interfaz intuitiva y fácil de usar para los usuarios.

El sistema de reservas ha sido implementado dentro del período de tiempo esperado, ofreciendo una interfaz intuitiva y sencilla de usar.

Estas funcionalidades permiten crear sesiones y entrenamientos, así como reservar o unirse a ellos. El cliente destacó positivamente el buscador de ejercicios, la visualización de las imágenes de los ejercicios y el diseño de las tarjetas de entrenamiento.

5. Integrar un sistema de chat en tiempo real dentro de la aplicación que permita a los usuarios comunicarse con entrenadores personales de manera efectiva y segura, asegurando una experiencia de usuario fluida y sin interrupciones en un plazo de 2 semanas desde el inicio del desarrollo del sistema.

El sistema de chat en tiempo real ha sido implementado de manera satisfactoria, tal como se menciona en el apartado de desarrollo, en el punto 6.6.

El cliente valoró positivamente el diseño, el chat en tiempo real y el buscador para crear conversaciones. Sin embargo, señaló como negativo no recibir notificaciones desde otras partes de la aplicación.

6. Desarrollar e implementar un motor de búsqueda robusto dentro de la aplicación que permita a los usuarios encontrar fácilmente rutinas de entrenamiento basadas en diversos criterios, asegurando resultados precisos y relevantes en un plazo de 6 días desde el inicio del desarrollo.

7. Diseñar, desarrollar e implementar filtros y etiquetas dentro del motor de búsqueda de la aplicación para mejorar la precisión de los resultados de búsqueda, asegurando una navegación y selección de rutinas de entrenamiento más eficiente durante la fase de desarrollo.

El sistema de búsqueda ha sido desarrollado de forma satisfactoria, permitiendo a los usuarios encontrar entrenamientos adecuados de manera sencilla y cumpliendo con el plazo establecido.

El cliente destacó que las imágenes y vídeos de la aplicación eran simples y fáciles de entender. Sin embargo, comentó que se podría haber añadido un apartado de consejos o preguntas frecuentes.

Además de la valoración y los comentarios mencionados, el cliente sugirió añadir un filtro por nombre de entrenamientos y un campo de tiempo, además de los filtros de series y repeticiones ya existentes.

8 Conclusiones

Durante todo el trabajo, el objetivo principal ha sido desarrollar una aplicación integral de ejercicio que brinde a los usuarios una experiencia completa y personalizada para mejorar su salud y bienestar a través del ejercicio físico. Para alcanzar este objetivo, se han utilizado diversas tecnologías y arquitecturas mencionadas en los apartados anteriores.

A lo largo del desarrollo, se implementaron características clave como la personalización de rutinas, el

seguimiento en tiempo real de los progresos y la integración con dispositivos de medición de actividad. Asimismo, se evaluó la usabilidad y la eficiencia de la aplicación mediante pruebas con usuarios y análisis de datos.

La mayoría de los objetivos específicos establecidos al inicio del proyecto han sido superados con éxito, lo que permite concluir que el objetivo principal del proyecto ha sido alcanzado.

Sin embargo, algunos de los objetivos específicos no se han cumplido. A continuación, se enumerarán estos objetivos y se explicarán los motivos de su incumplimiento.

1. Diseñar un algoritmo de planificación de entrenamientos que genere programas personalizados según los objetivos y el nivel de condición física del usuario, dentro de un plazo de 2 semanas desde el inicio del diseño y desarrollo del algoritmo.

2. Implementar un mecanismo para almacenar y analizar los datos de actividad del usuario, permitiendo un seguimiento detallado del progreso a lo largo del tiempo, dentro de un plazo de 6 semanas desde el inicio del diseño e implementación del mecanismo.

3. Desarrollar e implementar un sistema de notificaciones y recordatorios que mantenga a los usuarios informados sobre sus actividades programadas dentro de la aplicación, garantizando una entrega oportuna y confiable de las notificaciones en un plazo de 4 días desde el inicio del desarrollo.

4. Diseñar, desarrollar e implementar desafíos y sistemas de recompensas dentro de la aplicación que motiven a los usuarios a alcanzar sus objetivos de fitness, garantizando una variedad de desafíos y recompensas significativas en un plazo de 7 meses desde el inicio del diseño y desarrollo.

5. Integrar sistemas de puntuación y tablas de clasificación en la aplicación para fomentar una competencia amistosa entre los usuarios, asegurando una presentación clara y actualizada de las puntuaciones y clasificaciones en un plazo de 5 meses desde el inicio de la integración.

6. Desarrollar e implementar funciones de interacción social dentro de la aplicación que permitan a los usuarios compartir su progreso, logros y experiencias en plataformas de redes sociales, garantizando una integración fluida y segura en un plazo de 3 semanas desde el inicio del desarrollo.

7. Integrar API de redes sociales dentro de la aplicación para facilitar la conexión y la integración con plataformas populares como Facebook, Instagram y Twitter, asegurando una funcionalidad estable y segura antes de acabar la fase de desarrollo.

Los objetivos mencionados anteriormente no se han cumplido por dos razones principales. En algunos casos, la falta de tiempo derivada de una planificación insuficiente impidió la finalización de ciertas funcionalidades. En otros casos, la implementación de ciertas características requería una cuenta empresarial, lo cual no era viable dentro del alcance del proyecto.

A pesar de estos contratiempos, la valoración final del cliente ha sido positiva, indicando que el trabajo realizado ha sido exitoso. La aplicación ofrece al usuario una experiencia agradable y funcionalidades innovadoras.

Las tecnologías utilizadas, como *Node.js*, *PostgreSQL* y *websockets*, han resultado realmente eficaces, cumpliendo con las expectativas y proporcionando el rendimiento esperado.

Además, la arquitectura hexagonal y el diseño dirigido por el dominio (*DDD*) han superado las expectativas en términos de organización y escalabilidad del código. No obstante, la integración de estas arquitecturas en ciertas funcionalidades resultó costosa, lo que alargó el desarrollo de algunas características. Cabe mencionar que se podrían haber utilizado arquitecturas más sencillas que, potencialmente, habrían ofrecido resultados similares con un menor esfuerzo de implementación.

AGRADECIMIENTOS

Me gustaría agradecer, en primer lugar, a Ana Oropesa Física por su tutoría a lo largo del trabajo, por responder rápidamente a las dudas que surgieron y por estar pendiente del progreso en todo momento. También quiero expresar mi agradecimiento a mi compañero Jean, quien trabajó en la parte del front-end de la aplicación. Hemos tenido una comunicación excelente y hemos realizado un gran trabajo en equipo para completar la web.

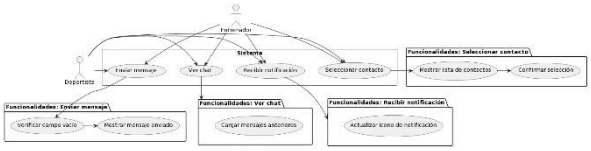
BIBLIOGRAFÍA

[1] Actividad física, la llave maestra para una salud integral [Online]. Disponible: https://www.forbes.com.mx/actividad-fisica-la-llave-maestrapara-una-salud-integral/
[2] Diagrama de Gantt: qué es y cómo crear uno con ejemplos [Online]. Disponible: https://asana.com/es/resources/gantt-chart-basics
[3] Qué es scrum y cómo empezar [Online]. Disponible: https://www.atlassian.com/es/agile/scrum
[4] ¿Qué es DevOps? [Online]. Disponible: https://www.netapp.com/es/devops-solutions/what-is-devops/
[5] ¿Qué es Docker y cómo funciona? [Online] . Disponible: https://www.redhat.com/es/topics/containers/what-is-docker
[6] ¿Qué es AWS? [Online]. Disponible: https://aws.amazon.com/whatis-aws/?nc2=h_ql_le_int

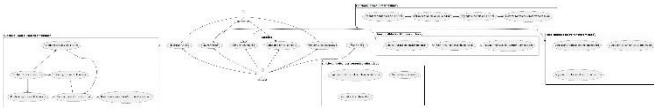
[7] Arquitectura hexagonal [Online]. Disponible: https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f
[8] Vertical Slice Architecture [Online]. Disponible: https://www.jimmybogard.com/vertical-slice-architecture/
[9] MVC [Online]. Disponible: https://developer.mozilla.org/es/docs/Glossary/MVC
[10] MVVM [Online]. Disponible: https://learn.microsoft.com/es-es/dotnet/architecture/maui/mvvm
[11] ¿Qué es Flux? Entendiendo su arquitectura [Online]. Disponible: https://carlosazaustre.es/como-funciona-flux
[12] Let's talk Domain Driven Design [Online]. Disponible: https://medium.com/@SkyscannerEng/lets-talk-domain-driven-design-d12e872c4611
[14] Como utilizar sockets en tu aplicación [Online]. Disponible: https://carlosazaustre.es/websockets-como-utilizar-socket-io-en-tu-aplicacion-web
[15] Librería de npm, socket.io [Online]. Disponible: https://www.npmjs.com/package/socket.io

APÉNDICE A

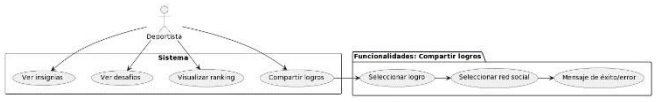
A1. CASO DE USO CHAT



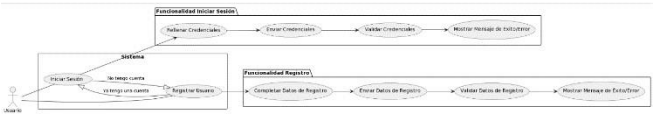
A2. CASO DE USO ENTRENAMIENTOS



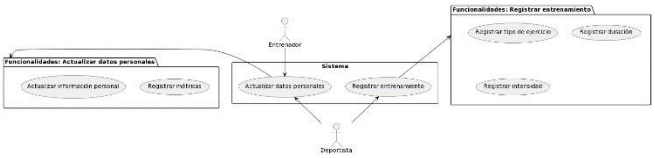
A3. CASO DE USO GAMIFICACIÓN



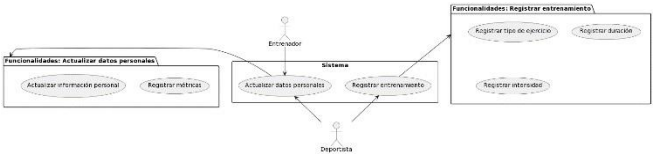
A4. CASO DE USO INICIÓ DE SESIÓN Y REGISTRO



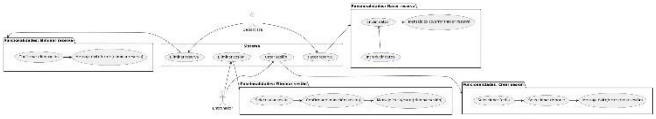
A5. CASO DE USO PERFILES DE USUARIO



A6. CASO DE USO PROGRESO



A7. CASO DE USO RESERVAS



APÉNDICE B

B1. GESTIÓN DE USUARIOS

B1.1. Encriptación de la contraseña

```
const encryptPassword = async (password: string): Promise<string> => {
    try {
        const hashedPassword = await bcrypt.hash(password, config.bcrypt.salts)
        return hashedPassword
    } catch (error: unknown) {
        throw new InternalServerException('Error encrypting password.')
    }
}
```

B1.2 Generación token de acceso seguro

```
const options: jwt.SignOptions = {
    algorithm: 'HS256',
    expiresIn: '1h',
}

const createJwt = (user: User): string => {
    try {
        return jwt.sign(
            { id: user.idValue, type: user.typeValue, email: user.emailValue },
            config.jwt.secret,
            options
        )
    } catch (error: unknown) {
        throw new InternalServerException('Error signing token.')
    }
}

export default createJwt
```

B1.3 Middleware de autentificación y autorización

```
const authMiddleware = (
    req: Request & { user: IUser },
    _: Response,
    next: NextFunction
) => {
    try {
        const header: string | null = req.headers['authorization']

        if (!header || !header.startsWith('Bearer')) {
            throw new UnauthorizedException('Unauthorized.')
        }
        const token: string = header.split(' ')[1]
        const decoded: UserDecoded = jwt.verify(token, config.jwt.secret, {
            complete: true,
        })
        if (decoded.payload && typeof decoded.payload !== 'string') {
            const user = {
                id: decoded.payload.id,
                type: decoded.payload.type,
                email: decoded.payload.email,
            }
            req.user = user
        }
        next()
    } catch (error: unknown) {
        next(error)
    }
}
```

B1.4 Middleware permisos de usuario

```
const typeMiddleware = (type: Type) => {
    return (
        req: Request & { user: IUser },
        _: Response,
        next: NextFunction
    ) => {
        try {
            if (req.user) {
                if (typeof type === 'string') {
                    if (req.user.type !== type) {
                        throw new UnauthorizedException('Type Unauthorized.')
                    }
                } else {
                    if (!type.includes(req.user.type)) {
                        throw new UnauthorizedException('Type Unauthorized.')
                    }
                }
            }
            next()
        } catch (error) {
            next(error)
        }
    }
}
```

B2. GESTIÓN DE SESIONES Y RESERVAS

B2.1 Consulta con mayor complejidad

```
SELECT
  sessions.id,
  sessions.name,
  sessions.description,
  sessions.duration,
  sessions.capacity,
  sessions.location,
  sessions.start_date,
  sessions.finish_date,
  json_build_object('id', MAX(users.id), 'name', MAX(users.name)) AS trainer,
  json_build_object('id', sessionstypes.id, 'name', sessionstypes.name, 'image', sessionstypes.image) AS type,
  COUNT(reservations.id) AS num_reservations
FROM
  sessions
LEFT JOIN
  reservations ON sessions.id = reservations.session_id AND reservations.status = $2
LEFT JOIN
  sessionstypes ON sessions.type_id = sessionstypes.id
LEFT JOIN
  users ON sessions.user_id = users.id
WHERE
  sessions.user_id = $1
GROUP BY
  sessions.id,
  sessions.name,
  sessions.description,
  sessions.duration,
  sessions.capacity,
  sessions.location,
  sessions.start_date,
  sessions.finish_date,
```

B3. GESTIÓN DE ENTRENAMIENTOS

B3.1 Asignación de un entrenamiento

```
class AddToFavoritesUseCase {
    private readonly workoutRepository: WorkoutRepository

    constructor(workoutRepository: WorkoutRepository) {
        this.workoutRepository = workoutRepository
    }

    public async execute(id: string, user_id: string): Promise<void> {
        try {
            const workoutId = new WorkoutId(id)
            const userId = new UserId(user_id)
            await this.workoutRepository.addToFavorites(workoutId, userId)
        } catch (error: unknown) {
            throw error
        }
    }
}
```

```
async addToFavorites(id: WorkoutId, userId: UserId): Promise<void> {
    try {
        const database: IDatabase = IOCService.get('database')
        const sql =
            'INSERT INTO users_workouts (user_id, workout_id) VALUES ($1, $2)'
        await database.query(sql, [userId.value, id.value])
    } catch (error: unknown) {
        throw error
    }
}
```

B3.2 Calificar un entrenamiento

```
class AddCalificationUseCase {
    private readonly workoutRepository: WorkoutRepository
    private readonly workoutDAO: WorkoutDAO

    constructor(workoutDAO: WorkoutDAO, workoutRepository: WorkoutRepository) {
        this.workoutDAO = workoutDAO
        this.workoutRepository = workoutRepository
    }

    public async execute(id: string, calification: number): Promise<void> {
        try {
            const workoutId = new WorkoutId(id)
            const workout = await this.workoutRepository.searchById(workoutId)
            const newRating =
                (workout.calificationValue * workout.numberOfRatingsValue +
                    calification) /
                (workout.numberOfRatingsValue + 1)
            workout.updateCalification(newRating)
            workout.updateNumberOfRatings(workout.numberOfRatingsValue + 1)
            await this.workoutDAO.update(workout)
        } catch (error: unknown) {
            throw error
        }
    }
}
```

```
async update(workout: Workout): Promise<void> {
  try {
    const database: IDatabase = IOCService.get('database')
    const data = workout.format()
    await database.update('workouts', data, 'id = $1')
    for (const updatedExercise of workout.exercisesValues) {
      await database.update(
        'workouts_exercises',
        {
          reps: updatedExercise.reps,
          series: updatedExercise.series,
        },
        'workout_id = $1 AND exercise_id = $2',
        [workout.idValue, updatedExercise.id]
      )
    }
  } catch (error: unknown) {
    throw error
  }
}
```

B4. SISTEMA DE MENSAJERÍA

B4.1 Interfaz y servicio para eventos (DDD)

```
export interface IEvent {
    type: string
    payload: any
}

export interface IEventBus {
    addEventListener(type: string, listener: (event: IEvent) => void): void
    emitEvent(event: IEvent): void
}
```

```
class EventBus implements IEventBus {
    private listeners: Map<string, ((event: IEvent) => void)[]> = new Map()

    public addEventListener(
        type: string,
        listener: (event: IEvent) => void
    ): void {
        if (!this.listeners.has(type)) {
            this.listeners.set(type, [])
        }
        this.listeners.get(type)?.push(listener)
    }

    public emitEvent(event: IEvent): void {
        const listeners = this.listeners.get(event.type)
        if (listeners) {
            listeners.forEach((listener: (event: IEvent) => void) =>
                listener(event)
            )
        }
    }
}
```

B4.2 Configuración de socket.io

```
const socketLoader = ({ server }: { server: Server }) => {
    const io = new SocketIOServer(server)
    const eventBus: IEventBus = IOCService.get('eventBus')

    io.on('connection', (socket: Socket) => {
        console.log('Client connected.')

        socket.on('messageCreated', (data: IMessageIncome) => {
            const event: IEvent = { type: 'messageCreated', payload: data }
            eventBus.emitEvent(event)
        })

        eventBus.addEventListener('messageSended', (event: IEvent) => {
            socket.emit('messageSended', event.payload as IMessageOutgoing)
        })
    })

    io.on('disconnect', (socket: Socket) => {
        socket.off('messageCreated', (data: IMessageIncome) => {
            const event: IEvent = { type: 'messageCreated', payload: data }
            eventBus.emitEvent(event)
        })
    })
}
```

B5. SISTEMA DE FILTRAJE

B5.1 Estructura patrón Criteria

```
export interface ICriteria {
    filters: IFilters
    offset: number
    limit: number
}
```

```
export default interface IFilters {
    calification?: IFilter
    type?: IFilter
    difficulty?: IFilter
}

interface IFilter {
    value: string
    operator?: OperatorValue<IOperators>
}
```

```
export enum IOperators {
    eq = '=', // Igual a
    gt = '>', // Mayor que
    gte = '>=', // Mayor o igual que
    lt = '<', // Menor que
    lte = '<=', // Menor o igual que
    ne = '<>', // Diferente de
    like = 'LIKE', // Similar a (usado para patrones)
    in = 'IN', // Dentro de un conjunto
    between = 'BETWEEN', // Entre dos valores
}

export type OperatorValue<T> = T[keyof T]
```

B5.2 Función que transforma una query en Criteria

```
create(query: any) {
    const { limit, offset, ...queryParams } = query
    const filters: IFilters = {}
    for (const param in queryParams) {
        if (Object.prototype.hasOwnProperty.call(queryParams, param)) {
            const data = queryParams[param]
            if (typeof data === 'object') {
                const validOperators = Object.keys(IOperators)
                const validOperator = Object.keys(data).find((operator) =>
                    validOperators.includes(operator)
                )
                if (validOperator) {
                    filters[param] = {
                        value: data[validOperator],
                        operator:
                            IOperators[
                                validOperator as keyof typeof IOperators
                            ],
                    }
                } else {
                    throw new InvalidArgumentException(
                        'Operator not valid.'
                    )
                }
            } else {
                filters[param] = { value: data, operator: '=' }
            }
        }
    }
    const criteria = {
        filters,
        limit: limit ? parseInt(limit) : 100,
        offset: offset ? parseInt(offset) : 0,
    }
    return criteria
}
```

B5.3 Función que transforma el Criteria en consulta SQL

```
buildQuery(queryBase: string, criteria: ICriteria): string {
    let newQuery = queryBase
    const { filters } = criteria
    const filterClauses: string[] = []
    for (const key in filters) {
        if (Object.prototype.hasOwnProperty.call(filters, key)) {
            const filter = filters[key]
            if (filter.operator) {
                filterClauses.push(
                    `${key} ${filter.operator} '${filter.value}'`
                )
            } else {
                filterClauses.push(`${key} = '${filter.value}'`)
            }
        }
    }
    if (filterClauses.length > 0) {
        const hasWhereClause = newQuery.toLowerCase().includes('where')
        const whereClause = hasWhereClause ? ' ' : ' WHERE '
        const groupByIndex = newQuery.toLowerCase().indexOf('group by')
        if (groupByIndex !== -1) {
            const partBeforeGroupBy = newQuery.substring(0, groupByIndex)
            const partAfterGroupBy = newQuery.substring(groupByIndex)
            newQuery = `${partBeforeGroupBy}\n${whereClause}${filterClauses.
            join(' AND ')}\n${partAfterGroupBy}`
        } else {
            newQuery += `${whereClause}${filterClauses.join(' AND ')}`
        }
    }
    newQuery += ` LIMIT ${criteria.limit} OFFSET ${criteria.offset}`
    return newQuery
}
```

APÉNDICE C

C.1 DISEÑO BASE DE DATOS