BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

Facultad de Ciencias de la Computación

Sistema web para trazabilidad personal de ejercicio físico

Tesis presentada para obtener el título de:
Ingeniero en Ciencias de la Computación

Presenta:
Alexander Hernández Popo

Directora de tesis:
Dra. María Teresa Torrijos Muñoz

Asesor de tesis
M.C. Carlos Armando Ríos Acevedo

19 de julio de 2025

DEDICATORIA

Agradezco a mis padres por inculcarme los valores pues sin ellos no pudiera llegar hasta este día, mi hermana y familiares por sus ánimos y a Tania Cansino por todo su apoyo emocional.

Además de a mi asesora pues sin su intervención y ayuda esta tesis no se hubiera llevado acabo

RESUMEN

La tesis se centra en el desarrollo de una aplicación web para la trazabilidad personal del ejercicio físico, con el objetivo de promover un estilo de vida saludable y activo; esto mediante llamadas de API´s a servicios de terceros para una mejor integración.

Desde un punto de vista de la salud, la aplicación busca facilitar a los usuarios el seguimiento y monitoreo de su actividad física en diversas modalidades, como ejercicio en el gimnasio, natación, usar la bicicleta, correr, caminar, entre otras.

Se pretende concienciar sobre la importancia del ejercicio para la prevención de enfermedades crónicas y el mantenimiento de un buen estado de salud y bienestar general.

En el ámbito de las ciencias computacionales, el proyecto implica el desarrollo de algoritmos y herramientas tecnológicas para trazabilidad, análisis y visualización de datos relacionados con la actividad física. Se utilizarán tecnologías web modernas, como Angular, Node.js, Mongo DB y Express.js, para construir una aplicación eficiente y escalable.

El impacto socioeconómico del proyecto es significativo. Por un lado, contribuye a la mejora de la salud y calidad de vida de las personas al fomentar la actividad física y promover un estilo de vida activo. Por otro lado, impulsa el avance en las ciencias computacionales al aplicar algoritmos y tecnologías web en un contexto práctico y relevante para la sociedad.

Índice

Tabla de contenido

Introducción

1.1. Antecedentes

Se ha observado un creciente interés en el desarrollo de aplicaciones móviles y plataformas web dedicadas al seguimiento del ejercicio físico y la promoción de un estilo de vida saludable. Las aplicaciones existentes en este ámbito se utilizan principalmente para registrar actividades físicas, establecer metas de ejercicio y proporcionar retroalimentación sobre el progreso del usuario.

Se encontró que las aplicaciones analizadas en general se usan para rastrear una variedad de actividades deportivas, como correr, caminar, andar en bicicleta y entrenamientos de fuerza. Además, estas aplicaciones suelen ofrecer funciones adicionales como la planificación de entrenamientos, el seguimiento del progreso a lo largo del tiempo, la integración con dispositivos de seguimiento de actividad y la participación en desafíos comunitarios.

En la Tabla 1 se presenta el análisis comparativo de las características de algunas aplicaciones para el seguimiento de ejercicio físico.

Característica	Strava	TrainingPeaks	Strong	Runkeeper	Endomondo	Google Fit	Sistema Web Propuesto
Enfoque Principal	Rastreo de actividades, rendimiento, social	Planificación y seguimiento de entrenamientos	Levantamiento de pesas, entrenamiento de fuerza	Rastreo de actividades, motivación	Rastreo de actividades, planes de entrenamiento, desafíos comunitarios	Rastreo de actividad física, objetivos, integración con dispositivos	Trazabilidad personal de ejercicio físico
Diferenciador Clave	Transcavidad del rendimiento físico, funciones sociales	Planes de entrenamiento personalizados, análisis de datos	Enfoque específico en levantamiento de pesas	Interfaz intuitiva, seguimiento de actividades al aire libre	Desafíos sociales, enfoque a la comunidad	Integración con el ecosistema de Google	Integración con otros dispositivos y servicios, personalización
Seguimiento del Rendimiento	Sí	Sí	Sí	Sí	No	No	Sí
Planificación de Entrenamientos	No	Sí	No	No	Sí	No	Sí
Desafíos Comunitarios	Sí	Sí	No	No	Sí	No	No
Integración con Dispositivos	Sí	No	No	No	No	Sí	Sí

Tabla 1. Análisis comparativo de aplicaciones para el seguimiento de ejercicio físico

Las aplicaciones analizadas son funcionales y tienen interfaces gráficas amigables, son intuitivas y de fácil manejo sin embargo no consideran la trazabilidad personal.

La aplicación web que se pretende desarrollar, se diferencia al considerar una serie de características clave que la destacan. En primer lugar, la aplicación web se centra en la trazabilidad personal del ejercicio físico, proporcionando a los usuarios una plataforma integral para registrar, analizar y mejorar su actividad física de manera personalizada. Además, la aplicación se distingue por su enfoque en la facilidad de integración con una variedad de dispositivos y servicios externos, lo que permite una experiencia de usuario más completa y adaptada a las necesidades individuales.

1.2. Planteamiento del problema

Situación Actual:

En México, la falta de importancia atribuida al ejercicio físico y a una actividad física regular es una preocupación creciente. Muchas personas, influenciadas por limitaciones de tiempo y recursos económicos, no logran incorporar el ejercicio como parte de su rutina diaria. Esta falta de actividad física contribuye a una serie de problemas de salud tanto físicos como mentales, que van desde la obesidad hasta problemas cardiovasculares, afectando negativamente el bienestar general de la población.

Problemas Identificados:

La falta de ejercicio contribuye a problemas de salud física y mental en la sociedad mexicana.

La percepción de que el ejercicio es inaccesible debido a limitaciones de tiempo y recursos económicos es un obstáculo significativo para muchas personas.

La falta de conciencia sobre los beneficios del ejercicio y la importancia de una actividad física regular contribuye a la prevalencia de problemas de salud relacionados con el sedentarismo.

Relevancia e Impacto:

Estos problemas tienen un impacto significativo en todos los sectores de la sociedad mexicana, independientemente del estatus económico. La obesidad y los problemas de salud relacionados con la falta de ejercicio no solo afectan la salud física de las personas, sino que también tienen un impacto en su bienestar emocional y mental. Abordar estos problemas es fundamental para mejorar la calidad de vida y el bienestar general de la población.

Problema Técnico en Otras Aplicaciones:

El análisis de otras aplicaciones similares revela una serie de desafíos técnicos desde la perspectiva de las ciencias computacionales. Estos desafíos incluyen:

Infraestructura Subóptima:

La infraestructura subóptima puede manifestarse de varias formas, desde la falta de configuración adecuada de servidores hasta la ausencia de estrategias de escalabilidad efectivas. Además de los aspectos mencionados, otros problemas comunes pueden incluir:

- **Load Balancer Inadecuado:** Un load balancer mal configurado o insuficiente puede no distribuir adecuadamente la carga entre los servidores, lo que resulta en una distribución desigual del tráfico y la sobrecarga de algunos servidores mientras otros permanecen subutilizados.
- **Autenticación Insegura:** La falta de medidas de autenticación sólidas, como OAuth, Okta o Microsoft Authenticator, puede comprometer la seguridad de la

aplicación, dejándola vulnerable a ataques de suplantación de identidad o acceso no autorizado.

- **Peticiones Asincrónicas no Manejadas:** Las peticiones asincrónicas de los usuarios pueden generar problemas si no se gestionan adecuadamente, como la falta de control sobre el estado de las solicitudes o la pérdida de datos debido a un manejo inadecuado de la concurrencia.

Rendimiento y Escalabilidad:

El rendimiento y la escalabilidad son fundamentales para garantizar que la aplicación pueda manejar eficazmente un aumento en la carga de trabajo y seguir siendo receptiva para los usuarios. Algunos desafíos específicos que pueden surgir incluyen:

- **Uso de Tecnologías No Escalables:** La elección de tecnologías no escalables o monolíticas puede limitar la capacidad de la aplicación para escalar horizontalmente y distribuir la carga de manera eficiente entre múltiples instancias.
- **Complejidad de la Arquitectura Monolítica:** Las arquitecturas monolíticas pueden volverse complejas y difíciles de mantener a medida que la aplicación crece, lo que puede afectar negativamente la escalabilidad y el rendimiento general.
- **Falta de Microservicios:** La falta de una arquitectura basada en microservicios puede dificultar la escala de componentes individuales de la aplicación de manera independiente, lo que resulta en cuellos de botella y puntos únicos de falla en el sistema.
- **Ausencia de Estrategias de Cacheado:** La falta de estrategias de cacheado adecuadas puede provocar un rendimiento deficiente al requerir consultas

repetidas a la base de datos u operaciones costosas de procesamiento de datos en cada solicitud.

- **Problemas de Latencia:** La latencia excesiva puede ser un problema si los servidores no están distribuidos geográficamente de manera adecuada o si no se implementan técnicas de optimización para reducir el tiempo de respuesta de la aplicación.

Complejidad de la Programación:

La complejidad en la programación puede resultar en varios desafíos técnicos que dificultan la gestión y el mantenimiento de la aplicación. Algunos aspectos adicionales a considerar son:

- **Falta de Modularidad:** Una arquitectura de código monolítica o poco modularizada puede aumentar la complejidad y la dificultad para agregar nuevas funcionalidades o realizar cambios sin afectar otras partes del sistema.
- **Dependencias Excesivas:** El uso excesivo de dependencias puede hacer que el sistema sea frágil y propenso a fallos, especialmente si las dependencias no se actualizan regularmente o si tienen conflictos entre sí.
- **Falta de Documentación:** La falta de documentación clara y concisa puede dificultar la comprensión del código y la colaboración entre desarrolladores, lo que aumenta el tiempo y los esfuerzos requeridos para realizar cambios o correcciones.
- **Escasa Cobertura de Pruebas:** Una cobertura de pruebas insuficiente puede dejar áreas críticas del código sin validar, lo que aumenta el riesgo de introducir errores y reduce la confianza en la estabilidad y fiabilidad de la aplicación.

Integración con Otros Servicios y Dispositivos:

La integración con otros servicios y dispositivos externos puede presentar desafíos únicos que afectan la interoperabilidad y la funcionalidad general de la aplicación. Algunos aspectos adicionales a considerar son:

- **Compatibilidad de Protocolos:** Las diferencias en los protocolos de comunicación entre la aplicación y los servicios externos pueden dificultar la integración y generar errores de comunicación o pérdida de datos.
- **Gestión de Tokens de Acceso:** La gestión inadecuada de tokens de acceso y credenciales de autenticación puede comprometer la seguridad y exponer la aplicación a riesgos de seguridad, como el acceso no autorizado o la suplantación de identidad.
- **Manejo de Sincronización:** La sincronización de datos entre la aplicación y los servicios externos puede ser compleja y propensa a errores si no se implementan adecuadamente mecanismos de control de concurrencia y gestión de transacciones.
- **Actualizaciones y Mantenimiento:** Las actualizaciones o cambios en los servicios externos pueden requerir modificaciones en la aplicación para mantener la compatibilidad y la funcionalidad, lo que aumenta la complejidad y el riesgo de errores.

Seguridad e Integridad de Datos:

La seguridad y la integridad de los datos son fundamentales para proteger la información del usuario y mantener la confianza en la aplicación. Algunos aspectos adicionales a considerar son:

- **Gestión de Vulnerabilidades:** La falta de gestión proactiva de vulnerabilidades en la aplicación puede dejarla expuesta a ataques conocidos o explotaciones

de seguridad, comprometiendo la confidencialidad, integridad y disponibilidad de los datos.

- **Auditoría de Acceso:** La falta de mecanismos de auditoría de acceso puede dificultar la identificación de actividades maliciosas o sospechosas, lo que limita la capacidad de detectar y responder a incidentes de seguridad de manera oportuna.
- **Cifrado de Datos Sensibles:** El cifrado inadecuado de datos sensibles puede dejar la información del usuario vulnerable a accesos no autorizados o filtraciones de datos, especialmente en caso de brechas de seguridad o robos de información.
- **Cumplimiento Normativo:** El incumplimiento de regulaciones y estándares de seguridad, como GDPR o HIPAA, puede exponer a la aplicación a sanciones legales y dañar la reputación de la empresa, especialmente en sectores sensibles como la salud o la privacidad de datos.

1.3. Objetivos

En esta sección se presentan los objetivos propuestos para esta tesis.

1.3.1. Objetivos General

Desarrollar una aplicación web que permita llevar la trazabilidad de ejercicio físico como medida de superación personal; esto mediante el uso de frame work angular con backend, node.js y express; usando la base de datos desplegada en la nube Mongo DB.

1.3.2. Objetivos específicos

Investigar y analizar las necesidades y preferencias de los usuarios en cuanto a la trazabilidad del ejercicio físico, mediante encuestas y estudios de mercado, para entender mejor los requisitos de la aplicación.

1. Diseñar la arquitectura de la aplicación web, definiendo los componentes front-end y back-end necesarios, así como la interacción entre ellos, utilizando el framework Angular para el desarrollo del cliente y Node.js con Express para el servidor.
2. Desarrollar la funcionalidad principal de la aplicación web, que incluye la capacidad de registrar y visualizar actividades físicas, establecer metas de ejercicio, y proporcionar retroalimentación y análisis de rendimiento.
3. Implementar la integración con servicios externos y dispositivos, como relojes inteligentes y aplicaciones de fitness, para permitir una recopilación automática de datos de actividad física y una experiencia más completa para el usuario.
4. Configurar y desplegar la base de datos MongoDB en la nube, garantizando la escalabilidad, disponibilidad y seguridad de los datos almacenados.
5. Realizar pruebas exhaustivas de la aplicación para garantizar su funcionamiento correcto y su compatibilidad con una variedad de dispositivos y navegadores web.
6. Documentar el proceso de desarrollo, incluyendo la arquitectura de la aplicación, los requisitos del sistema, los diagramas de flujo y la guía de usuario, para facilitar su comprensión y mantenimiento.
7. Evaluar la aplicación mediante pruebas piloto con usuarios reales, recopilando retroalimentación y realizando ajustes según sea necesario para mejorar la experiencia del usuario y la efectividad de la aplicación.

1.4. Estado del arte

Esta sección se encuentra enfocada a revisar y analizar el uso de procesos, infraestructura, algoritmos y técnicas computacionales que han sido empleados en el desarrollo de aplicaciones relacionadas con la actividad física y el seguimiento del ejercicio.

Este análisis permite mostrar el estado actual del campo tecnológico en este ámbito y establecer las bases para la implementación de la aplicación.

Uso de Procesos, Infraestructura, Algoritmos y Técnicas Computacionales

Se examinaron diversos aspectos relacionados con el desarrollo de aplicaciones, incluyendo:

- Procesos de desarrollo de software, como metodologías ágiles (por ejemplo, SCRUM) y buenas prácticas de ingeniería de software.
- Infraestructura tecnológica utilizada para alojar y gestionar aplicaciones web, incluyendo servidores, servicios en la nube y plataformas de desarrollo.
- Algoritmos y técnicas computacionales empleados para el procesamiento y análisis de datos relacionados con la actividad física, como algoritmos de seguimiento de movimiento, análisis de datos biométricos y técnicas de visualización de datos.

Se revisaron las aplicaciones existentes y los antecedentes relevantes en el campo de la actividad física y el seguimiento del ejercicio, incluyendo:

- Aplicaciones móviles y web diseñadas para registrar y monitorizar actividades físicas, ofreciendo funcionalidades como seguimiento de entrenamientos, análisis de rendimiento y motivación personal.
- Investigaciones y estudios previos que han explorado el impacto de la tecnología en la promoción de un estilo de vida activo y saludable, así como en la mejora del rendimiento deportivo.

Ventajas del Framework Utilizado

Para el desarrollo de la aplicación, se utilizan tecnologías como Angular, Node.js y Express.js. Estas tecnologías ofrecen varias ventajas, incluyendo:

- Facilidad de desarrollo y mantenimiento gracias a la estructura modular y la amplia comunidad de desarrolladores.
- Alta velocidad de desarrollo y rendimiento, lo que permite crear aplicaciones web rápidas y eficientes.
- Escalabilidad para manejar grandes volúmenes de datos y usuarios concurrentes.
- Integración sencilla con otras herramientas y servicios gracias a su arquitectura basada en componentes.

1.5. Metodología

De acuerdo con Molina, J. R., Honores, J. A., Pedreira-Souto, N., y Pardo, H. P. (2021), el desarrollo se soporta en el uso de una metodología formal de desarrollo de software con enfoque ágil en el marco de Scrum; lo anterior basado también en el conocimiento del área de la aplicación y que se cuenta con el apoyo de los stake holders para el desarrollo de esta.

De acuerdo con lo consultado en los sitios web Amazon Web Services , Atlassian y Red Hat, la metodología Scrum es esencial para el desarrollo de software y otros proyectos, ya que permite a los equipos adaptarse rápidamente a los cambios en los requisitos sin desviarse del presupuesto o del cronograma.

Esto se logra mediante la organización del trabajo en iteraciones cortas llamadas "Sprints", durante las cuales se desarrolla, prueba y entrega un incremento del producto. Scrum se centra en mantener la calidad al definir claramente los requisitos al inicio de cada Sprint y al obtener retroalimentación continua del cliente. Esto garantiza que los requisitos permanezcan relevantes y viables durante todo el proyecto.

Además, Scrum ayuda a maximizar el retorno de la inversión al priorizar los requisitos según su valor para el cliente y su riesgo. Se enfoca en desarrollar un producto funcional inicial que pueda lanzarse al mercado temprano para obtener retroalimentación rápida de los clientes. Esto reduce los defectos costosos y promueve la eficiencia del equipo, lo que ahorra dinero a largo plazo.

Esta metodología también nos permite tener una organización y administración más personalizada y analítica pues las métricas como el tiempo, presupuesto y calidad del proyecto pueden ser medidas, permitiéndonos adaptarnos a medida que se desarrolla el proyecto.

Capítulo 1

Requerimientos

En este capítulo se detallan los requerimientos necesarios para el diseño y desarrollo de la aplicación web de trazabilidad de ejercicio físico.

1.1. Funcionalidades Principales de la Aplicación

La aplicación desarrollada cuenta con un conjunto de funcionalidades clave diseñadas para mejorar la experiencia del usuario y facilitar el seguimiento de su actividad física. Estas funcionalidades permiten registrar, gestionar y analizar datos relevantes, proporcionando herramientas que fomentan la motivación y el cumplimiento de objetivos personales de entrenamiento. A continuación, se describen las principales características:

- **Registro de Actividades Físicas**: Los usuarios podrán trivializar diversas actividades físicas, como correr, caminar, nadar, entre otras.
- **Establecimiento de Metas de Ejercicio:** La aplicación permitirá a los usuarios convenir metas personalizadas de ejercicio, incluyendo distancia, tiempo, plazo, calorías quemadas, entre otros parámetros**.**
- **Visualización de Progreso y Estadísticas:** Se ofrecerá a los usuarios la solución de percibir su progreso y estadísticas de actividades físicas a lo largo del tiempo, mediante gráficos y tablas informativas.
- **Integración con Dispositivos Externos:** Se integrarán dispositivos externos, como relojes inteligentes y aplicaciones de fitness, para clasificar

automáticamente datos de actividad física y proporcionar una experiencia más completa al usuario.

- **Notificaciones y Recordatorios:** La aplicación enviará notificaciones y recordatorios para motivar al usuario a cumplir con sus metas de ejercicio y mantener una rutina activa.
- **Generación de Informes y Análisis de Rendimiento:** Se generarán informes y análisis detallados del rendimiento del usuario, ofreciendo insights y recomendaciones para mejorar su actividad física.

1.2. Requisitos de Usuario

Los requisitos de usuario establecen las características y condiciones que la aplicación debe cumplir para ofrecer una experiencia satisfactoria, segura y accesible a sus usuarios. Estos requisitos se enfocan en aspectos de usabilidad, personalización, disponibilidad y protección de datos, garantizando que la solución desarrollada responda de forma efectiva a las necesidades y expectativas del público objetivo. A continuación, se detallan los principales requisitos:

- **Interfaz de Usuario Intuitiva:** La aplicación debe contar con una interfaz de usuario intuitiva y fácil de usar, que permita a los usuarios navegar y utilizar todas las funcionalidades de manera sencilla.
- **Personalización de Metas y Preferencias:** Los usuarios podrán personalizar sus metas y preferencias de ejercicio de acuerdo con sus necesidades y objetivos individuales.
- **Acceso Multiplataforma:** La aplicación estará disponible para su acceso desde múltiples dispositivos, incluyendo computadoras, smartphones y tablets.

- **Seguridad y Privacidad de los Datos:** Se garantizará la seguridad y privacidad de los datos del usuario, implementando medidas de protección y cumpliendo con regulaciones de privacidad.
- **Disponibilidad sin Conexión:** Se proporcionará la opción de acceso y registro de actividades sin conexión a internet, para permitir a los usuarios registrar actividades en cualquier momento y lugar.

1.3. Información a Recopilar y Almacenar

La aplicación requiere recopilar y almacenar diversos tipos de información con el fin de ofrecer un seguimiento preciso de la actividad física del usuario, personalizar la experiencia y facilitar el análisis de su progreso. Estos datos se gestionarán de forma segura, respetando la privacidad y confidencialidad de cada usuario, y se emplearán exclusivamente para optimizar el uso de la aplicación y mejorar sus funcionalidades. A continuación, se detallan los principales tipos de información que serán registrados:

- **Datos de Actividad Física**: Distancia, duración, ritmo, entre otros parámetros de actividades físicas realizadas por el usuario.
- **Metas de Ejercicio:** Metas establecidas por el usuario, como distancia a recorrer, tiempo de ejercicio, calorías a quemar, entre otras**.**
- **Preferencias y Configuraciones:** Preferencias de usuario, configuraciones de la aplicación y ajustes personalizados.
- **Datos de Salud y Bienestar:** Información relevante sobre salud y bienestar, como peso, altura, frecuencia cardíaca, entre otros datos.

1.4. Integración con Dispositivos Externos y Servicios

La aplicación está diseñada para interactuar de forma eficiente con dispositivos y servicios externos, permitiendo ampliar sus capacidades y mejorar la experiencia del

usuario. Esta integración facilita la recolección de datos en tiempo real, la automatización de procesos y el acceso a funcionalidades adicionales mediante tecnologías de comunicación y autenticación seguras. A continuación, se describen los principales aspectos considerados para lograr esta integración:

- **Protocolos de Comunicación Compatibles:** Utilización de protocolos como Bluetooth y APIs web para la comunicación con dispositivos externos.
- **Autorización y Autenticación:** Implementación de mecanismos de autorización y autenticación de usuarios para dispositivos externos**.**
- **Sincronización Automática:** Sincronización automática de datos entre la aplicación y los dispositivos externos para garantizar la actualización de información en tiempo real.

1.5. Aspectos de Seguridad

La seguridad constituye un pilar fundamental en el diseño y desarrollo de la aplicación, garantizando la protección de la información del usuario y la integridad del sistema. Para ello, se contemplan medidas que aseguran la confidencialidad, autenticidad y disponibilidad de los datos, así como el cumplimiento de la normativa vigente en materia de privacidad. A continuación, se presentan los principales aspectos de seguridad implementados:

- **Protección de Datos Personales:** Implementación de medidas de protección de datos personales y confidenciales del usuario.
- **Encriptación de Comunicaciones y Almacenamiento:** Utilización de técnicas de encriptación para asegurar la confidencialidad de las comunicaciones y datos almacenados.
- **Autenticación de Usuarios**: Implementación de mecanismos de autenticación de usuarios y control de acceso.

- **Cumplimiento de Regulaciones:** Cumplimiento de regulaciones de privacidad y protección de datos vigentes en la jurisdicción correspondiente.

1.6. Requisitos Técnicos

Los requisitos técnicos definen las condiciones y especificaciones necesarias para garantizar el correcto desarrollo, funcionamiento y despliegue de la aplicación. Estos aspectos contemplan la selección de tecnologías, la optimización del rendimiento, la escalabilidad del sistema y la compatibilidad con diferentes entornos, asegurando que la solución final cumpla con los estándares de calidad y disponibilidad esperados. A continuación, se describen los principales requisitos técnicos considerados:

- **Uso de Tecnologías Específicas:** Utilización de tecnologías como Angular, Node.js y MongoDB para el desarrollo del sistema.
- **Escalabilidad y Disponibilidad**: Garantizar la escalabilidad y disponibilidad de la infraestructura de servidores para soportar la carga de usuarios y mantener la disponibilidad del servicio.
- **Rendimiento de la Aplicación:** Optimización del tiempo de respuesta y rendimiento de la aplicación para una experiencia fluida del usuario.
- **Compatibilidad con Navegadores y Dispositivos:** Asegurar la compatibilidad de la aplicación con una variedad de navegadores web y dispositivos móviles.

1.6.1. JavaScript y el Ecosistema de Node.js

JavaScript fue desarrollado en 1995 por Brendan Eich, mientras trabajaba en Netscape Communications, con el objetivo de dotar a las páginas web de interactividad directamente en el navegador. De manera sorprendente, la primera versión funcional del lenguaje se creó en tan solo diez días, marcando el inicio de una

evolución constante. Lo que comenzó como un lenguaje de scripting limitado para el lado del cliente se ha transformado, con el paso del tiempo, en una herramienta versátil capaz de ejecutarse tanto en el cliente como en el servidor.

Hoy en día, JavaScript es uno de los pilares del desarrollo web moderno. Su versatilidad y amplio ecosistema le permiten impulsar aplicaciones completas, abarcando desde la interfaz de usuario en el *front-end* hasta la lógica de negocio en el *back-end*, especialmente gracias a entornos como Node.js que han expandido sus posibilidades más allá del navegador.

Node.js: Plataforma de Desarrollo Back-end

Características de Node.js:

Ejecución de JavaScript en el servidor: Node.js permite que el código JavaScript se ejecute en el lado del servidor.

Modelo de operaciones sin bloqueo (Non-blocking I/O): Node.js utiliza un modelo de entrada/salida no bloqueante y orientado a eventos, lo que lo hace eficiente para aplicaciones que manejan muchas operaciones de I/O simultáneas.

Node Package Manager (NPM) es el gestor de paquetes por defecto para Node.js y juega un papel fundamental en el ecosistema de desarrollo con JavaScript. Ofrece un repositorio centralizado donde los desarrolladores pueden publicar y compartir módulos de Node.js, permitiendo así a otros integrar estas funcionalidades en sus propios proyectos de forma sencilla.

Gestión de Dependencias con NPM

La gestión de dependencias es un componente esencial en el desarrollo de aplicaciones modernas, ya que permite integrar bibliotecas, módulos y herramientas de terceros para ampliar las funcionalidades de un proyecto. En el ecosistema de JavaScript y Node.js, esta tarea se realiza principalmente mediante Node Package Manager (NPM), el gestor de paquetes por defecto para Node.js.

Funcionalidades y Beneficios de NPM:

- **Acceso a un vasto ecosistema**: NPM proporciona acceso a un amplio repositorio de paquetes, que al momento cuenta con más de un millón de paquetes disponibles. Esto permite a los desarrolladores encontrar casi cualquier biblioteca o herramienta que necesiten para sus proyectos.
- **Facilidad de uso**: Con simples comandos como **npm install <nombre_del_paquete>**, los desarrolladores pueden agregar rápidamente nuevas dependencias a sus proyectos.
- **Gestión eficiente de dependencias**: NPM facilita la gestión de versiones y dependencias, asegurando que los proyectos permanezcan actualizados y compatibles con otras bibliotecas. Además, el archivo **package.json** permite configurar y documentar todas las dependencias necesarias para el proyecto, simplificando la colaboración y el despliegue.

- **Ejemplos de Paquetes Comunes en NPM**:
  - **Express**: Un framework minimalista y flexible para aplicaciones web y APIs. Es ampliamente utilizado para simplificar la creación de servidores y manejo de rutas en aplicaciones Node.js.
  - **Mongoose**: Una biblioteca de modelado de objetos para MongoDB y Node.js. Facilita la construcción de esquemas para modelar los datos de la aplicación, proporciona métodos para consultar la base de datos de forma más sencilla y valida los datos antes de guardarlos en MongoDB.

1.6.2. Angular como Framework para Frontend con ts (Typescript)

Introducción a Angular

En el ámbito del desarrollo de aplicaciones web modernas, seleccionar un framework adecuado para el frontend es crucial debido a su impacto directo en la interactividad, la experiencia del usuario y la eficiencia del desarrollo. Angular, desarrollado y mantenido por Google, se presenta como una solución robusta y ampliamente adoptada en la industria para enfrentar estos desafíos. Este framework de código abierto es especialmente valorado por su capacidad para construir aplicaciones de una sola página (Single Page Applications, SPAs) que ofrecen experiencias fluidas y dinámicas a los usuarios, similares a las aplicaciones de escritorio.

Arquitectura de Angular

La arquitectura de Angular está basada en una jerarquía de componentes, lo que es ideal para la reutilización de código y la división modular del trabajo en equipo. Cada componente en Angular es esencialmente una clase de TypeScript que interactúa con su HTML y CSS asociados, facilitando así el desarrollo separado del comportamiento y la presentación de la aplicación. Angular también incluye servicios, que son singleton que se pueden inyectar en los componentes, permitiendo una forma eficiente de compartir código y funcionalidades.

Ventajas Competitivas

Optar por Angular para el desarrollo frontend ofrece varias ventajas competitivas. Su vinculación bidireccional de datos reduce el trabajo necesario para sincronizar el modelo y la vista, y su extenso ecosistema, incluyendo herramientas avanzadas como Angular CLI y Angular Material, acelera el proceso de desarrollo. Además, Angular garantiza aplicaciones más seguras de forma predeterminada, ofreciendo protecciones contra vulnerabilidades comunes como XSS (Cross-Site Scripting).

Elección de Angular

La elección de Angular como framework para el desarrollo del frontend de nuestra aplicación web se justifica no solo por sus capacidades técnicas y su robustez sino también por la experiencia previa que nuestro equipo tiene trabajando con este framework. Esta familiaridad con Angular reduce la curva de aprendizaje y aumenta la eficiencia del desarrollo, permitiéndonos aprovechar al máximo las características avanzadas del framework para cumplir los objetivos de nuestra tesis de manera efectiva y eficiente.

1.6.3. Typescript

TypeScript es un lenguaje de programación desarrollado y mantenido por Microsoft. Es un superset de JavaScript, lo que significa que cualquier código JavaScript válido es también código TypeScript válido. Este lenguaje fue diseñado para agregar características adicionales a JavaScript, especialmente el tipado estático y la orientación a objetos basada en clases, lo que facilita el desarrollo de aplicaciones complejas y a gran escala.

Características principales de Typescript

- **Tipado Estático**: introduce tipos estáticos en JavaScript. Esto permite a los desarrolladores definir tipos de variables, parámetros y objetos. El tipado estático ayuda a detectar errores en tiempo de compilación, mucho antes de que el código se ejecute, lo que puede reducir significativamente los errores en tiempo de ejecución y mejorar la mantenibilidad del código.
- **Clases, Interfaces y Herencia**: soporta características modernas de programación orientada a objetos como clases, interfaces y herencia. Esto hace que TypeScript sea una excelente opción para proyectos grandes y complejos donde tales estructuras ayudan a organizar y modularizar el código de manera más eficiente.

1.6.4 Mongo DB como Sistema de Gestión de Base de Datos

MongoDB es un sistema de gestión de bases de datos NoSQL que utiliza un formato de almacenamiento basado en documentos. Esta característica lo hace ideal para proyectos que requieren alta flexibilidad y escalabilidad. Elegimos MongoDB por varias razones clave que se alinean con las necesidades y condiciones específicas de nuestro proyecto.

Características principales de Mongo DB

- **Basado en Documentos**: Se almacenan los datos en documentos BSON (Variante binaria de JSON), esta permite una gran flexibilidad en la estructura de datos. Pueden contener campos variados y no necesitan seguir un esquema fijo, facilitando la evolución de la BD conforme los ajustes requeridos de la aplicación
- **Escalabilidad**: Ofrece alta escalabilidad en ambos sentidos como horizontal y vertical, manejando eficientemente grandes cantidades de datos y tráfico de usuarios

Ventajas de Usar MongoDB

- **Despliegue en la Nube**: MongoDB está ya desplegada en la nube, lo que proporciona varias ventajas como la reducción en la gestión de infraestructura, la mejora en la disponibilidad y la capacidad de escalar rápidamente según sea necesario.
- **Modelo de Pago por Uso**: ofrece opciones de pruebas gratuitas y modelos de pago por uso, lo que significa que podemos empezar sin costes iniciales y solo pagaremos más a medida que nuestras necesidades de almacenamiento y procesamiento crezcan.

- **Facilidad de Uso**: La experiencia previa del equipo con MongoDB nos permite aprovechar nuestro conocimiento existente para optimizar el desarrollo y mantenimiento de la base de datos

Capítulo 2: Análisis y Diseño

En este capítulo se presenta el análisis y diseño detallado de la aplicación web para la trazabilidad personal de ejercicio físico.

2.1. Revisión de Requerimientos:

Se realiza una revisión detallada de los requerimientos definidos en el Capítulo 2, para comprender completamente las funcionalidades y características que debe tener la aplicación.

2.2. Definición de la Arquitectura:

La arquitectura de una aplicación define cómo se estructuran e interactúan sus componentes. Para nuestro proyecto, hemos seleccionado una arquitectura que combina el modelo cliente-servidor con un enfoque monolítico, utilizando tecnologías específicas que incluyen Angular para el frontend, Node.js con Express para el backend, y MongoDB para la gestión de datos.

Modelo Cliente-Servidor

El modelo cliente-servidor es una estructura de aplicación distribuida que segrega las funcionalidades en dos partes principales: el cliente y el servidor. El cliente opera en el navegador del usuario y es responsable de presentar la interfaz de usuario y gestionar las interacciones del usuario. El servidor, implementado usando Node.js y el framework Express, maneja la lógica de negocio, el procesamiento de datos y las interacciones con la base de datos MongoDB.

- **Ventajas:**
    - **Separación de preocupaciones:** Facilita el desarrollo y mantenimiento al separar la presentación de la lógica de negocio.
    - **Escalabilidad:** Permite una expansión más sencilla si aumentan las demandas de la aplicación7

Arquitectura monolítica

Optamos por una arquitectura monolítica, en la cual el cliente y el servidor forman una única unidad cohesiva durante el despliegue. Esta estructura es ideal para aplicaciones con requisitos moderados de transacciones y simplifica tanto el desarrollo como la administración.

Tecnologías Especificas

Se utilizaron las siguientes tecnologías en el desarrollo del proyecto:

- Angular: Utilizado para el desarrollo del frontend, aprovecha TypeScript para mejorar la calidad del código y la productividad del desarrollo.
- Node.js con Express: Node.js sirve como el entorno de ejecución para el backend, mientras que Express proporciona un framework robusto para crear las APIs de la aplicación.
- Como sistema de gestión de base de datos, MongoDB ofrece una solución eficiente y escalable para manejar las operaciones de datos en formato de documentos, lo que se alinea bien con la naturaleza dinámica y flexible de JavaScript y Node.js.

Justificación de la Elección

La decisión de usar Angular, Node.js con Express y MongoDB se basa en la experiencia previa del equipo, así como en las características de rendimiento y escalabilidad de estas tecnologías. Además, la arquitectura monolítica seleccionada se justifica por la simplicidad operativa y la adecuación a los requisitos moderados de transacciones de nuestra aplicación.

Conclusión

La arquitectura cliente-servidor monolítica utilizando Angular, Node.js con Express y MongoDB es una elección estratégica para nuestro proyecto. Esta configuración nos permite aprovechar un stack tecnológico coherente y eficiente, asegurando al mismo tiempo la simplicidad en el desarrollo y la gestión de la aplicación.

2.3. Diseño de la Interfaz de Usuario:

Se desarrollaron los diseños de la interfaz de usuario (UI) y la experiencia de usuario (UX), empleando la herramienta Figma, un software libre que facilita la creación colaborativa de prototipos y diagramas interactivos. Estos diseños incluyen:

(UX), incluyendo:

- Creación de wireframes para visualizar el diseño de la aplicación.
- Definición de la estructura de navegación y flujo de interacción

2.3.1 Página de Inicio de Sesión

El wireframe para la página de inicio de sesión está diseñado para ofrecer una interfaz clara y funcional, facilitando a los usuarios el acceso a la aplicación. Este diseño incorpora un campo para ingresar el correo electrónico y la contraseña, seguido de un

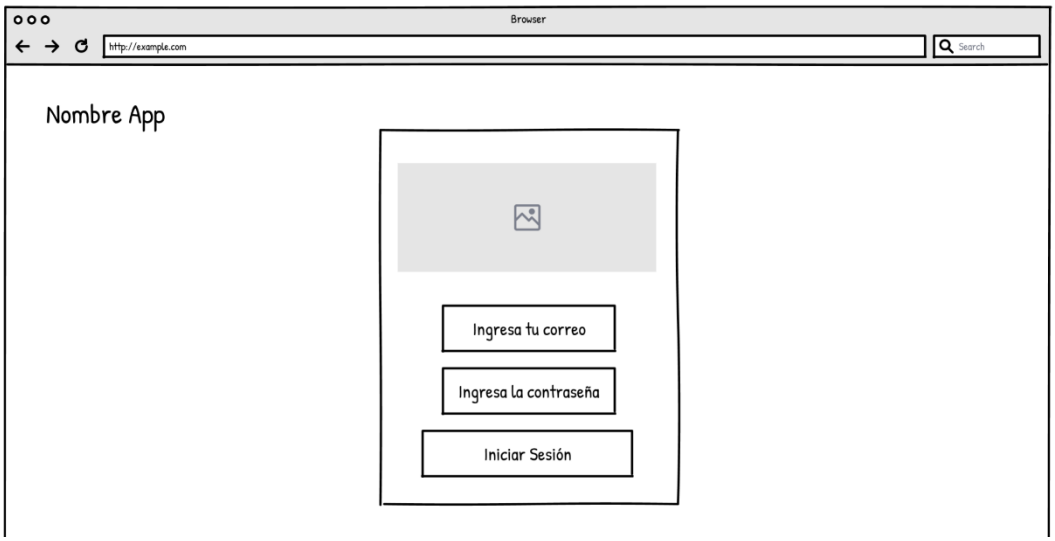

Figura 1 Página de inicio de sesión

botón para iniciar sesión, organizados en un formato intuitivo y fácil de usar. El logo de la aplicación está posicionado en la parte superior, reforzando la identidad visual de la marca y proporcionando coherencia estética a la interfaz. El diseño minimalista ayuda a mantener el foco del usuario en la autenticación, asegurando una navegación fluida y sin distracciones.

2.3.2 Pagina al iniciar sesión

El dashboard presentado es el principal punto de interacción para los usuarios tras iniciar sesión, diseñado para ofrecer una visión instantánea de sus métricas de salud y ejercicio más importantes. El diseño incluye:

- **Indicadores Clave**: Tres métricas principales se destacan en la parte superior del dashboard: "Kilocalorías Quemadas", "Km recorridos" y "% de grasa corporal". Estos indicadores están diseñados para proporcionar un acceso rápido y claro al progreso del usuario en sus objetivos de salud y fitness.
- **Tabla de Actividades**: Debajo de los indicadores, una tabla detallada muestra el desglose de actividades, incluyendo tipo de ejercicio, kilocalorías quemadas, tiempo dedicado, repeticiones, kilómetros recorridos y la fecha. Esta tabla permite a los usuarios ver y rastrear su rendimiento en diferentes actividades a lo largo del tiempo.

- **Recordatorio de Hidratación**: Un recordatorio para tomar la cantidad adecuada de agua diaria está visualmente resaltado, fomentando hábitos saludables de hidratación.

Figura 2 Dashboard.

2.3.3 Wireframe de creación de Registro

La página "Crear Registro" está diseñada para permitir a los usuarios documentar sus sesiones de entrenamiento de manera detallada. El wireframe muestra una interfaz de usuario clara y organizada, que incluye:

Figura 3 Wireframe de creación de Registro.

- **Campos Desplegables**: Para seleccionar el "Tipo de entrenamiento" y la "Intensidad de entrenamiento", facilitando a los usuarios clasificar su actividad según las categorías predefinidas.
- **Entradas Numéricas**: Campos específicos para ingresar el número de "Sets", "Repeticiones", "Kilómetros recorridos", y "Kilocalorías Quemadas", proporcionando una manera estructurada de registrar los detalles de cada actividad física.

- **Campo de Tiempo de Duración**: Un espacio para registrar la duración total del entrenamiento, permitiendo a los usuarios monitorizar el tiempo dedicado a cada sesión.
- **Selector de Fecha**: Un componente para seleccionar la fecha del entrenamiento, asegurando que la información se archive correctamente en el tiempo.
- **Área de Comentarios**: Un espacio amplio para que los usuarios añadan cualquier nota o comentario sobre la sesión, lo cual es útil para registrar detalles adicionales o reflexiones personales sobre el entrenamiento.

2.3.4 Wireframe de creación de registro de medidas corporales

La página "Crear Registro Medida Corporal" está diseñada para que los usuarios puedan ingresar de manera sistemática y precisa las medidas esenciales de su cuerpo. Este wireframe muestra una interfaz organizada que incluye:

- **Campos de Medidas Esenciales**: La interfaz cuenta con campos específicos para ingresar el peso en kilogramos, altura en centímetros, y las circunferencias de cadera, pecho, cintura y bíceps en centímetros, proporcionando una manera estructurada de registrar detalles críticos del cuerpo.
- **Campo del Índice de Masa Corporal (IMC)**: Un campo adicional permite a los usuarios ingresar o calcular su IMC, aunque este campo lo hace en automático una vez ingresados los datos de altura y peso, una métrica importante para evaluar la salud general relacionada con el peso.
    - **Fórmula para el IMC**: Para esto necesitamos
        - Peso representado en kilogramos
        - Estatura representada en metros.
        - $IMC = \frac{peso\ (kilogramos)}{altura\ (metros)^2}$

- **Botón de Registro**: Un botón claramente visible en la parte inferior para "Registrar" las medidas, facilitando a los usuarios la confirmación y el almacenamiento de la información introducida.

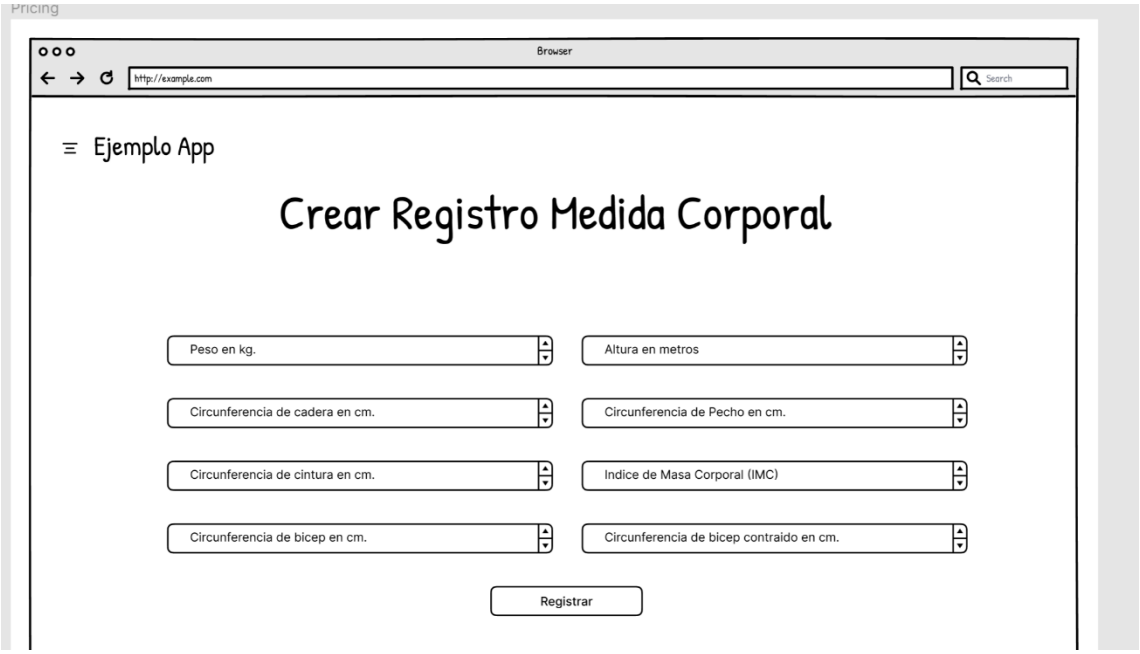

Figura 4 Wireframe de creación de registro de medidas corporales.

2.3.5 Wireframe de calendario de actividades y modal de detalles

Este wireframe presenta un calendario de actividades dentro de la aplicación, diseñado para facilitar a los usuarios la visualización y gestión de sus rutinas de ejercicio programadas y completadas. Los elementos clave incluyen:

Calendario de Actividades

- **Vista Mensual**: Muestra todas las actividades del usuario distribuidas a lo largo de un mes. Cada actividad está codificada por colores para indicar su estado:
    - **Verde**: Actividades completadas.
    - **Naranja**: Actividades pendientes.

- **Interactividad**: Los usuarios pueden hacer clic en cualquier actividad para ver detalles específicos. Esto abre un modal que proporciona más información y opciones adicionales.

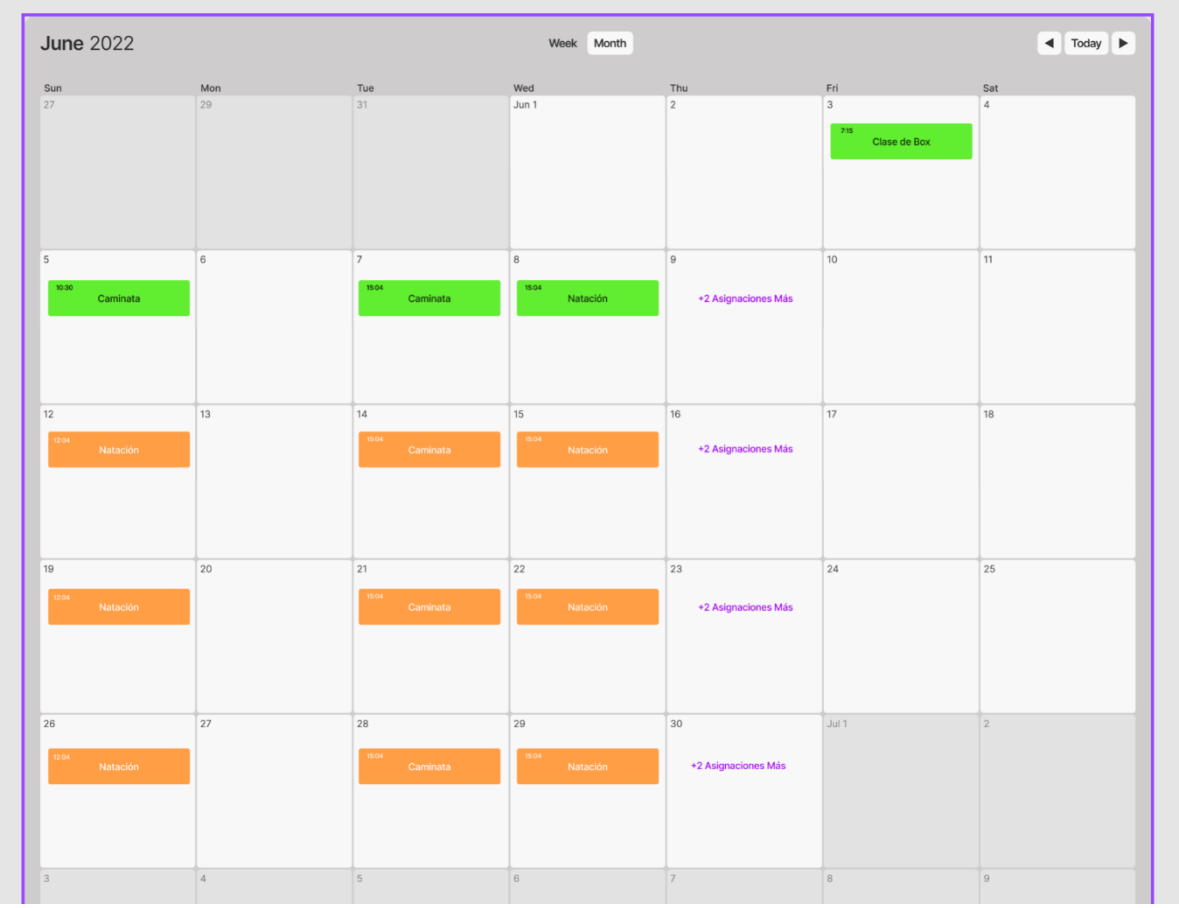

Figura 5 Wireframe de calendario de actividades

Modal de Detalles de Actividad

- **Información Detallada**: Al hacer clic en una actividad, se despliega un modal que muestra información detallada, como la distancia recorrida, el tiempo empleado y las kilocalorías quemadas.
- **Opciones del Modal**:
    - **Botón 'Cancelar'**: Permite al usuario cerrar el modal.

- **Botón 'Ver Análisis'**: Ofrece una opción para ir a una vista más detallada del análisis de la actividad, permitiendo un seguimiento más profundo y la evaluación del rendimiento.

Funcionalidad y Usabilidad

Este diseño no solo ayuda a los usuarios a mantener un seguimiento ordenado de sus actividades físicas, sino que también facilita la gestión y revisión de su progreso. La interfaz intuitiva y la respuesta visual inmediata (modal) mejoran significativamente la experiencia del usuario, alentando un mayor compromiso y regularidad en la realización de ejercicios.

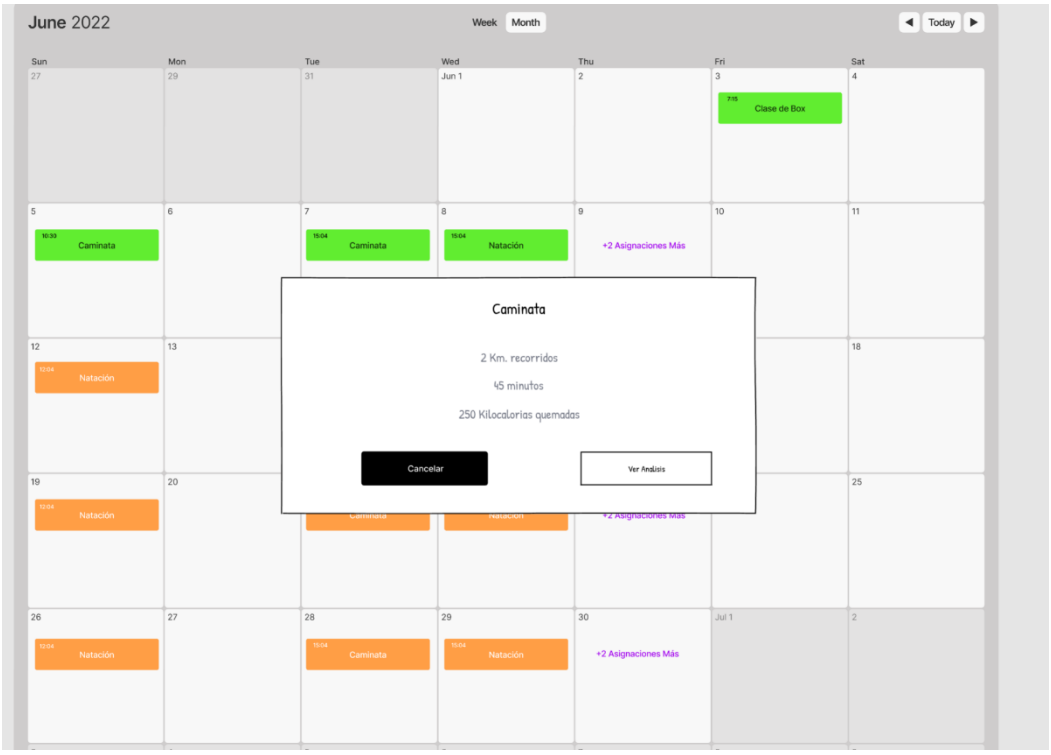

Figura 6 Modal de Detalles de Actividad

2.3.6 *Wireframe análisis de rendimiento*

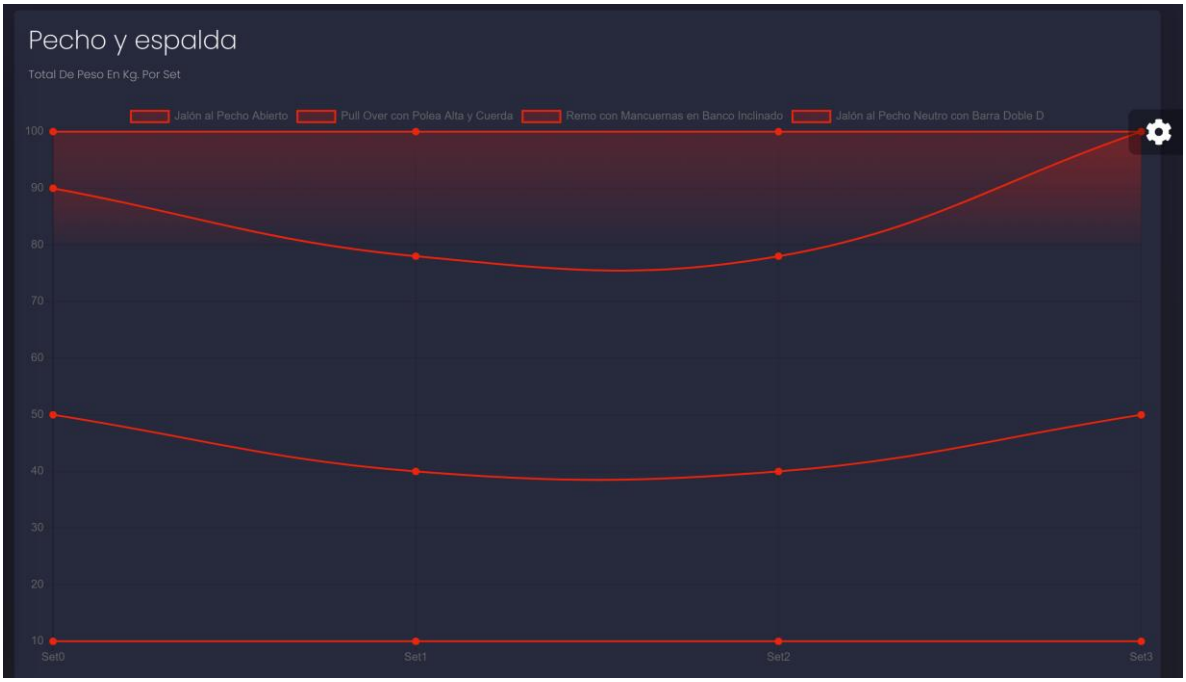

Figura 7 Wireframe análisis de rendimiento

La figura 7 muestra cómo se visualizarán los gráficos de análisis de rendimiento en la aplicación, proporcionando una interfaz intuitiva y funcional para los usuarios que desean monitorear y evaluar su progreso físico a través de diversas métricas.

Estructura del Gráfico

- **Eje X (Series o Tiempo)**: Representa las series de un ejercicio o el tiempo transcurrido durante una actividad. Este eje puede ajustarse para mostrar series consecutivas de ejercicios en el gimnasio o intervalos de tiempo durante una actividad cardio como correr o nadar.

- **Eje Y (Peso o Kilómetros)**: Dependiendo del ejercicio o actividad, este eje muestra el peso levantado en cada serie o los kilómetros recorridos en cada intervalo de tiempo.
- **Líneas de Datos**: Cada línea en el gráfico representa un tipo de ejercicio o actividad, utilizando diferentes colores para facilitar la distinción visual entre los datos.

Interactividad y Funcionalidades

- **Tooltip Detallado**: Al pasar el cursor sobre cualquier punto del gráfico, se despliega un tooltip que ofrece información detallada sobre esa instancia específica, como el peso exacto levantado o la distancia recorrida en ese momento.
- **Comparación de Desempeño**: Los usuarios pueden comparar su rendimiento entre diferentes sesiones o sobre la misma sesión, lo que permite identificar tendencias, mejorar la planificación de entrenamientos y establecer objetivos realistas basados en su progreso.
- **Filtrado de Datos**: Opciones de filtrado permiten a los usuarios visualizar solo los datos que desean revisar, como específicos ejercicios, rangos de fechas, o tipos de actividad.

2.4 Diseño de la Base de Datos:

Se diseña el esquema de la base de datos, que incluye:

- Identificación de las entidades y sus atributos.
- Definición de las relaciones entre las diferentes entidades.
- Diseño de la estructura de la base de datos para garantizar eficiencia y coherencia en el almacenamiento de la información.

2.4.1 Diagrama orientado a documentos

1. **Estructura No Relacional**: Estos diagramas están diseñados específicamente para bases de datos NoSQL como MongoDB. Representan datos en un formato que se asemeja más a JSON o BSON, que es nativo a MongoDB. Esto permite modelar directamente los documentos y las colecciones tal como se almacenarán y se manejarán en la base de datos.

2. **Flexibilidad**: Los diagramas orientados a documentos pueden manejar estructuras de datos flexibles y dinámicas sin esquemas fijos. Esto es especialmente útil para modelar datos que pueden tener campos opcionales, múltiples tipos de datos para un mismo campo, o estructuras de datos anidadas.

3. **Visualización de Relaciones**: Aunque MongoDB es una base de datos NoSQL y no utiliza relaciones en el sentido tradicional de las bases de datos relacionales, los diagramas orientados a documentos pueden mostrar relaciones en forma de referencias y anidamientos, lo cual es crucial para comprender cómo se interconectan los datos.

UML (Unified Modeling Language)

1. **Orientación Relacional**: UML es más común en el diseño de bases de datos relacionales y software orientado a objetos. Utiliza diagramas de clases para representar entidades y relaciones, lo cual es ideal para visualizar la estructura de una base de datos relacional con tablas, filas y columnas.

2. **Estructura y Normalización**: UML ayuda a diseñar sistemas altamente estructurados y normalizados. Esto es útil para evitar la redundancia de datos y mejorar la eficiencia en bases de datos relacionales, pero puede ser menos flexible que los modelos necesarios para las bases de datos NoSQL.
3. **Diagramas de Casos de Uso y Secuencia**: UML también incluye diagramas de casos de uso y secuencia que son útiles para modelar cómo los usuarios interactuarán con el sistema, algo que no está directamente representado en los diagramas orientados a documentos.

Comparación y Uso

- **Aplicación**: Mientras que UML puede ser excesivo y menos directo para proyectos que utilizan MongoDB debido a su naturaleza relacional y orientada a objetos, los diagramas orientados a documentos proporcionan una representación más intuitiva y directa de las estructuras de datos en aplicaciones NoSQL.
- **Visualización**: Los diagramas orientados a documentos pueden ser más sencillos y directos para visualizar estructuras de datos complejas y no uniformes típicas en las aplicaciones modernas web y móviles que utilizan MongoDB.

En resumen, la elección entre UML y diagramas orientados a documentos depende en gran medida del tipo de base de datos y de las necesidades específicas del proyecto. Para proyectos que utilizan MongoDB, los diagramas orientados a documentos son generalmente más adecuados y proporcionan una representación más fiel de cómo se manejarán los datos en la aplicación.

2.4.2 Implementación de la base de datos

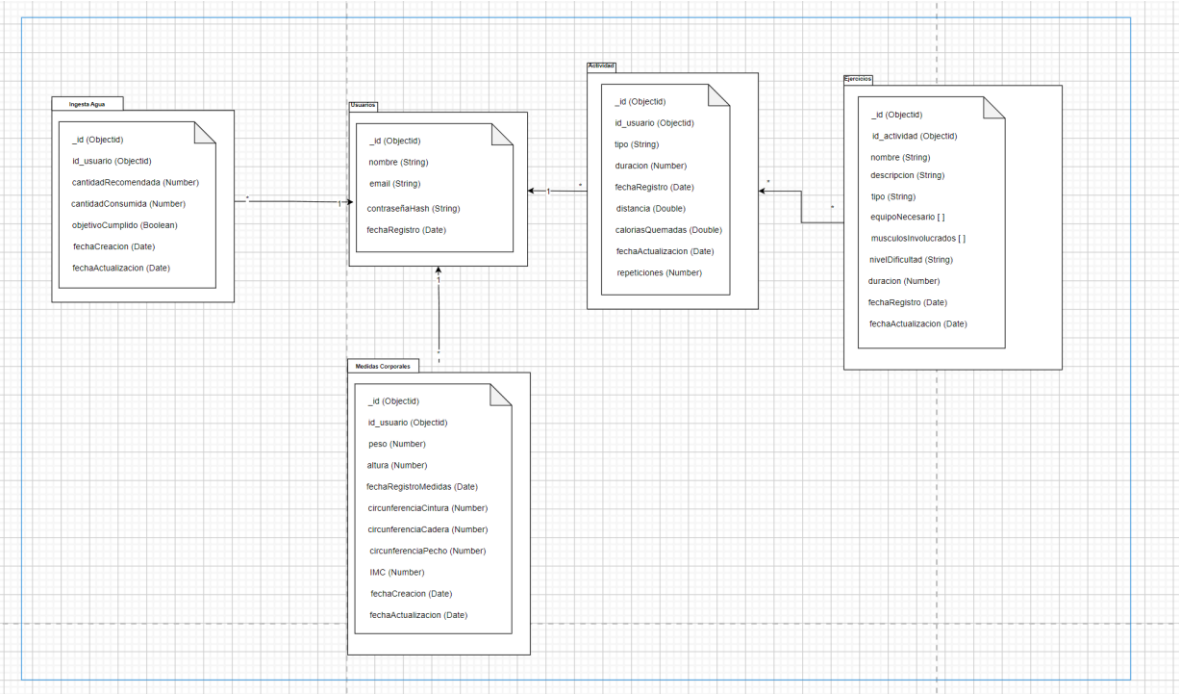

Figura 8 Diagrama de implementación de la base de datos del sistema.

En la figura 8 se presenta el diagrama de implementación de la base de datos del sistema, donde se definen las entidades, atributos y relaciones necesarios para el almacenamiento y gestión de la información.
La estructura está compuesta por colecciones que representan usuarios, rutinas, actividades, medidas corporales y registros de progreso, permitiendo organizar los datos de forma coherente y optimizar las consultas.

La base de datos diseñada para la aplicación de fitness utiliza MongoDB, una base de datos NoSQL, para manejar eficientemente la variedad y volumen de datos generados y utilizados por la aplicación. Se compone de varias colecciones principales: Usuarios, Actividades, Medidas Corporales, Ejercicios e Ingesta de Agua. Cada colección está optimizada para almacenar y recuperar información específica relacionada con la funcionalidad de la aplicación. Los **Usuarios** almacenan información de autenticación y perfil, mientras que **Actividades** y **Medidas Corporales** registran datos específicos del ejercicio y seguimiento de la salud del usuario, respectivamente. La colección

Ejercicios cataloga los distintos ejercicios disponibles para los usuarios, e **Ingesta de Agua** rastrea la hidratación diaria de los usuarios. Las relaciones entre colecciones están claramente definidas para permitir un análisis integrado y personalizado, facilitando así una experiencia de usuario robusta y personalizada. Estrategias de indexación están implementadas para mejorar la eficiencia de las consultas, crucial para el rendimiento de la aplicación en un entorno de producción.

2.4.2.1 Colección: Usuarios

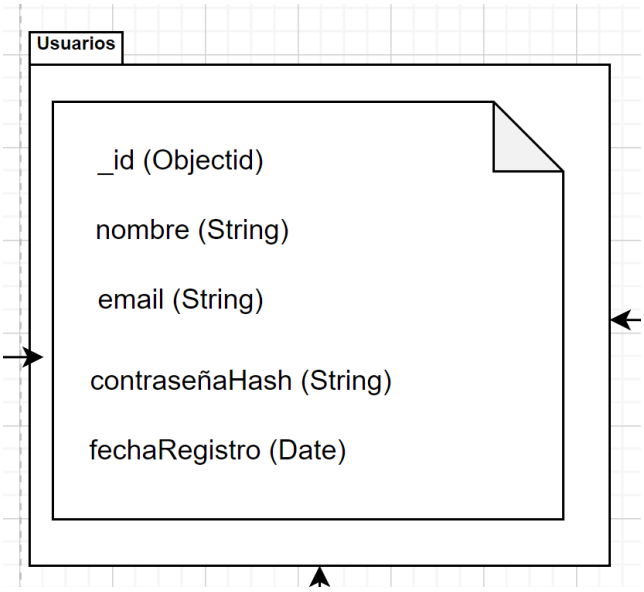

Figura 9 Estructura de la colección Usuarios.

Descripción General: La colección "Usuarios"(ver Figura 9) almacena todos los datos esenciales relacionados con los usuarios de la aplicación. Esta colección es crucial para el manejo de la autenticación y la personalización de la experiencia del usuario.

Estructura de la Colección:

- **_id (ObjectId):** Este es el identificador único generado automáticamente por MongoDB para cada documento en la colección. Es utilizado principalmente para la referencia interna y el acceso eficiente a los datos del usuario.

- **nombre (String):** Almacena el nombre completo del usuario. Este campo es utilizado para personalizar la interacción con el usuario dentro de la aplicación y para identificar al usuario en comunicaciones o informes.
- **email (String):** Dirección de correo electrónico del usuario, que también sirve como identificador único para el inicio de sesión. Este campo es esencial para las operaciones de autenticación y comunicación.
- **contraseñaHash (String):** Almacena un hash de la contraseña del usuario, lo que proporciona una medida de seguridad para almacenar contraseñas de manera segura. Es crucial asegurarse de que este campo utilice técnicas de *hash* robustas como *bcrypt* para proteger la seguridad del usuario.
- **fechaRegistro (Date):** Fecha en la que el usuario se registró en la aplicación. Este campo puede ser útil para analizar la antigüedad de la cuenta del usuario y para llevar registros de la actividad del usuario a lo largo del tiempo.

Funcionalidad y Uso:

- **Autenticación:** La colección es fundamental para verificar las credenciales del usuario durante el proceso de inicio de sesión, utilizando el **email** y **contraseñaHash**.
- **Personalización:** Los datos almacenados en esta colección permiten personalizar la experiencia del usuario dentro de la aplicación, adaptando la interfaz y las funcionalidades al perfil individual.
- **Seguridad:** La gestión segura de la contraseña a través del campo **contraseñaHash** es vital para proteger la cuenta del usuario contra accesos no autorizados.
- **Gestión de Usuarios:** Esta colección también facilita la gestión de usuarios dentro de la aplicación, permitiendo operaciones como la modificación de datos de perfil, recuperación de contraseña y eliminación de cuentas.

Seguridad

1. **Seguridad de Contraseñas:** La contraseña del usuario no se almacena directamente, sino que se guarda como un hash generado a través de un método de hash seguro. Esto asegura que las contraseñas originales nunca se almacenen en texto claro en la base de datos, lo cual es una práctica esencial para la seguridad de la información del usuario.

2. **Proceso de Autenticación:** Durante el proceso de inicio de sesión, la contraseña proporcionada por el usuario se *hashea* utilizando el mismo método y se compara con el hash almacenado en la base de datos. Si ambos hashes coinciden, se concede acceso al usuario, lo que verifica que la contraseña introducida es correcta sin necesidad de desencriptar ninguna información.

2.4.2.2 Colección: IngestaDeAgua

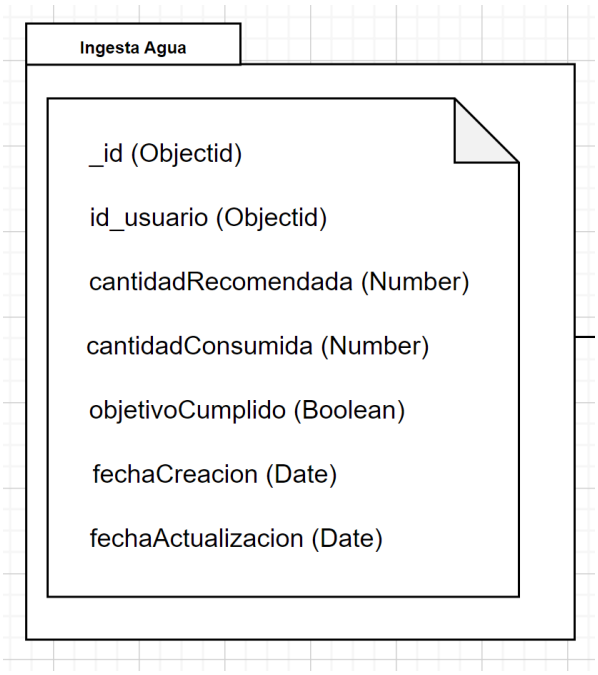

Ilustración 10 Estructura de la colección ingesta de agua

La colección "Ingesta de Agua"(ver Figura 10) es una parte esencial de la base de datos diseñada para la aplicación de fitness, cuyo propósito principal es monitorizar y apoyar la hidratación adecuada de los usuarios. Esta colección ayuda a los usuarios a alcanzar sus objetivos diarios de ingesta de agua, proporcionando un seguimiento continuo y feedback sobre su progreso.

Estructura de la Colección

- **_id (ObjectId)**: Identificador único generado automáticamente por MongoDB para cada registro, que facilita la indexación y recuperación eficiente de los datos.
- **id_usuario (ObjectId)**: Campo que almacena la referencia al documento del usuario en la colección "Usuarios", vinculando cada registro de ingesta de agua a un usuario específico. Esto permite una personalización y seguimiento detallado.
- **cantidadRecomendada (Number)**: Representa la cantidad de agua en mililitros que el usuario debería consumir en un día, según las recomendaciones personalizadas basadas en sus necesidades físicas y objetivos de salud.
- **cantidadConsumida (Number)**: Cantidad de agua que el usuario ha consumido realmente durante el día. Este campo es crucial para monitorizar el cumplimiento de los objetivos de hidratación.
- **objetivoCumplido (Boolean)**: Un campo booleano que indica si el usuario ha cumplido con su objetivo de ingesta de agua recomendada para el día. Esto puede ser utilizado para generar notificaciones o incentivos dentro de la aplicación.
- **fechaCreacion (Date)** y **fechaActualizacion (Date)**: Estos campos registran la fecha y hora en que se creó el registro de ingesta de agua y la última vez que fue

actualizado, respectivamente. Son esenciales para mantener un registro histórico y para asegurar que la información está actualizada.

Funcionalidades y Operaciones

- **Monitoreo Diario**: La colección permite un monitoreo diario del consumo de agua del usuario, apoyando la gestión de la hidratación en tiempo real y ajustes dinámicos basados en la actividad del usuario y las condiciones ambientales.
- **Feedback y Motivación**: Utilizando los datos de **cantidadConsumida** y **objetivoCumplido**, la aplicación puede proporcionar feedback motivacional y sugerencias prácticas para mejorar la ingesta de agua.
- **Análisis de Tendencias**: Los datos acumulados pueden ser analizados para identificar patrones en el comportamiento de hidratación del usuario, ayudando a hacer ajustes personalizados en las recomendaciones de agua.

2.4.2.3 Colección: Actividades

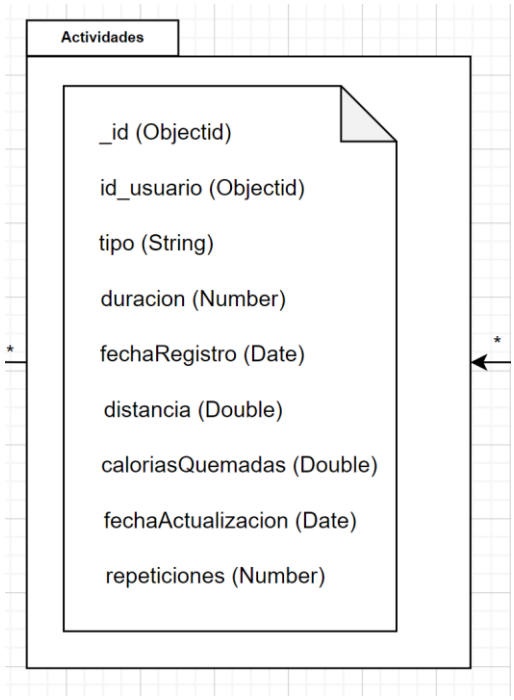

Figura 11 Estructura de la colección: actividades.

La colección "Actividades"(ver Figura 11) en la base de datos MongoDB es crucial para el registro y análisis de las actividades físicas realizadas por los usuarios de la aplicación de fitness. Esta colección almacena detalles específicos de cada actividad, permitiendo a la aplicación proporcionar análisis detallados sobre el rendimiento y progreso del usuario.

Estructura de la Colección

- **_id (ObjectId)**: Identificador único generado automáticamente para cada actividad, facilitando la indexación y recuperación eficiente de datos.
- **id_usuario (ObjectId)**: Enlace al documento del usuario en la colección "Usuarios". Este campo vincula cada actividad registrada a un usuario específico, permitiendo un seguimiento personalizado.

- **tipo (String)**: Tipo de actividad realizada, como "correr", "nadar", o "ciclismo". Este campo ayuda a clasificar las actividades y permite filtrar los datos para análisis específicos por tipo de ejercicio.
- **duración (Number)**: Duración de la actividad en minutos. Este dato es esencial para calcular la intensidad y la eficacia del ejercicio, así como para monitorizar el tiempo total dedicado a la actividad física.
- **fechaRegistro (Date)**: La fecha en que se realizó y registró la actividad. Este campo es crucial para mantener un historial de las actividades del usuario y para realizar comparaciones y tendencias a lo largo del tiempo.
- **distancia (Double)**: Distancia cubierta durante la actividad, medida en kilómetros. Este campo es particularmente relevante para actividades como correr o ciclismo y es fundamental para el cálculo de rendimiento.
- **caloriasQuemadas (Double)**: Estimación de las calorías quemadas durante la actividad. Este dato es vital para los usuarios que siguen un plan de manejo de peso o rendimiento deportivo.
- **fechaActualizacion (Date)**: Fecha en que se actualizó por última vez el registro de la actividad. Esto permite seguir las modificaciones y asegurar que los datos estén siempre actualizados.
- **repeticiones (Number)**: Número de repeticiones realizadas en actividades que lo requieran, como levantamiento de pesas. Este campo complementa la información de la actividad para ejercicios específicos que se miden por repeticiones.

Funcionalidades y Operaciones

- **Seguimiento y Análisis de Actividad**: Esta colección es fundamental para el seguimiento del progreso físico del usuario y permite a la aplicación analizar el

rendimiento a lo largo del tiempo, ajustar planes de entrenamiento y establecer metas realistas basadas en datos históricos.

- **Personalización del Entrenamiento**: Los datos de actividad permiten la personalización de los programas de entrenamiento según las preferencias y capacidades físicas del usuario, optimizando así los resultados de fitness.
- **Motivación y Gamificación**: La información sobre las actividades y logros puede ser utilizada para motivar a los usuarios mediante sistemas de gamificación, donde pueden desbloquear logros y comparar su rendimiento con el de otros usuarios.

2.4.2.4 Colección: Ejercicios

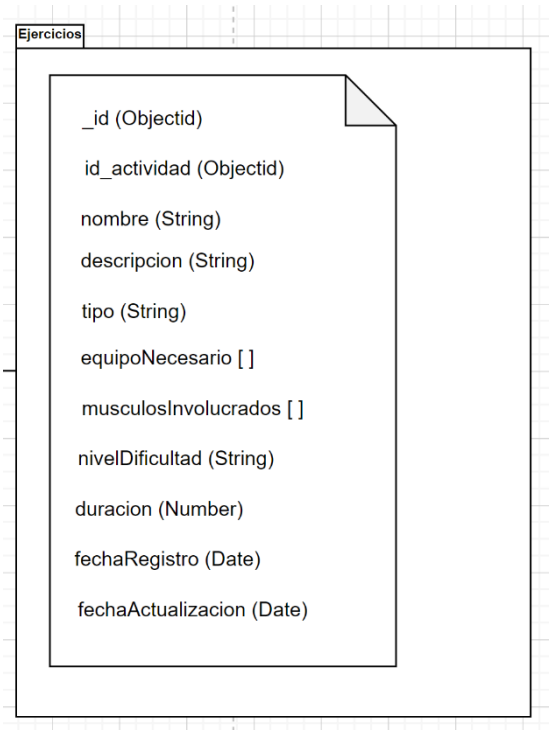

Figura 12 Estructura de la colección: ejercicios.

La colección "Ejercicios" (ver Figura 12) en la base de datos MongoDB de la aplicación de fitness almacena datos detallados sobre los diversos ejercicios disponibles para los

usuarios. Esta colección es esencial para facilitar una amplia variedad de rutinas de entrenamiento y asegurar que los usuarios puedan acceder a información precisa y útil para realizar ejercicios de manera segura y efectiva.

Estructura de la Colección

- **_id (ObjectId)**: Identificador único generado automáticamente para cada ejercicio registrado en la base de datos. Facilita la referencia y recuperación eficiente de información sobre los ejercicios.
- **id_actividad (ObjectId)**: Enlace opcional a una actividad específica en la colección "Actividad". Este campo permite relacionar ejercicios con actividades particulares registradas por los usuarios, aunque su uso puede ser opcional dependiendo de cómo se desee estructurar la interacción entre actividades y ejercicios.
- **nombre (String)**: Nombre del ejercicio, proporcionando una identificación fácil y rápida del tipo de ejercicio para los usuarios y entrenadores.
- **descripcion (String)**: Descripción detallada de cómo realizar el ejercicio, incluyendo técnicas y precauciones para asegurar su ejecución segura.
- **tipo (String)**: Categoría del ejercicio, como cardiovascular, fuerza, flexibilidad, etc., lo que ayuda a los usuarios a seleccionar ejercicios basados en sus objetivos de entrenamiento.
- **equipoNecesario ([String])**: Lista de equipos necesarios para realizar el ejercicio, asegurando que los usuarios estén adecuadamente preparados antes de comenzar.
- **musculosInvolucrados ([String])**: Detalle de los músculos principales que se trabajan con el ejercicio, lo que es útil para usuarios interesados en entrenamientos dirigidos a áreas específicas del cuerpo.

- **nivelDificultad (String)**: Indicador del nivel de dificultad del ejercicio, que puede variar de principiante a avanzado, ayudando a los usuarios a elegir ejercicios adecuados a su nivel de habilidad y condición física.
- **duracion (Number)**: Duración recomendada o típica del ejercicio en minutos, lo que puede guiar a los usuarios en la planificación de sus rutinas.
- **fechaRegistro (Date)** y **fechaActualizacion (Date)**: Estos campos registran las fechas de creación y última actualización del documento del ejercicio, proporcionando un control de versiones y mantenimiento de la actualidad de la información.

Funcionalidades y Operaciones

- **Planificación de Rutinas**: Utilizando la información detallada en esta colección, los usuarios y entrenadores pueden planificar rutinas de ejercicios que se ajusten a necesidades específicas, maximizando la efectividad del entrenamiento.
- **Educación y Seguridad**: Las descripciones detalladas y la información sobre el equipo necesario contribuyen a realizar los ejercicios de forma segura y efectiva, reduciendo el riesgo de lesiones.
- **Motivación y Progreso**: Al proporcionar una variedad de ejercicios con diferentes niveles de dificultad y requerimientos, la colección ayuda a mantener la motivación de los usuarios, permitiendo progresos graduales y medibles.

2.4.2.5 Colección: medidasCorporales

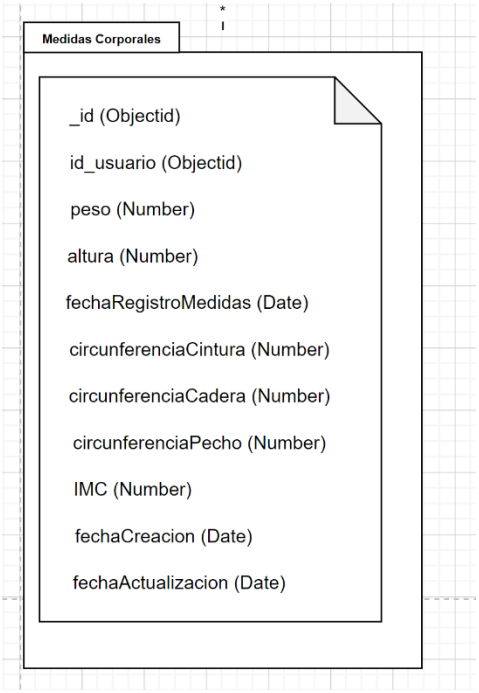

Figura 13 estructura de la colección: medidas corporales.

La colección "Medidas Corporales" (ver Figura 13) en la base de datos de la aplicación de fitness almacena datos cruciales relacionados con las medidas físicas de los usuarios. Esta información permite a la aplicación monitorizar y analizar cambios en la condición física del usuario, ayudando a adaptar programas de entrenamiento y dietas, y a observar tendencias de salud a lo largo del tiempo.

Estructura de la Colección

- **_id (ObjectId)**: Identificador único para cada registro de medida, generado automáticamente por MongoDB. Este identificador es crucial para la indexación y el acceso eficiente a los datos.
- **id_usuario (ObjectId)**: Enlace referencial al usuario correspondiente en la colección "Usuarios". Este campo asegura que cada conjunto de medidas esté claramente asociado con un usuario específico.

- **peso (Number)**: Peso del usuario registrado en kilogramos. Este dato es fundamental para calcular el índice de masa corporal (IMC) y para monitorear cambios en el peso a lo largo del tiempo.
- **altura (Number)**: Altura del usuario en centímetros. Junto con el peso, la altura es utilizada para calcular el IMC.
- **fechaRegistroMedidas (Date)**: Fecha en que se registraron estas medidas. Este campo es esencial para rastrear la evolución de las medidas corporales y asociarlas con otras entradas de datos temporales.
- **circunferenciaCintura, circunferenciaCadera, circunferenciaPecho (Number)**: Medidas específicas que proporcionan datos adicionales sobre la forma y composición corporal del usuario.
- **IMC (Number)**: Índice de Masa Corporal, calculado a partir del peso y la altura. Es un indicador importante de salud general y estado físico.
- **fechaCreacion (Date)** y **fechaActualizacion (Date)**: Estos campos registran la fecha de creación del documento y la última actualización, respectivamente, permitiendo un seguimiento detallado de la historia de medidas del usuario.

Funcionalidades y Operaciones

- **Análisis de Tendencias**: La información almacenada en esta colección se puede utilizar para analizar tendencias de salud y cambios físicos en los usuarios, facilitando ajustes personalizados en los regímenes de entrenamiento y nutrición.
- **Personalización de Programas**: Las medidas corporales detalladas permiten una alta personalización de los programas de fitness, asegurando que los entrenamientos y recomendaciones dietéticas sean apropiados y efectivos para las necesidades individuales del usuario.

- **Motivación y Retroalimentación**: Ver cambios positivos en las medidas corporales puede servir como una herramienta motivacional para los usuarios, mientras que la consistencia en la recolección de datos permite a la aplicación proporcionar retroalimentación específica y oportuna.

2.5 Definición de Componentes y Servicios:

Se identifican los componentes y servicios necesarios para la implementación de la aplicación, así como las funcionalidades específicas que realizarán.

2.6 Planificación del Desarrollo:

Se elabora un plan detallado para el desarrollo de la aplicación, que incluye:

2.6.1 Definición de tareas específicas.

1. **Desarrollo de Wireframes y Prototipos**:
    - **Objetivo**: Crear representaciones visuales de la interfaz de usuario para cada componente, incluyendo inicio de sesión, registro, dashboard, creación de actividades, ingesta de agua, visualización del calendario y análisis de actividades.
    - **Responsable**: Equipo de Diseño UX/UI.
    - **Herramientas**: Herramientas de diseño como Figma.
    - **Entregable**: Wireframes y prototipos completos para cada componente.
2. **Implementación de la Base de Datos**:
    - **Objetivo**: Configurar y diseñar la base de datos conforme a los requisitos definidos y las estructuras necesarias para soportar las funcionalidades del sistema.

- **Responsable**: Equipo de Backend y DBA (Administrador de Base de Datos).
- **Herramientas**: MongoDB, herramientas de modelado de datos, como en este caso mongoose.
- **Entregable**: Esquemas y modelos de datos implementados y documentados.

3. **Desarrollo de Componentes del Frontend**:
    - **Inicio de Sesión y Registro**: Desarrollo de componentes para autenticación de usuarios.
    - **Dashboard**: Creación de un panel de control para visualizar actividades y otras métricas relevantes.
    - **Creación de Actividades**: Interfaz para registrar nuevas actividades físicas.
    - **Ingesta de Agua**: Componente para registrar y monitorizar la ingesta diaria de agua.
    - **Calendario de Actividades**: Visualización del calendario con actividades realizadas.
    - **Análisis de Actividades**: Desarrollo de interfaces para el análisis detallado de actividades físicas.
    - **Conexión a endpoints proporcionados por el backend**: Asegúrese de que los servicios de frontend estén configurados para interactuar con los puntos finales de backend definidos. Maneje operaciones CRUD para actividades, autenticación de usuarios, métricas y seguimiento de consumo de agua.

- **Manejo de tokens**: almacene los tokens de forma segura (por ejemplo, en localStorage), envíelos con cada solicitud relevante y maneje la renovación o el vencimiento de los tokens con precaución ya que son datos sensibles.
- **Cumplir con las restricciones de solicitud**: implementar el manejo de errores para administrar las respuestas del backend, como el acceso no autorizado o el envío de datos no válidos, en angular esto puede ser manejado por guards, nativos del propio framework Angular.
- **Responsable**: Equipo de Desarrollo Frontend.
- **Herramientas**: Angular, ngx-bootstrap, FontAwesome.
- **Entregable**: Componentes funcionales integrados en la aplicación.

4. Desarrollo de componentes del Backend:
    - Autenticación del usuario:
        1. **POST /api/users/register**: Endpoint para el registro de usuarios. Convierte la contraseña en un hash antes de guardarla en la base de datos.
        2. **POST /api/users/login**: Endpoint para el inicio de sesión del usuario. Valida las credenciales y devuelve un JWT si tiene éxito.
    - Actividades del usuario:
        1. POST /api/activities: Endpoint para crear una nueva actividad..
        2. GET /api/activities: Endpoint para recuperar las actividades de un usuario.
        3. GET /api/activities/{id}: Endpoint para recuperar una actividad específica por ID..

4. PUT /api/activities/{id}: Endpoint para actualizar una actividad.
5. DELETE /api/activities/{id}: Endpoint para eliminar una actividad.

- Seguimiento de la ingesta de agua:
    1. POST /api/water-intake: Endpoint para registrar la ingesta de agua.
    2. GET /api/water-intake: Endpoint para recuperar registros de ingesta de agua.
- Métricas y mediciones de usuario:
    1. POST /api/metrics: Endpoint para crear medidas de usuarios.
    2. GET /api/metrics: Endpoint para obtener una lista de las medidas de los usuarios.

- Manejo de sesiones por Middleware
    1. Configure JWT para que caduque después de un cierto período de inactividad (por ejemplo, 1 hora).
    2. Proporcione opciones de renovación automática de sesión o solicite a los usuarios que se vuelvan a autenticar después de que caduque la sesión.

5. Autenticación y seguridad
    - JWT Middleware: para validar JWT en rutas protegidas. Garantiza que el JWT sea válido y no esté caducado en cada petición realizada.

- Contraseñas y Hashing: Utilización *bcrypt* para codificar contraseñas antes de almacenarlas en la base de datos. Esto es crucial para la seguridad.
- Configuración del entorno: Utilización de un archivo .env para almacenar información confidencial como la URL de la base de datos, el secreto JWT y otros parámetros configurables.

6. **Conexión y Esquemas de Base de Datos**:
    - Uso de MongoDB con Mongoose para definir esquemas claros y concisos que reflejen las estructuras de datos necesarias, como usuarios, actividades y registros de ingesta de agua.
7. **Pruebas de Integración y Calidad**:
    - **Objetivo**: Asegurar que todos los componentes funcionen correctamente en conjunto y cumplan con los requisitos establecidos.
    - **Responsable**: Equipo de QA (Aseguramiento de la Calidad).
    - **Herramientas**: Frameworks de pruebas como Jasmine y herramientas de pruebas de integración.
    - **Entregable**: Reportes de pruebas, registro de bugs y validación de funcionalidades.
8. **Despliegue y Puesta en Producción**:
    - **Objetivo**: Desplegar la aplicación en un entorno de producción asegurando su disponibilidad y rendimiento óptimo.
    - **Responsable**: Equipo de Operaciones y Desarrollo.
    - **Herramientas**: Herramientas de CI/CD como Jenkins, entornos de alojamiento como AWS o Azure.

- **Entregable**: Aplicación desplegada y operativa, documentación de despliegue.

Consideraciones Adicionales

- **Documentación Continua**: Cada equipo deberá documentar sus procesos y resultados para asegurar la trazabilidad y facilitar las fases de mantenimiento y escalabilidad futuras.
- **Revisión Periódica**: Implementar reuniones regulares para revisar el progreso, ajustar la planificación si es necesario y resolver impedimentos.

2.6.2 Establecimiento de plazos de entrega.

Integración de la Metodología Ágil SCRUM en la Planificación del Proyecto

Dado que estamos utilizando la metodología ágil SCRUM, los plazos de entrega y las fases de desarrollo se organizarán en sprints, que son períodos cortos y consistentes durante los cuales se completan conjuntos predefinidos de funcionalidades. Esto permite una mayor flexibilidad y adaptabilidad en el desarrollo, ofreciendo la oportunidad de ajustar el producto basado en el feedback continuo y los cambios en los requisitos del proyecto.

Planificación de Sprints

Cada sprint típicamente dura entre 2 y 4 semanas. A continuación, se detalla cómo se pueden dividir las tareas principales en sprints utilizando SCRUM:

1. **Sprint 1: Investigación y Diseño de Wireframes**
    - **Objetivo**: Completar los wireframes y prototipos iniciales.
    - **Duración**: 4 semanas.
    - **Tareas Incluidas**:

- Creación de wireframes para cada componente principal.
- Revisión y ajustes basados en el feedback del equipo y stakeholders.

2. **Sprint 2 y 3: Implementación de la Base de Datos y Backend**
    - **Objetivo**: Establecer la arquitectura de base de datos y desarrollar las funcionalidades básicas del backend.
    - **Duración**: 3 semanas cada sprint.
    - **Tareas Incluidas**:
        - Configuración inicial de MongoDB y creación de esquemas con Mongoose.
        - Desarrollo de endpoints para autenticación y manejo de usuarios.
3. **Sprint 4 a 6: Desarrollo Frontend y Backend Continuo**
    - **Objetivo**: Desarrollar y integrar los componentes del frontend con los servicios del backend.
    - **Duración**: 2 semanas cada sprint.
    - **Tareas Incluidas**:
        - Implementación del inicio de sesión, registro, y dashboard.
        - Creación de la funcionalidad de actividades y registro de ingesta de agua.
4. **Sprint 7: Pruebas y Ajustes**
    - **Objetivo**: Realizar pruebas exhaustivas y asegurar la calidad del software.
    - **Duración**: 4 semanas.

- **Tareas Incluidas**:
    - Pruebas de integración y funcionales.
    - Corrección de errores y optimización de rendimiento.

5. **Sprint 8: Despliegue y Retroalimentación Inicial**
    - **Objetivo**: Lanzamiento de la aplicación en un entorno de producción.
    - **Duración**: 2 semanas.
    - **Tareas Incluidas**:
        - Despliegue final y monitoreo de la aplicación.
        - Recolección y análisis de feedback inicial de los usuarios.

Ceremonias SCRUM Claves

- **Daily Stand-ups**: Reuniones diarias cortas para discutir avances y obstáculos.
- **Sprint Planning**: Planificación detallada al inicio de cada sprint para definir las tareas y objetivos.
- **Sprint Review**: Revisión al final de cada sprint para presentar las funcionalidades desarrolladas.
- **Sprint Retrospective**: Reflexión sobre el sprint completado para identificar mejoras en el proceso.

Rol de SCRUM Master y Product Owner

- **SCRUM Master**: Facilita las ceremonias SCRUM, asegura que el equipo siga las prácticas ágiles y ayuda a resolver impedimentos.
- **Product Owner**: Define los requisitos del producto, prioriza las tareas del backlog y actúa como enlace con los stakeholders.

2.6.3 Asignación de recursos necesarios para cada tarea.

La asignación de recursos es importante ya que nos ayuda a garantizar que todas las tareas planificadas se completen de manera efectiva y eficiente. A continuación, se detalla cómo se asignarán los recursos para cada tarea en el proyecto, asegurando que cada equipo tenga las herramientas, el personal y el apoyo necesarios para alcanzar sus objetivos.

Recursos Humanos

1. **Equipo de Diseño UX/UI:**
    - **Recursos:** 3 diseñadores gráficos y 2 diseñadores UX.
    - **Herramientas:** Licencias para Figma, Adobe XD y herramientas de prototipado como InVision.
2. **Equipo de Backend y BD:**
    - **Recursos:** 4 desarrolladores backend y 1 Administrador de Base de Datos.
    - **Herramientas:** Licencias para MongoDB (aunque una cuenta gratuita puede funcionar), entornos de desarrollo integrado (IDEs) y acceso a servidores para desarrollo y testing.
    - **Apoyo adicional:** Cursos de capacitación en seguridad de bases de datos y optimización de consultas.
3. **Equipo de Desarrollo Frontend:**
    - **Recursos:** 5 desarrolladores frontend.
    - **Herramientas:** Licencias para Angular, ngx-bootstrap, FontAwesome, y acceso a plataformas de testing de frontend como BrowserStack.

Recursos Técnicos

1. **Servidores y Almacenamiento:**
    - **Propósito:** Hosting de la aplicación y almacenamiento de backups.

- **Especificaciones:** Alquiler de servidores en AWS o Azure con configuraciones apropiadas para balance de carga, redundancia y recuperación ante desastres.

2. **Software y Herramientas de Desarrollo:**
   - **Propósito:** Desarrollo, testing y despliegue de la aplicación.
   - **Especificaciones:** Licencias para IDEs o editores de código como Visual Studio Code, herramientas de control de versiones como Git, y plataformas de CI/CD como Jenkins.

Recursos Financieros

- **Presupuesto para Desarrollo:** Estimación de costos basada en salarios, compra de hardware y software, y otros gastos operativos.
- **Fondo de Contingencia:** Reserva de un porcentaje del presupuesto total para imprevistos o ajustes en el proyecto.

Recursos de Capacitación y Desarrollo

- **Capacitaciones Técnicas:** Workshops y cursos online para mejorar habilidades en nuevas tecnologías y metodologías empleadas en el proyecto.
- **Seminarios de Soft Skills:** Entrenamientos en gestión del tiempo, comunicación y liderazgo para mejorar la colaboración y eficiencia del equipo.

Planificación Temporal de Recursos

- **Planificación Detallada:** Asignación de recursos al inicio de cada sprint, revisión durante las reuniones de planificación de sprint y ajustes según las necesidades que surjan en las retrospectivas y revisiones de sprint.

Evaluación y Ajuste de Recursos

- **Evaluación Continua:** Revisión del rendimiento del equipo y de los recursos al final de cada sprint para asegurar la adecuación de los recursos asignados y hacer ajustes necesarios basados en el progreso del proyecto y los feedbacks del equipo.

Capítulo 3: Construcción

3.1. construcción en Angular

3.1.1. Uso de Angular en el Frontend

Para el desarrollo del frontend de nuestra aplicación, hemos optado por utilizar Angular, un framework moderno y robusto que es ideal para la creación de aplicaciones web dinámicas y escalables. Angular no solo facilita la construcción de interfaces ricas

e interactivas, sino que también proporciona una estructura coherente a través de su adopción del patrón Modelo-Vista-Controlador (MVC), aunque con algunas adaptaciones propias.

¿Qué es MVC?

Modelo-Vista-Controlador (MVC) es un patrón de diseño arquitectónico que separa una aplicación en tres componentes principales:

- **Modelo**: Representa la lógica de datos de la aplicación. En Angular, esto incluye la gestión de datos y la lógica de negocio, donde los modelos manejan la carga, la persistencia y la validación de los datos.
- **Vista**: Es la interfaz de usuario que presenta los datos (el modelo) al usuario. En Angular, las vistas son plantillas (usualmente escritas con HTML mejorado) que se renderizan con los datos provenientes del modelo.
- **Controlador**: Actúa como un intermediario entre el modelo y la vista, manejando la lógica de entrada del usuario y la actualización de la vista. En Angular, esta lógica es manejada por los componentes y servicios que pueden alterar datos y comportamientos en la vista.

Angular y su Adaptación de MVC

Angular adapta el patrón MVC de una manera que es más adecuada para el desarrollo web moderno:

- **Componentes**: En Angular, los componentes son la unidad principal de desarrollo. Un componente controla una parte de la pantalla (una vista) a través de su clase asociada que maneja la lógica y los datos. Los componentes facilitan la reutilización del código y la separación de preocupaciones.
- **Servicios**: Los servicios en Angular son clases que se encargan de la lógica de negocio que no está relacionada directamente con las vistas. Estos proporcionan una manera de organizar y compartir código a través de la

aplicación. Los servicios pueden ser inyectados en los componentes mediante la inyección de dependencias de Angular, una característica poderosa que promueve la eficiencia y la modularidad.

- **Enlace de Datos**: Angular mejora la sincronización entre el modelo y la vista mediante el uso de enlace de datos bidireccional. Esto significa que los cambios en el modelo de datos se reflejan automáticamente en la vista y viceversa, facilitando el desarrollo y reduciendo la posibilidad de errores.

Ventajas de Usar Angular y MVC

- **Mantenibilidad**: La clara separación de responsabilidades facilita la gestión y el mantenimiento del código.
- **Modularidad**: La arquitectura basada en componentes y servicios hace que las aplicaciones sean más modulares y fáciles de escalar.
- **Productividad**: Angular ofrece un conjunto extenso de funcionalidades listas para usar, como enrutamiento, gestión de estado, y más, lo que puede acelerar significativamente el desarrollo de la aplicación.
- **Interactividad**: El enlace de datos y la detección de cambios eficiente permiten crear interfaces de usuario dinámicas y reactivas.

3.1.1.1. Bootstrap

En el desarrollo del frontend de nuestra aplicación, además de utilizar Angular, hemos integrado Bootstrap con su preprocesador SCSS y FontAwesome para la gestión de estilos e iconos, respectivamente. Estas herramientas nos permiten crear interfaces de usuario estéticamente agradables, responsivas y accesibles.

3.1.1.1.1. Bootstrap y SCSS

Bootstrap es un framework de front-end ampliamente utilizado que facilita el diseño de sitios web responsivos y móviles primero a través de una extensa biblioteca de componentes prediseñados. Utilizando su **versión SCSS**, que es un preprocesador de CSS, nos permite aprovechar variables, mixins y funciones para escribir estilos más dinámicos y mantenibles.

- **Personalización**: SCSS nos permite personalizar Bootstrap de una manera más estructurada y modular. Podemos ajustar colores, tamaños de fuente y otros elementos visuales a través de variables SCSS, lo que facilita mantener un tema consistente a través de toda la aplicación.
- **Eficiencia**: Al usar SCSS, podemos escribir hojas de estilo más claras y concisas con el uso de funciones avanzadas como anidamiento, herencia y operaciones matemáticas, lo que mejora la legibilidad y mantenimiento del código.

3.1.1.1.2. Integración de ngx-bootstrap en Angular

Para mejorar la interfaz de usuario y facilitar el desarrollo con Angular, hemos optado por integrar **ngx-bootstrap** y **FontAwesome**. Ambas bibliotecas ofrecen herramientas útiles para la creación y gestión de componentes visuales ricos y accesibles.

Integración de ngx-bootstrap

ngx-bootstrap proporciona componentes nativos de Bootstrap 3 y Bootstrap 4 para Angular, sin depender de jQuery. Al usar **ngx-bootstrap**, podemos incorporar fácilmente la funcionalidad de Bootstrap en nuestra aplicación Angular utilizando módulos Angular dedicados.

Instalación de Dependencias:

Para integrar **ngx-bootstrap** en un proyecto Angular, primero necesitamos instalar el paquete a través de npm:

```
npm install ngx-bootstrap --save
```

Posteriormente, se pueden importar módulos específicos de **ngx-bootstrap** en nuestro módulo Angular (por ejemplo, **ModalModule**, **AlertsModule**) según sea necesario

Esta integración permite utilizar los componentes de Bootstrap directamente en las plantillas Angular sin necesidad de cargas adicionales de jQuery, haciendo que la aplicación sea más ligera y rápida.

Para más detalles sobre la instalación y el uso de **ngx-bootstrap**, visita la página oficial de ngx-bootstrap. https://valor-software.com/ngx-bootstrap/#/

3.1.1.2. Fontawesome

FontAwesome es una biblioteca popular de iconos y herramientas gráficas que se integra fácilmente con cualquier tecnología web. Proporciona acceso a miles de iconos que se pueden personalizar en tamaño, color y sombra, entre otras propiedades, directamente a través de clases CSS.

- **Variedad de Iconos**: FontAwesome ofrece una amplia gama de iconos que cubren diversas categorías, desde interfaces de usuario hasta iconos de redes sociales, lo que nos permite encontrar el icono perfecto para casi cualquier función de nuestra aplicación.
- **Accesibilidad y Rendimiento**: Los iconos son accesibles y compatibles con lectores de pantalla, mejorando la usabilidad para todos los usuarios. Además, al ser vectoriales, los iconos se escalan y se mantienen nítidos en cualquier resolución de pantalla.

Integración en Angular:

FontAwesome ofrece una amplia gama de iconos accesibles que se pueden incorporar fácilmente en cualquier proyecto web. Para integrar FontAwesome en Angular, una forma común es incluir el enlace al kit FontAwesome en el archivo **index.html** del proyecto Angular:

```
<head>
  <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/all.min.css" rel="stylesheet">
</head>
```

Este método de integración garantiza que todos los iconos de FontAwesome estén disponibles globalmente en la aplicación Angular, permitiendo su uso en cualquier componente con solo agregar la clase apropiada en el elemento HTML:

```
<i class="fas fa-user"></i> <!-- Icono de usuario -->
```

3.1.2. Desarrollo e Integración de Componentes Frontend

En esta sección, nos centraremos en la implementación y la integración de los componentes frontend, utilizando Angular junto con herramientas complementarias como Bootstrap, ngx-bootstrap, y FontAwesome para optimizar la interfaz de usuario y mejorar la interactividad de la aplicación.

3.1.2.1. Implementación de Componentes Angular

Angular, al ser un framework estructurado y modular, nos permite construir componentes reutilizables que se integran perfectamente con los servicios para manejar la lógica de negocio y la interacción con el backend. Cada componente en

Angular se desarrolla con una combinación de HTML para la estructura, CSS/SCSS para el estilo y TypeScript para la lógica, facilitando una separación clara de responsabilidades y mejorando la mantenibilidad del código.

Componentes Principales:

- **Componente de Autenticación**: Incluye formularios de inicio de sesión y registro, utilizando validaciones reactivas para mejorar la experiencia del usuario.
- **Dashboard**: Un panel de control que muestra resúmenes y estadísticas, como actividades recientes y progreso en metas de fitness.
- **Gestión de Actividades**: Permite a los usuarios añadir, modificar y eliminar actividades físicas.
- **Registro de Ingesta de Agua**: Facilita el seguimiento diario del consumo de agua, integrando alertas para fomentar una hidratación adecuada.
- **Calendario de Actividades**: Visualiza las actividades en un formato de calendario, permitiendo una fácil planificación y revisión.

3.1.2.2. Uso de ngx-bootstrap y SCSS

Para asegurar que la aplicación sea responsiva y estéticamente agradable, utilizamos ngx-bootstrap que se integra con Angular sin necesidad de jQuery, proporcionando una variedad de componentes listos para usar, como modales, alertas y tooltips, que son personalizables y fáciles de implementar. SCSS, siendo un preprocesador de CSS, permite un mayor control sobre los estilos con funcionalidades como variables y mixins, lo que simplifica el manejo de temas y estilos en gran escala.

Integración:

- Se instala ngx-bootstrap y se configura en el módulo principal de Angular para garantizar que los componentes de Bootstrap estén disponibles en toda la aplicación.
- Se utiliza SCSS para estructurar los estilos de forma jerárquica y modular, facilitando la coherencia en el diseño visual en todos los componentes.

3.1.2.3. Integración de FontAwesome

FontAwesome se utiliza para incorporar iconos que mejoran la interfaz de usuario y la hacen más intuitiva. Los iconos se añaden directamente en los componentes Angular utilizando enlaces en el archivo **index.html**, lo que permite su uso global en toda la aplicación.

Uso de Iconos:

- Los iconos de FontAwesome se utilizan en botones, menús y en el dashboard para mejorar visualmente la presentación de información y acciones disponibles.

3.1.2.4. Ventajas y Beneficios

- **Mantenibilidad y Escalabilidad**: Angular y sus herramientas asociadas facilitan la gestión del código y permiten una fácil expansión de la aplicación.
- **Productividad y Eficiencia**: La reutilización de componentes y la consistencia en el diseño aumentan la productividad del equipo de desarrollo.
- **Interactividad y Experiencia del Usuario**: La interfaz rica e interactiva garantiza una experiencia de usuario agradable y funcional.

3.1.2.5. Conclusión

La construcción del frontend, utilizando Angular junto con tecnologías complementarias como ngx-bootstrap y FontAwesome, no solo proporciona una base sólida para el desarrollo eficiente, sino que también, asegura que la aplicación final sea robusta, intuitiva y fácil de usar. Estas tecnologías han sido elegidas no solo por su

compatibilidad y rendimiento sino también por su amplio soporte y comunidad activa, asegurando que la aplicación se mantenga actualizada con las mejores prácticas y tecnologías modernas.

3.2. Construcción del Backend

En el desarrollo del backend de nuestra aplicación, hemos optado por utilizar tecnologías modernas y eficientes que facilitan la creación de una infraestructura robusta y escalable, centrada en la seguridad, la gestión de datos y la integración con el frontend.

3.2.1. Uso de Node.js y Express

Node.js es un entorno de ejecución para JavaScript del lado del servidor que nos permite construir aplicaciones de red rápidas y escalables. Express es un framework para Node.js que simplifica el manejo de rutas y solicitudes HTTP, haciendo más eficiente el desarrollo del backend.

Características Clave:

- **Asincronía**: Node.js opera sobre un modelo de I/O no bloqueante y orientado a eventos, que lo hace especialmente adecuado para aplicaciones que requieren mucha lectura y escritura en la red.
- **Middleware**: Express permite utilizar middleware que puede acceder a la solicitud y respuesta, y modificarlas o ejecutar código específico entre la recepción de la solicitud y el envío de la respuesta, lo cual es útil para autenticación, manejo de errores, y más.

3.2.2. Autenticación y seguridad

Para manejar la autenticación de los usuarios y asegurar la seguridad de la aplicación, implementamos varias estrategias:

- **JWT (JSON Web Tokens)**: Utilizamos tokens para manejar las sesiones de usuario. Los tokens se generan al iniciar sesión y se verifican en cada solicitud para confirmar que el usuario está autenticado.
- **Bcrypt**: Para la seguridad de las contraseñas, utilizamos Bcrypt para hashear las contraseñas antes de almacenarlas en la base de datos, proporcionando una capa adicional de seguridad contra ataques de fuerza bruta.

3.2.3. Integracion de MongoDB con Mongoose

MongoDB es una base de datos NoSQL que nos permite almacenar documentos en un formato flexible y escalable, lo que es ideal para nuestras necesidades de datos variados y en constante cambio.

- **Mongoose**: Es una biblioteca para MongoDB que facilita la modelación de datos a través de esquemas robustos que permiten validar los datos antes de guardarlos en la base de datos.

Ejemplo de un Esquema de Usuario:

```
const userSchema = new mongoose.Schema({
  username: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  passwordHash: { type: String, required: true },
  registrationDate: { type: Date, default: Date.now }
```

});

3.2.4. implementación de Endpoints API

Desarrollamos endpoints específicos que manejan las operaciones CRUD para usuarios, actividades y otras funciones de la aplicación:

- **Endpoints de Usuario**: Incluyen registro y login, y manejo de perfiles.
- **Endpoints de Actividades**: Permiten a los usuarios crear, modificar, y borrar actividades registradas.
- **Endpoints de Ingesta de Agua y Métricas**: Controlan el registro y consulta de la ingesta diaria de agua y las métricas de salud de los usuarios.

3.2.5. Middleware para la gestión de sesiones

Implementamos middleware en Express para manejar sesiones de usuario:

- **Manejo de Tokens**: Configuramos JWT para expirar después de un periodo de inactividad y permitimos la renovación automática para mantener la sesión del usuario.

3.2.6. Conclusión

El backend de nuestra aplicación se construye con un enfoque en la eficiencia, la seguridad y la escalabilidad, utilizando tecnologías modernas que facilitan la integración con el frontend y ofrecen una experiencia de usuario fluida y segura. Esta infraestructura nos permite adaptarnos rápidamente a los cambios y escalar según las necesidades del proyecto o los requerimientos del usuario.

3.3. Construcción de la Base de Datos

La base de datos de nuestra aplicación es un componente crucial que maneja el almacenamiento, recuperación y seguridad de los datos generados y utilizados por la aplicación. Para el desarrollo de nuestra aplicación, hemos optado por utilizar MongoDB, una base de datos NoSQL que ofrece flexibilidad en el manejo de estructuras de datos complejas y un escalado sencillo.

3.3.1. Creación y Configuración de la Base de Datos en MongoDB

Cuenta y Prueba Gratuita:

- Al iniciar el proyecto, configuramos una cuenta nueva en MongoDB, aprovechando la oferta de prueba gratuita que proporciona capacidad de almacenamiento suficiente para las etapas iniciales del desarrollo sin incurrir en costos adicionales. Esto permite validar la aplicación y su desempeño antes de comprometerse a un plan de pago más amplio.

Configuración de Seguridad:

- **Restricción de IP**: Para asegurar que solo nuestro equipo y servicios autorizados puedan acceder a la base de datos, implementamos restricciones de IP que limitan las conexiones a direcciones IP específicas. Esto reduce significativamente la superficie de ataque y ayuda a prevenir accesos no autorizados.
- **Roles de Usuario**: Configuramos roles específicos dentro de MongoDB para controlar el nivel de acceso que tienen diferentes usuarios y servicios a la base de datos. Esto es crucial para cumplir con las mejores prácticas de seguridad y garantizar que cada servicio o usuario tenga solo el nivel de acceso necesario para realizar sus funciones.

3.3.2. Conexión a la Base de Datos desde el Código

Para conectar nuestra aplicación al backend de MongoDB, utilizamos el driver de MongoDB o Mongoose (un ODM que simplifica las interacciones con MongoDB) dependiendo de las necesidades específicas del proyecto. La conexión se establece a través de cadenas de conexión que utilizan las credenciales y configuraciones definidas en el entorno protegido de la aplicación (a menudo almacenadas en variables de entorno para mayor seguridad).

- **Ejemplo de cadena de conexión con Mongoose**:

```
const mongoose = require('mongoose');

mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => {
  console.log('MongoDB Connected...')
}).catch(err => console.log(err));
```

3.3.3. Conclusión

La construcción y configuración de la base de datos no solo abarca la implementación técnica, sino que también incluye la configuración cuidadosa de aspectos de seguridad y accesibilidad. Utilizar herramientas como MongoDB Compass y configurar adecuadamente la seguridad son pasos fundamentales para asegurar que la base de

datos funcione de manera eficaz y segura, apoyando las necesidades de la aplicación sin comprometer la integridad o la privacidad de los datos.

Capítulo 4: Pruebas y liberación

Este capítulo proporciona un análisis exhaustivo de las pruebas realizadas para asegurar la calidad y funcionalidad de la aplicación web, y describe el proceso seguido para transicionar desde el ambiente de desarrollo hasta el ambiente de pruebas y, finalmente, al de liberación.

4.1. Imágenes del Proyecto

4.1.1. Pantalla de Inicio de Sesión

La Figura 14 muestra la pantalla de inicio de sesión de la aplicación, que es uno de los puntos críticos de interacción del usuario. Esta pantalla incluye campos para el correo electrónico y la contraseña, un enlace para recuperar la contraseña olvidada y un botón para continuar. Es esencial que esta pantalla sea funcional y estéticamente agradable para garantizar una buena experiencia de usuario.

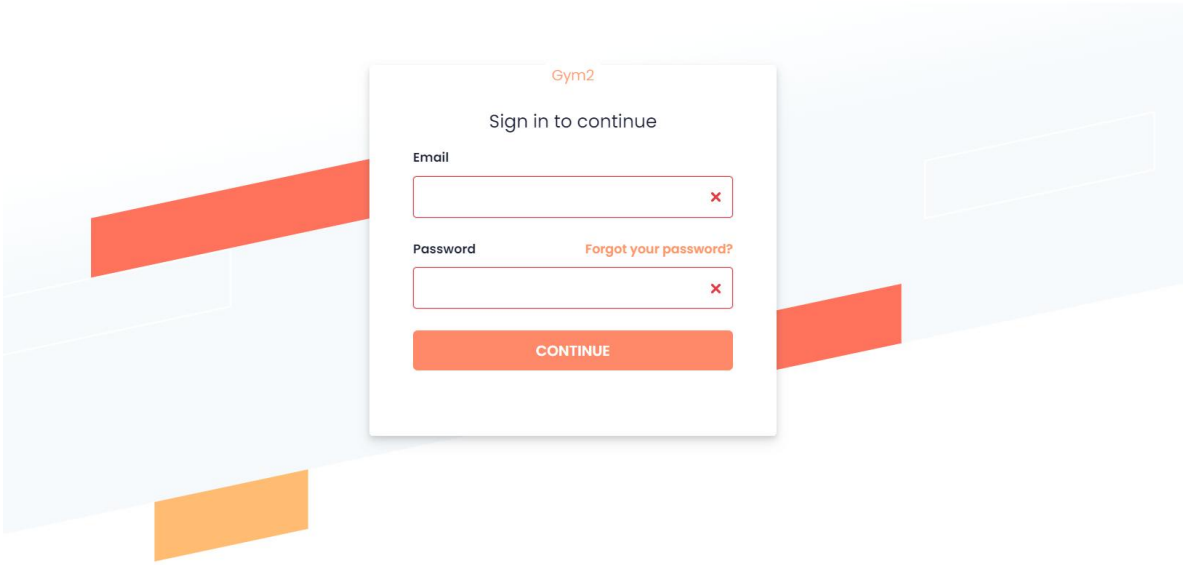

Figura 14 Pantalla de inicio de sesión

4.1.1.1. Descripción del Tipo de Pruebas

Para garantizar la calidad del producto, se realizaron varios tipos de pruebas a lo largo del desarrollo de la aplicación:

- **Pruebas Funcionales:** Verificar que cada función de la aplicación, como el inicio de sesión, registro de datos e ingesta de agua, funciona según lo esperado.
- **Pruebas de Usabilidad:** Asegurar que la aplicación sea intuitiva y fácil de usar para todos los usuarios, especialmente en pantallas críticas como la de inicio de sesión.
- **Pruebas de Rendimiento:** Asegurar que la aplicación maneja cargas de usuario esperadas sin degradación del rendimiento.
- **Pruebas de Compatibilidad:** Confirmar que la aplicación funciona correctamente en diferentes dispositivos y navegadores.

- **Pruebas de Validación de Formulario**:
    - **Pruebas de Validación de Entrada**: Asegurarse de que todos los campos de entrada, como correo electrónico y contraseña, validen correctamente la información del usuario antes de procesarla. Esto incluye pruebas para campos vacíos y formatos no válidos.
    - **Pruebas de Seguridad de Autenticación**: Verificar que el backend maneje correctamente la autenticación utilizando hashing de contraseñas. Esto implica comprobar que la contraseña introducida por el usuario, una vez transformada en hash, coincida con el hash almacenado en la base de datos.

4.1.2. Ingreso de Medidas Corporales

Esta pantalla permite a los usuarios o administradores ingresar y registrar medidas corporales como peso, altura, circunferencias de diferentes partes del cuerpo e Índice de Masa Corporal (IMC). Esta funcionalidad es esencial para asegurar que los usuarios puedan seguir su progreso físico de manera detallada.

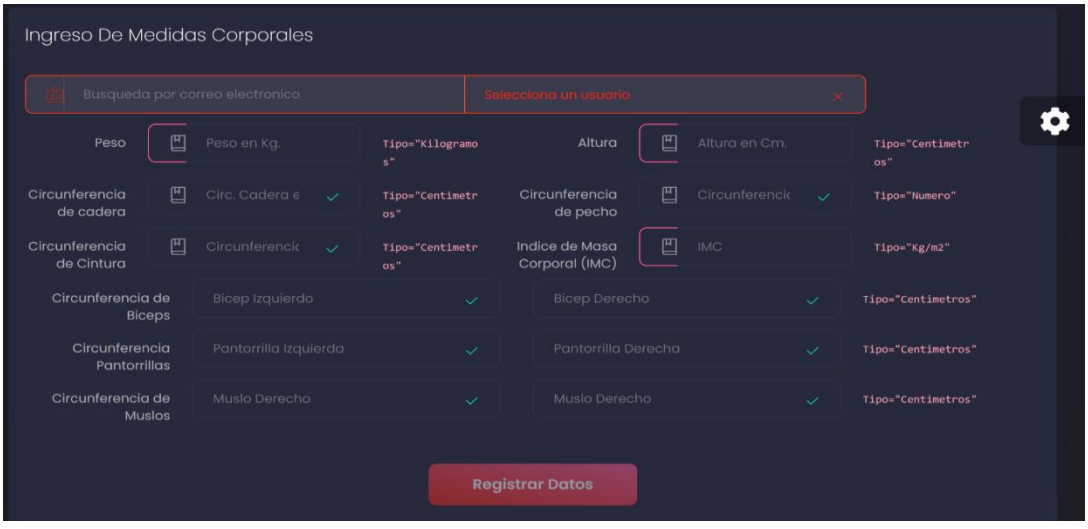

Figura 15 Ingreso de medidas.

4.1.2.1. Descripción del Tipo de Pruebas

Pruebas de Funcionalidad:

- **Pruebas de Validación de Entrada:** Verificar que los campos solo acepten el tipo de datos adecuado (por ejemplo, kilogramos, centímetros, números). Esto incluye pruebas para evitar la entrada de caracteres no válidos y asegurar que los datos se almacenen correctamente en la base de datos.
- **Pruebas de Lógica de Cálculo:** Asegurar que cálculos automáticos, como el del Índice de Masa Corporal, se realicen correctamente basándose en las entradas de peso y altura.

Pruebas de Usabilidad:

- **Pruebas de Interfaz de Usuario:** Confirmar que la interfaz es intuitiva y los elementos de la UI como botones y campos de entrada están accesibles y son fáciles de usar.
- **Pruebas de Respuesta Visual:** Verificar que los indicadores visuales (como los ticks verdes) funcionan correctamente al completar cada campo.

Pruebas de Seguridad:

- **Pruebas de Manejo de Datos Sensibles:** Asegurar que todas las medidas corporales se manejen y almacenen de manera segura, protegiendo la privacidad del usuario.

Pruebas de Rendimiento:

- **Pruebas de Carga:** Evaluar cómo maneja el sistema el ingreso simultáneo de datos por múltiples usuarios, especialmente en configuraciones de gimnasio o

clínica donde múltiples personas podrían estar ingresando información al mismo tiempo.

4.1.3. Dashboard de Administración:

Esta pantalla (ver Figura 16) es esencial para proporcionar una visión general rápida y eficiente de los indicadores clave de rendimiento del sistema, permitiendo a los administradores monitorear y gestionar las actividades dentro de la plataforma con facilidad.

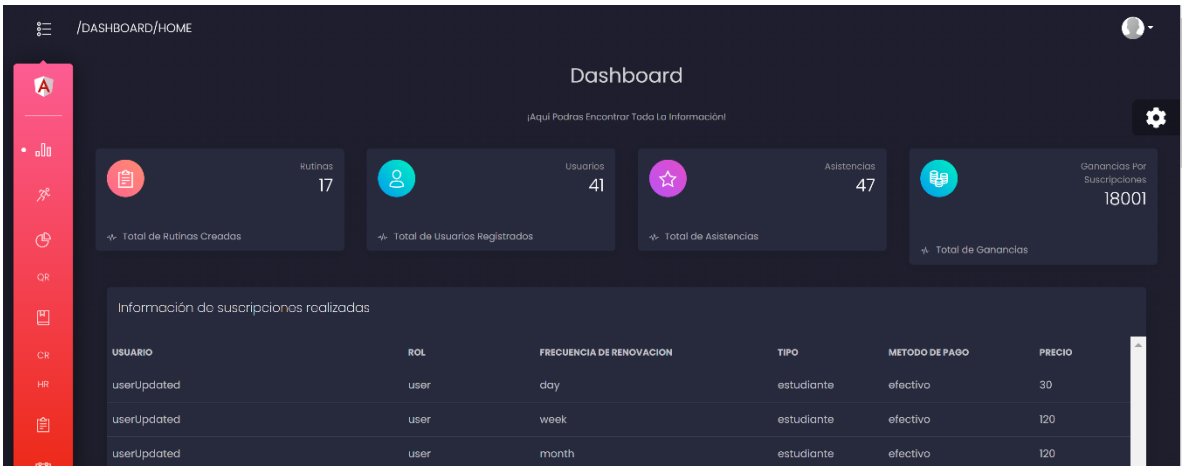

Figura 16 Dashboard de administración.

4.1.3.1. Descripción del Tipo de Pruebas

Pruebas de Funcionalidad:

- **Pruebas de Carga de Datos:** Verificar que todos los widgets del dashboard carguen correctamente los datos desde la base de datos, incluyendo rutinas creadas, usuarios registrados, asistencias y ganancias.
- **Pruebas de Consistencia de Datos:** Asegurar que los datos mostrados en el dashboard sean consistentes y precisos en comparación con los datos almacenados en la base de datos.

Pruebas de Interfaz de Usuario:

- **Pruebas de Comportamiento de Widgets:** Comprobar que todos los componentes interactivos del dashboard, como botones y enlaces, funcionen como se espera.
- **Pruebas de Rendimiento del Dashboard:** Asegurar que la página cargue rápidamente incluso cuando maneje grandes volúmenes de datos.

Pruebas de Seguridad:

- **Pruebas de Acceso Seguro:** Confirmar que solo los usuarios autorizados puedan acceder al dashboard.
- **Pruebas de Roles y Permisos:** Verificar que la visualización de información en el dashboard respete los roles y permisos del usuario, mostrando solo la información que es relevante para el usuario autenticado.

Pruebas de Usabilidad:

- **Evaluación de la Experiencia de Usuario:** Asegurar que el dashboard sea intuitivo y fácil de navegar, ofreciendo una experiencia de usuario positiva.
- **Pruebas de Visualización en Diferentes Dispositivos:** Confirmar que el dashboard sea responsivo y se vea bien en diferentes tamaños de pantalla y dispositivos.

4.1.4. Creación de Rutinas

Esta pantalla es fundamental para el proceso de configuración de rutinas de ejercicios personalizadas, permitiendo a los administradores especificar diversos detalles que personalizan la rutina a las necesidades del usuario.

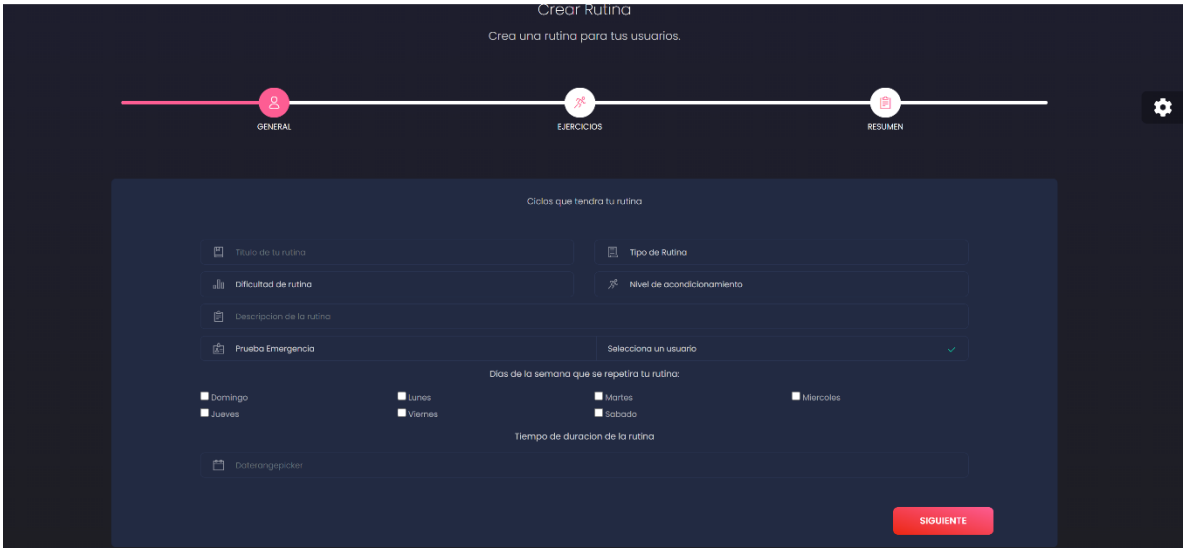

Figura 17 Interfaz de configuración de rutinas personalizadas.

4.1.4.1. Descripción del Tipo de Pruebas

Pruebas de Funcionalidad:

- **Pruebas de Validación de Formulario:** Verificar que todos los campos del formulario validen la entrada según los requisitos especificados, como formatos de fechas y selección de múltiples días.
- **Pruebas de Funciones de Guardado y Recuperación de Datos:** Asegurar que la información ingresada en el formulario pueda guardarse correctamente en la base de datos y recuperarse cuando sea necesario.

Pruebas de Interacción de Usuario:

- **Pruebas de Navegabilidad:** Confirmar que el flujo de creación de la rutina (desde el inicio hasta el resumen y guardado final) sea fluido y libre de errores.

- **Pruebas de Componentes Interactivos:** Comprobar que elementos como selección de días y configuración de tipo de rutina funcionen de manera intuitiva y sin errores.

Pruebas de Usabilidad:

- **Evaluación de la Claridad del Formulario:** Verificar que la disposición y etiquetas de los campos sean claras y entendibles para los usuarios, asegurando una experiencia de usuario eficiente y efectiva.
- **Pruebas de Diseño Responsivo:** Evaluar que la página se adapte y funcione correctamente en diferentes dispositivos y resoluciones de pantalla.

Pruebas de Seguridad:

- **Validación de Entrada del Usuario:** Asegurar que las entradas de los usuarios sean validadas adecuadamente para evitar la inyección de SQL y otros ataques basados en entradas de formulario.

4.1.5. Resumen de Rutina de Ejercicio

Esta pantalla proporciona un análisis visual del desempeño del usuario en la rutina de piernas, mostrando claramente el peso levantado en cada serie para diferentes ejercicios.

Ilustración 18 Interfaz gráfica del resumen de rutina de piernas, mostrando el peso levantado por serie en distintos ejercicios.

4.1.5.1. Descripción del Tipo de Pruebas

Pruebas de Funcionalidad:

- **Pruebas de Precisión de Datos:** Verificar que los datos mostrados en la gráfica correspondan exactamente a los datos registrados durante las sesiones de ejercicio.

- **Pruebas de Rendimiento de la Gráfica:** Asegurar que la gráfica se renderice rápidamente y sin errores, incluso con grandes volúmenes de datos.

Pruebas de Seguridad:

- **Pruebas de Control de Acceso:** Aunque ya está asegurado en el frontend y backend, realizar pruebas adicionales para confirmar que solo los usuarios autorizados (el usuario que realizó la rutina y los administradores) puedan acceder a esta información.

Pruebas de Usabilidad:

- **Evaluación de la Legibilidad de la Gráfica:** Comprobar que la gráfica sea fácil de leer y entender, con etiquetas claras para los ejercicios y los sets.
- **Pruebas de Responsividad:** Asegurar que la página se visualice correctamente en diferentes dispositivos y tamaños de pantalla, proporcionando una experiencia de usuario consistente.

Pruebas de Interfaz de Usuario:

- **Pruebas de Interacción con Gráficos:** Confirmar que los gráficos respondan adecuadamente a las interacciones del usuario, como el paso del ratón sobre los puntos para revelar tooltips.
- **Pruebas de Carga de Datos**: Comprobar que los gráficos se carguen rápidamente y manejen adecuadamente los conjuntos de datos**.**

4.2. Descripción de las Actividades para la Transición entre Ambientes

La transición del ambiente de desarrollo al de pruebas y finalmente al de liberación implica varias actividades clave:

1. **Revisión de Código:** Antes de pasar al ambiente de pruebas, el código es revisado por otros desarrolladores para asegurar que sigue las buenas prácticas de programación y no introduce errores nuevos.
2. **Configuración del Ambiente de Pruebas:** Establecer un ambiente de pruebas que simule el ambiente de producción lo más cercanamente posible para validar el comportamiento de la aplicación.
3. **Ejecución de Pruebas Automatizadas:** Utilizar herramientas de continuous Integration and Continuous Delivery (CI/CD) para ejecutar pruebas automatizadas que verifiquen la funcionalidad, seguridad y rendimiento de la aplicación.
4. **Pruebas Manuales:** Realizar pruebas manuales para explorar funcionalidades que son difíciles de verificar a través de pruebas automatizadas.
5. **Corrección de Errores:** Corregir cualquier problema encontrado durante las pruebas.
6. **Validación Final:** Una vez que la aplicación pasa todas las pruebas en el ambiente de pruebas, se realiza una validación final antes de su liberación.
7. **Despliegue en Producción:** Tras la validación final, la aplicación se despliega en el ambiente de producción.
8. **Monitoreo Post-Liberación:** Monitorear la aplicación en producción para detectar y resolver rápidamente cualquier problema no identificado previamente.

Capítulo 5: Conclusiones

El desarrollo y lanzamiento de esta aplicación han constituido un desafío significativo y a la vez un logro notable para mí. Este proyecto integró tecnologías avanzadas y metodologías ágiles para crear una solución de software robusta, escalable y altamente funcional destinada a mejorar la gestión y el seguimiento de actividades físicas y de salud de los usuarios.

Integración Tecnológica

La elección de Angular para el frontend y Node.js para el backend demostró ser acertada, proporcionando una base sólida para construir una aplicación interactiva y fácil de usar. La integración de MongoDB como sistema de gestión de base de datos nos permitió manejar eficientemente grandes volúmenes de datos no estructurados, mientras que el uso de herramientas como Mongoose facilitó la manipulación de los datos y aseguró la integridad de los mismos. La adopción de ngx-bootstrap y FontAwesome enriqueció la interfaz de usuario, mejorando la estética y la funcionalidad sin comprometer el rendimiento.

Metodología Ágil

La implementación de la metodología SCRUM fue crucial en el manejo eficaz del proyecto. Esta metodología permitió adaptarme rápidamente a los cambios, resolver problemas de manera eficiente y mejorar continuamente el producto a través de iteraciones regulares y feedback constante. Las reuniones de planificación, las revisiones de sprint y las retrospectivas facilitaron una comunicación efectiva y una colaboración estrecha entre todos los miembros del equipo, lo que se tradujo en entregas exitosas y a tiempo.

Pruebas y Calidad

El enfoque exhaustivo en las pruebas, desde pruebas unitarias hasta pruebas de aceptación del usuario, aseguró que la aplicación no solo cumpliera con los requisitos técnicos, sino que también respondiera a las necesidades reales de los usuarios.

Impacto y Recepción

El proyecto ha demostrado su valor al proporcionar una herramienta útil que ayuda a los usuarios a monitorear y gestionar su salud y actividades físicas de manera más efectiva. La recepción inicial de la aplicación ha sido positiva, lo que indica un buen alineamiento con las expectativas del mercado y las necesidades del usuario.

Reflexión Final

La realización de este proyecto ha sido una oportunidad invaluable para aplicar conocimientos teóricos en un contexto práctico, enfrentar y superar desafíos reales de ingeniería de software, y aprender de cada etapa del ciclo de vida del desarrollo de software. Mientras se avanzó hacia la fase de mantenimiento y actualizaciones futuras, continúo comprometiéndome a mejorar y expandir la funcionalidad de la aplicación para satisfacer mejor las demandas cambiantes de nuestros usuarios.

Este proyecto no solo cumplió con sus objetivos iniciales, sino que también sentó las bases para futuras innovaciones en el ámbito del software de salud y fitness, demostrando el poder de la tecnología moderna aplicada al bienestar y la salud personal.

Capítulo 6: trabajo futuro

Este proyecto ha establecido una sólida plataforma de desarrollo y ha demostrado ser eficaz en su propósito actual. Sin embargo, el campo de las aplicaciones de salud y fitness es vasto y en constante evolución, presentando numerosas oportunidades para la expansión y mejora. A continuación, se delinean varias áreas de desarrollo futuro que podrían aumentar significativamente la utilidad, interactividad y efectividad de la aplicación.

6.1. Instrucciones Específicas por Deporte

Propuesta: Integrar un módulo que ofrezca instrucciones específicas y personalizadas para una variedad de deportes, proporcionadas por profesionales en cada disciplina. Esto podría incluir videos, guías paso a paso, y consejos prácticos adaptados a las habilidades y progresos del usuario.

Objetivos:

- Enriquecer la experiencia del usuario al proporcionar contenido educativo y de entrenamiento específico del deporte que practica.
- Fomentar el uso correcto y seguro de las técnicas deportivas para evitar lesiones y mejorar el rendimiento.

6.2. Integración de Inteligencia Artificial

Propuesta: Desarrollar e integrar un sistema de inteligencia artificial que funcione como un asistente virtual motivacional y de asesoramiento. Este sistema podría analizar los datos de actividad del usuario para proporcionar recomendaciones personalizadas, motivación basada en el progreso y ajustes en los planes de entrenamiento.

Objetivos:

- Mejorar la retención y satisfacción del usuario mediante interacciones personalizadas y motivacionales.
- Utilizar el análisis predictivo para ofrecer consejos proactivos que ayuden a los usuarios a alcanzar sus metas de salud y fitness más eficazmente.

6.3. Monitoreo del Estado de Ánimo y Bienestar Emocional

Propuesta: Incorporar características que permitan a los usuarios registrar y monitorear su estado de ánimo y bienestar emocional, correlacionando estos datos con sus actividades físicas.

Objetivos:

- Explorar la relación entre el ejercicio físico y el bienestar emocional, proporcionando a los usuarios información valiosa sobre cómo su actividad física afecta su estado de ánimo.
- Ofrecer recomendaciones personalizadas para mejorar el bienestar emocional a través de actividades físicas específicas, basadas en los datos recopilados.

Implementación y Consideraciones

Para llevar a cabo estas expansiones, será crucial:

- **Colaborar con Expertos**: Trabajar junto a profesionales del deporte y psicólogos para desarrollar el contenido y las funcionalidades relacionadas con el bienestar emocional.
- **Invertir en Tecnología de IA**: Explorar y adquirir tecnologías avanzadas en inteligencia artificial y aprendizaje automático para desarrollar un asistente virtual eficaz.
- **Pruebas Continuas**: Implementar ciclos de prueba rigurosos para asegurar que las nuevas funcionalidades no solo sean eficaces, sino también seguras y fáciles de usar para todos los usuarios.

Conclusión

Estas áreas de desarrollo no solo prometen mejorar la funcionalidad de la aplicación, sino que también alinean con las tendencias actuales en tecnología y bienestar personal. A través de la implementación de estas propuestas, la aplicación puede evolucionar para ofrecer un servicio más integral y personalizado, posicionándose como una herramienta indispensable en la vida de sus usuarios.

Referencias Bibliográficas

Angular. (s. f.). *What is Angular?*. Recuperado el 9 de noviembre de 2024, de https://v17.angular.io/guide/what-is-angular

Node.js. (s. f.-a). *Introduction to Node.js*. Recuperado el 9 de noviembre de 2024, de https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

Node.js. (s. f.-b). *About Node.js*. Recuperado el 9 de noviembre de 2024, de https://nodejs.org/en/about

Mozilla Developer Network. (s. f.). *Express/Node introduction*. Recuperado el 9 de noviembre de 2024, de https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

IBM. (s. f.). *What is MongoDB?*. Recuperado el 9 de noviembre de 2024, de https://www.ibm.com/topics/mongodb

Amazon Web Services. (s. f.-a). *What is MongoDB?*. Recuperado el 9 de noviembre de 2024, de https://aws.amazon.com/es/documentdb/what-is-mongodb/

Amazon Web Services. (s. f.-b). *What is Scrum?*. Recuperado el 9 de noviembre de 2024, de https://aws.amazon.com/es/what-is/scrum/

Atlassian. (s. f.). *¿Qué es Scrum?*. Recuperado el 9 de noviembre de 2024, de https://www.atlassian.com/es/agile/scrum

Scrum Guides. (2020). *Guía de Scrum 2020*. Recuperado el 9 de noviembre de 2024, de https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf

Scrum Manager. (s. f.). *Documento*. Recuperado el 9 de noviembre de 2024, de https://www.scrummanager.com/files/sm_proyecto.pdf

IBM. (s. f.). *JWT Authentication*. Recuperado el 9 de noviembre de 2024, de https://www.ibm.com/docs/es/order-management?topic=users-jwt-authentication

Auth0. (s. f.). *Learn JSON Web Tokens*. Recuperado el 9 de noviembre de 2024, de https://auth0.com/es/learn/json-web-tokens

BBVA. (s. f.). *JSON Web Tokens (JWT): claves para usarlos de manera segura*. Recuperado el 9 de noviembre de 2024, de https://www.bbva.com/es/innovacion/json-web-tokens-jwt-claves-para-usarlos-de-manera-segura/

Gobierno del Estado de México. (s. f.). *MongoDB*. Recuperado el 9 de noviembre de 2024, de https://dgsei.edomex.gob.mx/sites/dgsei.edomex.gob.mx/files/files/MongoDB.pdf

Universidad Distrital Francisco José de Caldas. (2015). *Documento PDF*. Recuperado el 9 de noviembre de 2024, de https://repository.udistrital.edu.co/bitstream/handle/11349/2742/PovedaGalvisJuanPablo2015.pdf?sequence=1