Facultad de Ciencias

Herramienta web para la provisión de entrenamientos y planes nutricionales personalizados en centros deportivos

Web tool for the provision of customized training and nutritional plans in sports centers.

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Daniel Ahedo García

Director: Alfonso De La Vega Ruiz

Septiembre – 2022

Índice de contenido

Índice de figuras:

Índice de Tablas:

Resumen

A lo largo de los últimos años, ha aumentado la preocupación de la población por su estado de salud, tanto físico como mental. Prueba de ello es que cada vez se ven más centros deportivos o "gimnasios" en nuestras ciudades.

Inicialmente, se conocían los gimnasios como lugares a los que solo iban culturistas o deportistas de alto rendimiento, pero actualmente son un lugar para todo aquel que se quiera mantener en forma. Ya no se va al gimnasio con el único fin de hacer pesas, con el paso del tiempo estos gimnasios han ido incluyendo nuevas áreas de entrenamiento, como por ejemplo natación, clases de yoga, baile, etc. con el objetivo de ofrecer completos programas de fitness. Estos programas incluyen todo tipo de ejercicios con el objetivo de ayudarnos a ser personas más sanas y de mejorar nuestra calidad de vida.

Sin embargo, a pesar de todos estos beneficios y un gran esfuerzo por su parte, muchas personas no logran obtener los resultados esperados. Hay que tener en cuenta algunos factores. Por ejemplo, al realizar cualquier ejercicio, es importante saber si el movimiento realizado puede ser útil para lograr nuestro objetivo o si por el contrario existe la posibilidad de lesionarse. Esto puede ocurrir especialmente cuando se realizan entrenamientos con pesas o entrenamiento de resistencia intensos, por lo que siempre se debe tener cuidado y se debe adaptar el entrenamiento a realizar a nuestras capacidades físicas.

Otro factor muy importante, por el cual algunos clientes no consiguen sus objetivos, es la nutrición. Hoy en día es posible comer comidas saludables en casa con relativa facilidad, pero a la mayoría de las personas les resulta difícil llevar una alimentación equilibrada y que les permita mantener sus niveles de energía durante todo el día o incluso durante una sesión de entrenamiento.

El proyecto llevado a cabo pretende brindar a los clientes de dichos gimnasios una plataforma que les ayude a cumplir sus objetivos mediante un sistema de asesorías deportivas y nutricionales, con el cual el personal cualificado que trabaje en su centro deportivo les haga un seguimiento personalizado.

Este proyecto consiste en el diseño e implementación de dicho sistema mediante una web multiusuario, a la cual acceden tanto los clientes del gimnasio como el propio personal. Los clientes accederán con el fin de obtener acceso a dietas y entrenamientos, bien sean asignados por un especialista, o los genéricos proporcionados por el sistema, además de para llevar un seguimiento semanal de su estado físico. Por otra parte, el personal del gimnasio podrá acceder para asesorar a todos los clientes que les sean asignados. En función del estado físico y el objetivo de cada uno, les asignarán un entrenamiento y una dieta que se adapte a sus necesidades.

Palabras clave: Java, Desarrollo Web, React, Gimnasio.

Abstract

Over the last few years, the population has become increasingly concerned about its state of health, both physical and mental. Proof of this is that we are seeing more and more sports centers or "gyms" in our cities.

Initially, gyms were known as places where only bodybuilders or high-performance athletes went, but nowadays they are a place for anyone who wants to keep in shape. No longer do people go to the gym for the sole purpose of doing weights, over time these gyms have been including new areas of training, such as swimming, yoga classes, dance, etc. with the aim of offering complete fitness programs. These programs include all kinds of exercises with the aim of helping us to become healthier people and to improve our quality of life.

However, despite all these benefits and a great effort on their part, many people fail to achieve the expected results. Some factors must be taken into account. For example, when performing any exercise, it is important to know if the movement performed can be useful to achieve our goal or if on the contrary there is the possibility of getting injured. This can occur especially when performing weight training or intense resistance training, so you should always be careful and adapt the training to be performed to our physical capabilities.

Another very important factor, because of which some clients do not achieve their goals, is nutrition. Nowadays it is possible to eat healthy meals at home with relative ease, but most people find it difficult to eat a balanced diet that allows them to maintain their energy levels throughout the day or even during a training session.

The project aims to provide customers of these gyms with a platform to help them meet their goals through a system of sports and nutritional counseling, with which qualified personnel working in their sports center will provide them with a personalized follow-up.

This project consists of the design and implementation of such a system through a multi-user website, which can be accessed by both the clients of the gym and the staff themselves. The clients will access the system in order to obtain access to diets and workouts, either assigned by a specialist or the generic ones provided by the system, as well as to keep a weekly follow-up of their physical condition. On the other hand, the gym staff will be able to access to advise all the clients assigned to them. Depending on the physical condition and the objective of each one, they will assign them a training and a diet that adapts to their needs.

Keywords: Java, Web Development, React, Gym.

1 Introducción

Mediante la realización de este trabajo se pretende dar soporte a los nuevos servicios de entrenamiento y nutrición ofertados por los gimnasios a sus clientes.

Se ha construido una herramienta web que pretende proveer a distintos centros deportivos de un servicio de nutrición y entrenamiento online personalizado para sus clientes. A través de la web, los clientes del centro deportivo podrán acceder a dietas y/o entrenamientos asignados por un especialista (entrenador y/o nutricionista) en función de sus necesidades.

Antes de continuar y con el fin de ser precisos, se aclara la nomenclatura que se está utilizando, siendo ésta la siguiente:

- Clientes: son los usuarios directos del gimnasio, los que pagan por el uso de las instalaciones y los que contratarán en caso de desearlo los servicios de entrenamiento o nutrición. Serán los usuarios finales de la aplicación.
- Nutricionistas y entrenadores: Forman parte del personal del gimnasio. Estos usuarios, a través de nuestra herramienta crearán y asignarán dietas/entrenamientos a los clientes.
- Usuarios: Hace referencia a cualquiera de los anteriores mencionados. Cualquier persona con acceso a la herramienta.

Dicha plataforma web se divide en tres ramas principales en función del tipo de usuario que interactúe con ella. Por una parte, los clientes podrán conectarse y ver sus entrenamientos y dietas en caso de tenerlos asignados. Por otra, los trabajadores del centro (entrenadores o nutricionistas) podrán ver sus clientes y asignarles distintos planes.

La aplicación se ha desarrollado en base a las siguientes tecnologías principales: Mysql [1] para la base de datos, Java [2] con Spring [3] para el backend y ReactJs [4] para el frontend.

1.1 Motivación

En los centros deportivos ya mencionados anteriormente, así como en la mayoría de los negocios existentes es común que la información se gestione mediante sistemas del tipo Enterprise Resource Planning (ERP). Estos sistemas ayudan a gestionar a los clientes, así como sus recibos, pagos etc. Generalmente, estos sistemas se encargan de la parte administrativa del negocio, pero no suelen estar preparados para incorporar funcionalidades más allá de las relacionadas con la gestión.

Con la elaboración de este TFG se pretende aportar a un negocio (en este caso un centro deportivo) nuevas funcionalidades que otorguen valor a sus usuarios. Por una parte, los propios trabajadores del gimnasio tendrían un fácil acceso a la información de los clientes que tengan asignados. De esta forma, cuando un entrenador se disponga a crear un entrenamiento para un cliente, podrá disponer de toda su información, por ejemplo, edad, estado físico, lesiones, etc. con el fin de adaptarse lo máximo posible al cliente final.

Por otra parte, esta herramienta permitiría asignar de forma fácil y rápida entrenamientos creados previamente a los clientes.

Además, esta herramienta les facilitaría a los clientes el acceso a la información recibida por parte del personal del gimnasio y, en caso de no tener contratados los servicios de entrenamiento o nutrición, dispondrían de entrenamientos y dietas genéricas de las que hacer uso.

Por último, los clientes dispondrían de una comunicación asíncrona con el personal del centro. Es decir, los clientes al realizar revisiones escribirían comentarios, que posteriormente, serían respondidos por el personal del centro cuando los revisen. De esta forma, la información quedaría registrada en las revisiones a modo de histórico, y los usuarios podrían volver y leer esos comentarios siempre y cuando lo consideren necesario.

1.2 Objetivos

El principal objetivo de este trabajo de fin de grado es brindar a distintos centros deportivos una plataforma que se pueda integrar con los sistemas de los que dispongan y aportarles la funcionalidad ya mencionada en los apartados anteriores.

A continuación, se detallan los objetivos generales y específicos del trabajo.

1.2.1 Objetivos generales

- Crear una plataforma web multiusuario para un centro deportivo.

1.2.2 Objetivos específicos

- Implementar un mecanismo de inicio de sesión que permita a los usuarios acceder a la plataforma web.
- Implementar un mecanismo de navegación por pantallas que permita a los clientes acceder a entrenamientos, dietas, …
- Permitir a los clientes introducir datos a través de la web, para poder ser revisados posteriormente por el personal del gimnasio.
- Implementar funcionalidades de creación de dietas y entrenamientos en los perfiles de entrenadores y nutricionistas para ser asignados posteriormente a los clientes.

2 Background de la aplicación

En este apartado se tratan los elementos empleados en la creación de la herramienta web de este TFG. Se listarán los distintos elementos, tecnologías, herramientas, así como una pequeña descripción de cada uno, que nos aporte un pequeño conocimiento para su comprensión en apartados posteriores.

2.1 Tecnologías

Primeramente, se listan las tecnologías utilizadas tanto para las interfaces de usuario, como para las bases de datos, o el backend de la aplicación.

2.1.1 Java / Spring

Java [2] es un lenguaje de programación de alto nivel que permite a los desarrolladores crear aplicaciones robustas y confiables. Tiene muchas características que lo hacen ideal para el desarrollo web, como, por ejemplo, la concurrencia, orientación a objetos, anotaciones, reflexión, servicios web etc. En este caso, se ha utilizado el framework Spring [3] para llevar a cabo el desarrollo backend. Dentro del framework de Spring [3], está Spring Boot que proporciona una forma sencilla de desarrollar aplicaciones web y las hace fáciles de mantener con el tiempo y Spring security, que nos provee de atajos a la hora de desarrollar mecanismos de autorización y autenticación de usuarios.

2.1.2 Maven

Maven [5] es una herramienta de software para administrar y construir proyectos Java [2]. Su modelo de construcción se basa principalmente en un fichero XML llamado POM. Maven [5] utiliza este POM para describir el proyecto, sus dependencias, librerías y componentes externos, así como el orden de construcción de los elementos.

2.1.3 JavaScript / React

Para el frontend de la aplicación se ha decidido utilizar JavaScript, en este caso integrado con el framework React [4]. JavaScript es un lenguaje de programación utilizado del lado del cliente e interpretado por parte del navegador web. React [4], por otra parte, es una biblioteca de JavaScript utilizada en el desarrollo de interfaces de usuario. En este caso se ha utilizado para crear nuestra interfaz de usuario, siguiendo una estructura web SPA (Single Page Application).

La implementación se ha desarrollado mediante ficheros *.jsx. Este tipo de ficheros podría parecer HTML, pero no son más que una extensión del lenguaje JavaScript que produce componentes de React [4] que se renderizarán en el DOM (Document Object Model). Estos componentes son fragmentos de código completamente independientes, que tienen una funcionalidad concreta y se integran entre sí para dar forma a la página web. Cuando se habla del DOM, se hace referencia a la estructura en forma de árbol de los elementos HTML. En este caso nuestros componentes pasaran a formar parte de este árbol de elementos.

2.1.4 NPM

NPM [6] se puede considerar como una tecnología análoga a Maven [5] en Java [2] para el código JavaScript. Es un gestor de paquetes que nos permite instalar y gestionar versiones de paquetes y librerías de JavaScript.

2.1.5 MySQL

MySQL [1] es un sistema de gestión de bases de datos relacionales. Se ha decidido utilizar este en lugar de otros debido a que tiene muy buena integración con Spring [3] y es de código abierto.

2.1.6 Git

Git [7] es lo que se conoce como un sistema de control de versiones. Permite crear un repositorio en el que ir almacenado las distintas versiones de nuestro código, crear distintas ramas para trabajar, además de otras ventajas para desarrollos multi-usuario que no se han utilizado al tratarse de un proyecto individual.

2.2 Herramientas:

Este punto se centra en algunas herramientas utilizadas, bien sea para facilitar el propio desarrollo, para realizar pruebas, o para el control de versiones, entre otras.

2.2.1 Gitkraken

Gitkraken [8] no es más que un cliente para git [7]. Nos proporciona una interfaz gráfica para manejar el estado y el contenido del repositorio. En vez de utilizar comandos a través de un terminal, nos permite visualizar el progreso gráficamente permitiéndonos ahorrar bastante en tiempo y ganar comodidad

2.2.2 DBeaver

DBeaver [9] es una herramienta que funciona como cliente de bases de datos genérico compatible con multitud de gestores. En este caso se ha utilizado para crear y conectarse a la base de datos MySQL [1] de la plataforma web. Esta herramienta nos facilita algunas tareas como:

- La ejecución de sentencias SQL / ejecución de scripts
- Exportar datos
- Búsqueda de objetos en la BBDD
- Crear y editar la BBDD de forma sencilla e intuitiva.

2.2.3 Postman

Postman [10] es un software gratuito que nos permite realizar peticiones contra un API REST [11]. En este caso se ha utilizado tanto a la hora de la implementación como a la hora de realizar las pruebas del correcto funcionamiento de todos los endpoints de nuestra API REST[11].

2.2.4 Editores

Otro elemento fundamental son los editores. En este caso concreto, y con el fin de disponer del código de frontend y de backend abierto y lanzado a la par, se han utilizado los siguientes editores.

2.2.4.1 Spring Tool Suite (STS)

Spring Tool Suite [12] es un entorno de trabajo creado a partir del editor eclipse. Es un entorno denominado como *ready to use*, es decir, desde el momento en que se instala, se pueden comenzar a implementar, depurar, ejecutar y desplegar las aplicaciones Spring [3] que se desarrollen.

Además, este editor tiene embebido un servidor (en este caso tomcat) que nos permite arrancar nuestra aplicación (en nuestro caso el backend) para poder realizar pruebas sobre ella.

2.2.4.2 Visual Studio Code (VSC)

Por otra parte, se ha utilizado Visual Studio Code (VSC) [13] como editor para el frontend. Visual Studio Code [13] es un editor de código fuente desarrollado por Microsoft, libre y multiplataforma. Además de poseer una muy buena integración con git [7], este editor tiene infinidad de extensiones que nos facilitarán la vida a la hora de realizar el desarrollo, lo cual contribuye a mejorar la calidad de nuestro código.

Además de esto, el editor dispone de una consola integrada, mediante la cual, y con el uso del comando de npm “npm start”, se puede iniciar un servidor que sirva nuestra interfaz frontend.

3 Metodología seguida en el desarrollo del proyecto

El desarrollo de este proyecto se ha llevado a cabo siguiendo una metodología iterativa inspirada en el sistema ágil Scrum [14] y adaptada para el desarrollo por parte de una sola persona, realizando los pasos que se comentan en esta sección.

3.1 Diseño de la aplicación a alto nivel

En la primera fase del proyecto, lo que se realizó fue un primer diseño de la aplicación. Se planteó el "¿Qué queremos hacer?" y "¿Cómo lo vamos a llevar a cabo?". Se realizaron varias reuniones en las que se definieron los principales puntos del proyecto, como por ejemplo el uso de tres roles distintos, clientes, nutricionistas y entrenadores, además de seleccionar las tecnologías a utilizar (Java [2], React [4], MySQL [1]) y la metodología a seguir.

3.2 Creación de un primer esqueleto de aplicación.

Antes de empezar a trabajar de forma iterativa siguiendo lo definido por la metodología agile, se comenzó por implementar un "Walking Skeleton". Un Walking Skeleton es una estructura básica de la aplicación de principio a fin, es decir, desde el frontend hasta la base de datos, pero aún incompleta como para poder ser útil al usuario. Este primer esqueleto no dispone siquiera de la estructura final de la aplicación, pero nos sirve como primera versión sobre la que poder comenzar a iterar e ir realizando tareas de forma semanal.

A nivel de backend, simplemente se creó una estructura de tablas básica (Ilustración 1) con un único perfil de cliente, ejercicios, entrenamientos y dietas.

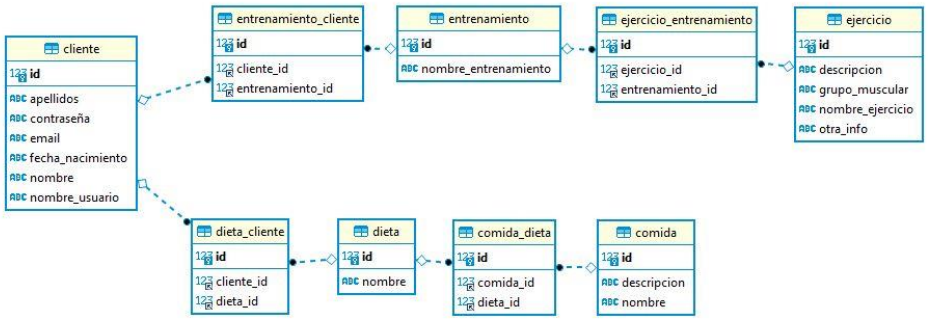

Ilustración 1: Estructura de la BBDD en el Walking Skeleton

Por otra parte, a nivel frontend, se creó una vista principal (Ilustración 2) con un conjunto de botones, que redirigen a vistas de ejercicios, entrenamientos, alimentación y seguimiento.

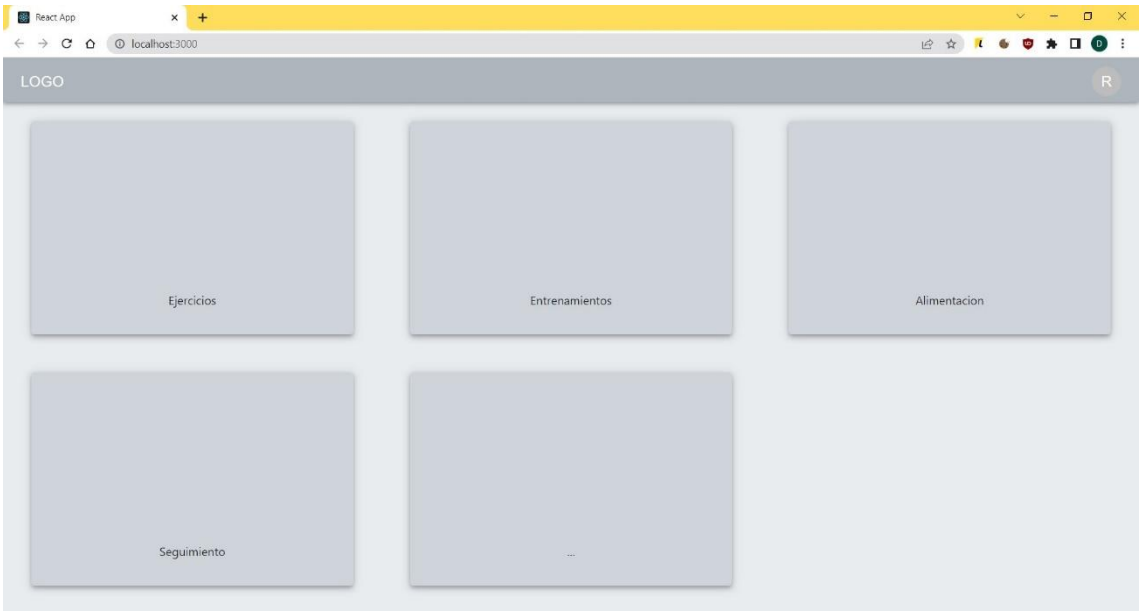

Ilustración 2: Estructura del frontend en el Walking Skeleton

3.3 Desarrollo del proyecto (Sprints)

Una vez definida esta primera versión se comenzó a aplicar la metodología agile a través del marco de Scrum [14]. Cabe mencionar que no se ha empleado Scrum al uso, dado que el trabajo se ha realizado de forma autónoma en lugar de junto a un equipo. Por tanto, algunas ceremonias no se han llevado a cabo, como por ejemplo el daily meeting. En cambio, sí se ha trabajado de forma iterativa, realizando sprints de una semana y teniendo reuniones con el director del TFG (quien ha hecho el papel de Product Owner) de forma semanal. En estas reuniones se comprobaban el avance semanal, posibles cambios o mejoras a realizar en dichas implementaciones, y se definían nuevas tareas de cara al siguiente sprint. En este apartado se explicarán brevemente las características implementadas a lo largo de cada uno de los sprints.

Destacar que se ha realizado control de versiones mediante un repositorio alojado en github [15]. Dicho repositorio contiene una rama master con un MVP (Producto Mínimo Viable) de la aplicación y una rama develop en la que se han realizado todos los desarrollos.

Sprint 01: A partir del primer esqueleto se crearon dos vistas: una vista principal, y una vista de inicio de sesión previas a los distintos menús. Para conseguir que este login fuera funcional, a nivel backend se creó una tabla de usuarios, que contiene los nombres de usuario y contraseñas de los usuarios de nuestra aplicación. Esta tabla está directamente relacionada tanto con la tabla clientes ya existente, como con las tablas entrenadores y nutricionistas que se crearon en este sprint.

Sprint 02: Una vez se tuvo una versión de login, se comenzó a desarrollar un menú de cliente, de tal forma que cuando el cliente se inicie sesión vaya directamente a su propio menú. De esta forma se dejó de utilizar el menú común definido en la primera versión de esqueleto.

En la raíz del proyecto, se definió una redirección en base a la ruta de la URL de acceso y el tipo de usuario que haya iniciado sesión. De esta forma, quedaron definidas las siguientes rutas:

- /: nos muestra la página principal (landing page).
- /login: nos muestra el login.
- /menuCliente: nos muestra el menú del cliente. A este apartado sólo puede acceder un usuario logueado con rol cliente.
- /menuEntrenador: nos muestra el menú del entrenador. A este apartado sólo puede acceder un usuario logueado con rol entrenador.
- /menuNutricionista: nos muestra el menú del nutricionista. A este apartado sólo puede acceder un usuario logueado con rol nutricionista.

Además de esto y para cerrar el círculo de la sesión, se implementó un botón de cierre de sesión o logout.

Sprint 03: Una vez el cliente ha iniciado sesión, se debe adquirir su información. Por tanto, la próxima característica que se implementó a nivel de backend es un endpoint que nos permite obtener la información de dicho cliente a través de su ID. Cuando se utiliza la palabra endpoint o punto final, se hace referencia a las URLs del backend que reciben y responden a una petición.

Aprovechando la necesidad de crear este endpoint para el cliente y dada la similitud entre los distintos tipos de usuario, también se crearon endpoints similares (findByID) para el entrenador y nutricionista. Adicionalmente, se creó un pequeño script SQL mediante el cual se insertaron datos en la base de datos que se utilizaron para comprobar que los endpoints devolvían la información de manera correcta (desde Postman [10]) y el frontend lo mostraba de igual manera.

Sprint 04: En este sprint se comenzó el desarrollo de dos características a nivel frontend, una de nutricionista y otra de cliente. Por una parte, a nivel de cliente, se comenzó el desarrollo de una vista de perfil en la que se muestran sus datos, así como el Test PAR-Q [16] (Physical Activity Readiness). Por otra parte, a nivel nutricionista se modificó la redirección para su rol y se comenzó el desarrollo de su menú principal.

Sprint 05: Al ser las características del sprint 04 de gran magnitud, la implementación llevó más de un sprint, terminando su implementación al final de este.

Sprint 06: Como siguiente paso se decidió continuar trabajando sobre el menú de nutricionista. Por la parte del backend se reestructuró el modelo de dietas, creando por una parte dietas semanales y por otra parte dietas diarias, ambas relacionadas entre sí y con el cliente. Para acceder a esta nueva estructura, se crearon varios endpoint, uno que nos permite obtener todas las dietas diarias, y otros dos con los que crear y actualizar las dietas diarias y semanales.

Como en anteriores ocasiones al crear nuevos endpoints, se añadieron datos al script de pruebas, para comprobar desde Postman [10] que estos funcionaban correctamente, y posteriormente se muestre de dicha información en las respectivas vistas.

Por la parte frontend, a nivel de cliente se comenzó el desarrollo del apartado de visualización de dietas, mientras que a nivel de nutricionista se desarrolló la parte de

creación de dietas diarias y semanales y una primera versión de pantalla de visualización de dietas.

Sprint 07: Hasta este punto se venía arrastrando un error de CORS (control de acceso http) y en este sprint se dedicó tiempo a encontrarlo y solucionarlo. Este error hacía que se permitiesen las peticiones desde Postman [10], pero cuando se realizaban desde la propia web generaba error.

Solucionado este error que existía a nivel backend, se introdujeron algunos pequeños cambios en el modelo de los ejercicios, para completar su estructura y se añadió el endpoint que permite obtener un listado de todos los ejercicios.

En la parte frontend se llevaron a cabo varias implementaciones. Por la parte del cliente se creó una nueva vista de visualización de ejercicios, que invoca al endpoint que se creó en este mismo sprint y lista en forma de tabla todos los ejercicios disponibles. Por la parte del nutricionista se finalizó la implementación de las vistas de creación de dieta diaria y semanal.

Sprint 08: Llegado este punto, al haber creado varios componentes a nivel frontend, se decidió reorganizar el código en directorios para tenerlo agrupado y más organizado. De esta manera el código quedó agrupado en tres directorios principales, uno para cada tipo de usuario, y dentro de cada directorio otro en base a cada vista dentro de cada menú.

En este sprint la mayoría de las implementaciones han correspondido al apartado del cliente. Por una parte, se implementó las revisiones tanto a nivel backend como frontend y, además se creó un documento JSON que contiene una lista de dietas predeterminadas, que se muestran en el menú de dietas del cliente. Por la parte del nutricionista se continuó avanzando con la parte de visualización y edición de dietas.

Sprint 09: Continuando con la parte de ver y editar dietas en el nutricionista, se añadió un pequeño popup o mensaje emergente que aparece cuando se utiliza algún botón de guardar. Este pequeño popup, puede aparecer en color verde o rojo, e indica si la petición de guardado se ha realizado correctamente o no, respectivamente. Además de esto, se comenzó a implementar en el menú de nutricionista la última de sus vistas, correspondiente a los clientes asignados.

Por otra parte, se desarrolló el menú del entrenador, así como con una primera versión de la vista de crear entrenamientos diarios y semanales.

Sprint 10: Al haberse comenzado el desarrollo del apartado de entrenador en el sprint anterior, el siguiente paso a realizar se corresponde con la gestión de entrenamientos. A nivel backend se implementó todo lo necesario relativo a los entrenamientos diarios y se comenzó con los semanales, de forma que se pudieran utilizar desde el frontend. Además, en este sprint, se tomó la decisión de refactorizar el modelo de cliente, extrayendo a una tabla todos los parámetros de salud, como, por ejemplo, el test PAR-Q [16], las lesiones y las alergias.

Sprint 11: Este sprint se enfocó principalmente en avanzar con la parte del entrenador. Por una parte, se continuó avanzando con la creación de entrenamientos diarios del anterior sprint, y por otra se crearon los endpoints de entrenamientos semanales. Al igual

que en las anteriores ocasiones, se actualizó el script con los datos de prueba. En este caso se añadió información sobre los entrenamientos.

Sprint 12: hasta el momento, se habían utilizado en la web unas imágenes de pruebas, pero en este sprint se cambiaron por imágenes ([17]–[19]) con licencia CC0 [20], es decir, de dominio público. Además, se comprobó que los gifs de los ejercicios utilizados también son de uso libre.

En este punto se decidió cambiar la forma en la que se mostraba la información en los distintos menús. Hasta este punto se cargaba una página en blanco hasta que el usuario seleccionase un submenú, del cual se cargaba su información en la pantalla. En este sprint se eliminaron todas esas páginas en blanco, mostrando para el cliente por defecto la página de su perfil y para entrenadores y nutricionistas mostrando por defecto la página que contiene sus clientes asignados.

Como los datos tardan en llegar un determinado tiempo desde la base de datos y ahora se pintan según se accede al menú, se implementó el uso de flags de carga en los tres componentes de menú. De esta forma, hasta que no se obtienen todos los datos de base de datos, no se muestran en la propia pantalla, sino que se muestra un icono de carga.

A nivel de entrenador, en este sprint se creó la vista y funcionalidad de ver entrenamientos diarios y semanales, así como la de creación de entrenamientos semanales y la vista de clientes asignados.

A nivel de nutricionista, se implementó un procesado de datos, que divide los clientes asignados en tres categorías, *nuevos*, *revisados* y *sin revisar*. Para cada uno de los clientes asignados que se muestra en cada categoría se puede visualizar sus revisiones.

Sprint 13: Para finalizar con la implementación del cliente, se creó un JSON con entrenamientos por defecto, al igual que se realizó con las dietas. Estos entrenamientos genéricos los podrá ver bajo la pantalla de entrenamientos. Por la parte del entrenador, en este sprint se implementó la funcionalidad correspondiente a la actualización de entrenamientos diarios y semanales

Sprint 14: finalizando las implementaciones de entrenador y nutricionista, se implementó a nivel de clientes asignados, la asignación a los clientes tanto de dietas como de entrenamientos.

Sprint 15: En un último sprint, se llevó a cabo la autorización mediante JWT [21]. Durante todo el desarrollo solo se autorizaba a los usuarios mediante el login, es decir, se le daba acceso al sistema, pero todos los usuarios podían acceder a toda la información. Con esta última implementación, solo tendrán acceso a determinados recursos dependiendo del rol que tengan asignado, no pudiendo acceder a endpoints que no estén habilitados para su rol.

4 Estructura de la aplicación

En este apartado se describe el proceso que se ha seguido a la hora de estructurar la aplicación, los casos de uso de la aplicación y la captura de los requisitos tanto funcionales como no funcionales.

4.1 Explicación de la aplicación

Como se menciona en apartados anteriores, ésta es una aplicación multiusuario, por tanto, para explicar completamente su funcionalidad, se deberá tratar por separado a cada tipo de usuario.

Punto de vista del usuario: Independientemente del tipo de usuario que se conecte a la web, todos siguen un procedimiento de inicio de sesión común. Los usuarios accederán a través de una URL, que por defecto cargará la página de inicio. En la parte superior derecha hay un botón de inicio de sesión, que redirige a una pantalla de login. Una vez en esta pantalla y haciendo uso de unas credenciales que le hayan sido asignadas por el personal del gimnasio, podrá iniciar sesión. Si el inicio de sesión es satisfactorio, se accederá a un menú personalizado en función del rol del usuario (cliente, entrenador o nutricionista).

Punto de vista del cliente: En el caso del cliente, una vez iniciada sesión, accederá a su menú de cliente, donde, a la izquierda, dispone de un conjunto de botones que le permiten navegar por los submenús, y a la derecha se mostrará la información en función de lo seleccionado a través de dichos botones.

El primer submenú que aparecerá seleccionado por defecto será su perfil, en él podrá tanto ver como editar su información personal. Esta información hace referencia por una parte a su información propia (nombre, apellidos, fecha de nacimiento…) y por otra a su estado físico y de salud. Con el fin de aportar la máxima información, parte de estas preguntas son un test PAR-Q [16]. Este es un test desarrollado por la Sociedad Canadiense de Fisiología del Ejercicio. Es reconocido a nivel mundial y tiene como fin determinar de forma fácil y rápida los posibles riesgos para la salud del usuario antes de realizar cualquier tipo de ejercicio.

Los siguientes submenús corresponden a las dietas y los entrenamientos. Aquí el usuario podrá acceder a una lista de dietas genéricas y entrenamientos genéricos, dependiendo del submenú seleccionado. Además, aquí se le indica si tiene asignada alguna dieta/entrenamiento, y en caso de tener una de las dos o ambas cosas asignadas, podrá visualizar su contenido.

El siguiente submenú no es más que una lista de ejercicios. Aquí se muestra un listado de ejercicios en forma de tabla, con la que el cliente podrá interactuar. Estos ejercicios tienen un nombre, por el cual podrán ser filtrados, además de otras propiedades como el grupo muscular que ejercita o un gif describiendo como se deberá realizar el ejercicio.

E el último submenú al que podrán acceder los clientes es el de revisiones, donde cada 7 días el usuario podrá imputar datos de su estado físico. Estos datos son principalmente

peso y medidas corporales además de un comentario en el que podrá escribirle al personal que tenga asignado sus sensaciones o necesidades. Personal que a su vez podrá darle feedback.

Punto de vista del nutricionista: Una vez iniciada sesión en el sistema, este detectará que es un nutricionista y le mostrará su propio menú. Este menú tiene la misma estructura que en el caso de los clientes, apareciendo en la parte izquierda una lista de botones para navegar por los submenús y a la derecha el contenido del que esté seleccionado en ese momento.

Por defecto aparece seleccionado el submenú de los clientes. En este apartado, el nutricionista podrá ver los clientes que tiene asignados a través de tres pestañas. Por un lado, los clientes nuevos, por otro los que ya han sido revisados, y finalmente los que necesitan ser revisados. Dependiendo del tipo de cliente, el nutricionista podrá asignarles una dieta, o cambiarles la que ya tengan asignada, además de realizar comentarios cuando los clientes tengan una revisión.

El siguiente submenú, les permitirá crear dietas. Podrán crear dietas tanto diarias como semanales. Para crear dietas diarias simplemente deberán asignarles un nombre y rellenar el número de comidas deseadas. En cambio, para crear una dieta semanal, deberá seleccionarse una dieta diaria para cada día de la semana. En caso de querer dejar algún día de la semana en blanco, el nutricionista puede generar una dieta diaria llamada “Dia libre” para poder asignar a dichos días.

Finalmente, y como es lógico, el ultimo submenú da la posibilidad a los nutricionistas de poder visualizar tanto las dietas diarias como las semanales, además de poder editar su contenido.

Punto de vista del entrenador: El punto de vista del entrenador es muy similar al del nutricionista. Al igual que en los anteriores casos, al iniciar sesión se le redirige a su propio menú de entrenador.

Una vez aquí, se muestra por defecto la vista con los clientes que tenga asignados el entrenador logueado. Estos clientes, como en el caso anterior, se clasifican en nuevos, revisados y sin revisar, y se les podrán asignar o modificar los entrenamientos, además de realizar comentarios.

En el siguiente submenú, el entrenador podrá crear entrenamientos diarios y semanales. En este caso, para los entrenamientos diarios, el entrenador seleccionará tantos ejercicios como desee añadir de un listado proporcionado, añadiendo a cada ejercicio un numero de series y de repeticiones por sesión. Cuando el entrenador comienza un entrenamiento vacío, le da un nombre y comienza a añadir ejercicios. Por ejemplo, se podría seleccionar un ejercicio de bíceps, del que se realizarían 3 series; posteriormente un ejercicio de piernas con sus correspondientes parámetros, y así sucesivamente hasta completar el entrenamiento. Para crear un entrenamiento semanal, solo habrán de asignar un entrenamiento diario a cada día de la semana. Al igual que en el caso del nutricionista, en caso de desear imputar un día de descanso, se podría generar un entrenamiento diario llamado “día de descanso” en el que no se añadirá ningún ejercicio.

Además, en el último submenú, podrán ver y editar los entrenamientos creados.

De forma común a todos los usuarios, cuando hayan terminado sus tareas en la aplicación, podrán cerrar sesión en la esquina superior derecha, en el mismo lugar en el que realizaron el inicio de sesión. En caso de que el usuario cierre el navegador sin cerrar sesión, esta caduca tras un determinado tiempo desde el inicio.

4.2 Casos de uso

Con la definición de los casos de uso lo que se pretende es describir las distintas interacciones que pueden existir entre cada tipo de usuario y el sistema. En este caso se muestra mediante una representación gráfica para cada uno de los tres tipos de usuario. En la siguiente imagen (Ilustración 3) está dicha representación.

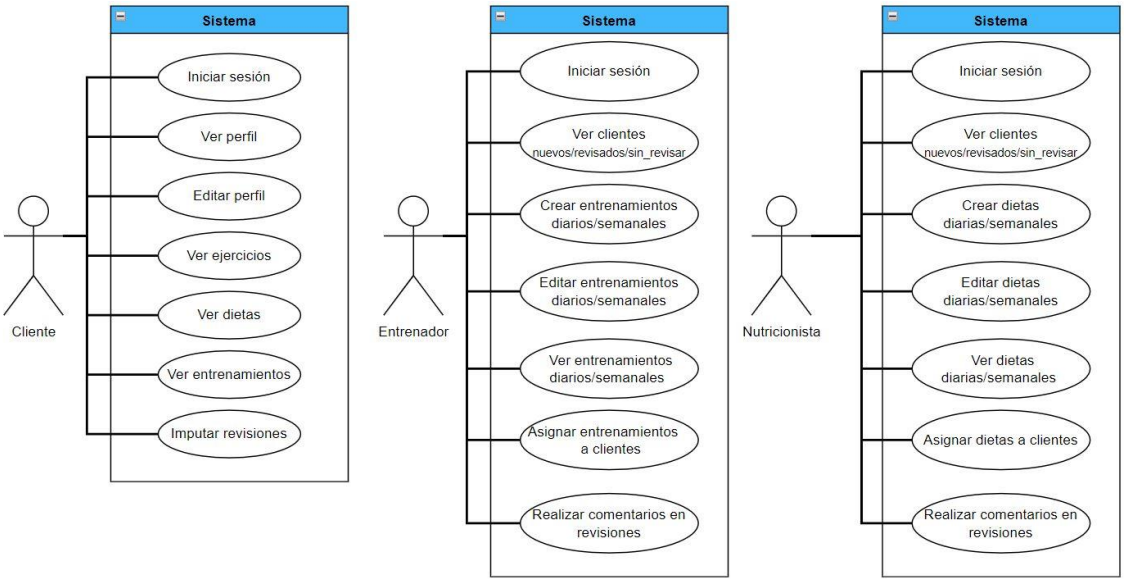

Ilustración 3: Casos de uso

Por simplicidad, en la Ilustración 3 se ha decidido representar solamente los tres tipos de usuario, pero existen relaciones a un nivel superior. Por una parte, estaría el agente "usuario", del que heredan los 3 de la imagen, los cuales tienen una funcionalidad común de inicio de sesión. Y, por otra parte, existe el agente "personal", el cual agrupa también la funcionalidad común entre los tipos de usuario entrenador y nutricionista.

4.3 Requisitos funcionales

Cuando se habla de los requisitos funcionales de una aplicación, se hace referencia a los servicios que ofrecerá y a la forma en que reaccionará en distintas situaciones. En la siguiente tabla se detallan los requisitos funcionales de la aplicación:

Identificador	Requisito	Descripción
RF001	Acceso de usuario	Todo usuario registrado podrá acceder a la plataforma haciendo uso de unas credenciales de inicio de sesión.
RF002	Edición de perfil	Los clientes podrán visualizar sus datos y editar su perfil

RF003	Visualización de información genérica	Todos los clientes podrán visualizar los entrenamientos y dietas genéricos a través de su menú.
RF004	Visualización de información especifica	Los clientes que tengan asignado un entrenador o un nutricionista podrán ver la información que se les asigne.
RF005	Creación de revisiones	Los clientes podrán crear revisiones semanalmente
RF006	Visualización de clientes asignados	El personal del gimnasio podrá ver los clientes que le han sido asignados.
RF007	Creación de dietas	Los nutricionistas podrán tanto crear como editar dietas diarias y semanales
RF008	Visualización de dietas	Los nutricionistas podrán ver todas las dietas
RF009	Creación de entrenamientos	Los entrenadores podrán tanto crear como editar entrenamientos diarios y semanales
RF010	Visualización de entrenamientos	Los entrenadores podrán ver todos los entrenamientos

Tabla 1: Requisitos funcionales

4.4 Requisitos no funcionales

Los requisitos no funcionales de una aplicación hacen referencia a aquellos que no tratan las funciones del sistema sino a sus propiedades.

En la siguiente tabla se detallan los requisitos no funcionales de la aplicación:

Identificador	**Requisito**	**Descripción**
RNF001	Seguridad	La aplicación no presentará vulnerabilidades que afecten a su seguridad o a la integración de los datos de la misma.
RNF002	Visualización	La aplicación debe ser correctamente visible en cualquier dispositivo de escritorio y navegador.
RNF003	Expiración de sesión	Tras un tiempo determinado tras el inicio de sesión de cualquier usuario, este caducará y será necesario que inicie sesión de nuevo.

Tabla 2: Requisitos no funcionales

5 Arquitectura de la aplicación

Una vez se ha definido cómo ha de funcionar la aplicación mediante los casos de uso y la captura de requisitos, se puede proceder al diseño del sistema.

5.1 Diseño arquitectónico

En este apartado se detalla el diseño arquitectónico de la aplicación, es decir, los distintos componentes que la conforman, así como las comunicaciones entre ellos.

La aplicación se estructura en tres componentes básicos, el frontend, el backend y la base de datos. Sigue el esquema arquitectónico representado en la Ilustración 4 además de una arquitectura de capas que se detalla en el apartado 5.2.

El frontend es el encargado de proporcionar la interfaz de usuario. Está implementado en React [4]. Durante el desarrollo del proyecto, y dado el carácter prototípico de esta aplicación web, este frontend, se arrancaba mediante un servidor en NodeJS que se alojaba en la propia máquina (localhost) haciendo uso del puerto 3000.

Además de servir como interfaz de usuario, el frontend interactúa con la API REST [11] del backend, enviando solicitudes HTTP de tipo GET y POST.

El backend, en cambio, es completamente invisible a ojos del usuario. Es una capa intermedia que contiene la lógica de negocio de la aplicación y actúa como intermediaria entre el frontend y la base de datos. Está construido en el lenguaje Java [2], utilizando el framework Spring [3] e implementado en forma de API REST [11]. Este componente se arranca en un servidor tomcat también de forma local (Un servidor que ofrece embebido el editor de Spring [12] introducido en la sección 2.2.4.1) exponiéndolo a través del puerto 8080. Las funciones principales del backend, son: recibir las peticiones del frontend, aplicar la lógica necesaria en función del tipo de petición, solicitar información a base de datos en caso de ser necesario, generar y devolver respuestas al frontend.

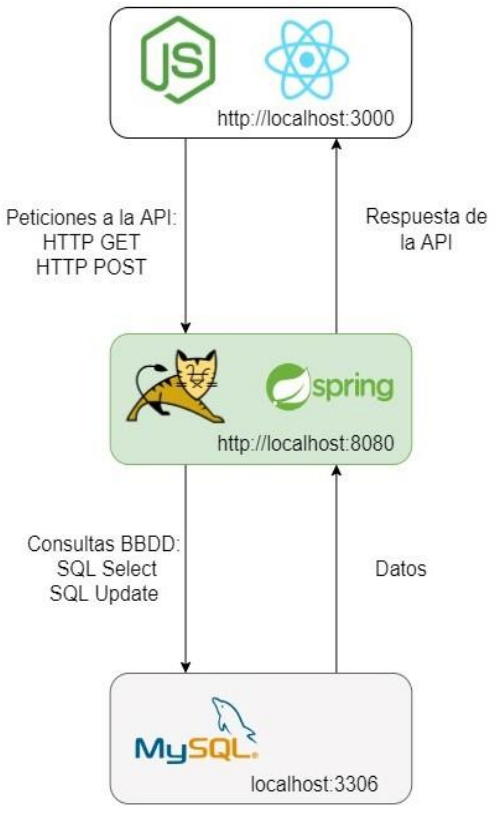

Ilustración 4: Diseño arquitectónico y tecnologías utilizadas

La base de datos es el último eslabón de este sistema, pero no por ello menos importante, ya que es la encargada de persistir toda la información del mismo. Recibe las consultas del backend, y devuelve la información solicitada. También se da el caso que las solicitudes sean de modificación de la información almacenada. El gestor de base de datos utilizado es MySQL [1], y se ha alojado localmente en nuestra máquina, al igual que el resto de los elementos, pero en este caso se expone en el puerto 3306.

Como se ha comentado, los tres elementos del sistema se han alojado en la misma máquina durante las pruebas, pero esto no tiene por qué ser así. Los 3 elementos son completamente independientes y se podrían alojar en tres máquinas diferentes (bien sean ordenadores, servidores o servidores en la nube), siempre y cuando se configure correctamente la comunicación entre sí.

5.2 Arquitectura de la aplicación

La arquitectura de la aplicación es el diseño de más alto nivel de la estructura de un sistema. Hace referencia a la forma en que se estructuran los datos y la estructura de los componentes que se requieren para construir el sistema. Se tiene en cuenta la estructura del sistema, los componentes que lo construyen y las relaciones entre ellos.

Existen muchos tipos de patrones de arquitectura, patrón de capas, patrón cliente-servidor, patrón maestro-esclavo, patrón de filtro de tubería…

En este caso, el desarrollo de esta aplicación se basa principalmente en una implementación por capas, más concretamente una arquitectura en tres capas: capa de presentación, capa de negocio y capa de datos, como detalla la Ilustración 5.

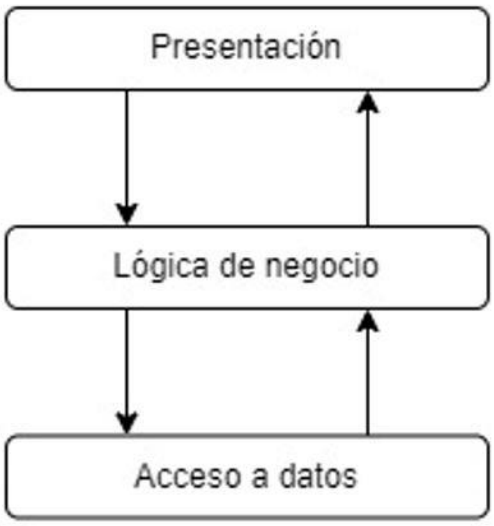

Ilustración 5: Arquitectura en 3 capas

El objetivo principal de este tipo de arquitectura es la separación de cada una de las partes que componen el sistema software. Como indica el esquema, cada una de las capas solo podrá interactuar con capas vecinas. De esta forma, debería ser más sencillo poder realizar cualquier cambio en la aplicación, dado que este solo afectará a la capa involucrada y a lo sumo, a las vecinas.

A continuación, se explican cada una de las capas, comenzando desde la interacción del usuario con el sistema hasta el almacenaje y solicitud de los datos.

5.2.1 Capa de presentación

La capa de presentación es la que ve el usuario y mediante la cual interactúa con el sistema. Generalmente consiste en una interfaz gráfica, que es la encargada de comunicar la información al usuario y capturar su información, de forma amigable, entendible y procurando ser fácil de utilizar. Esta capa se comunicará únicamente con la inmediatamente inferior, es decir, con la capa de negocio.

En este proyecto se ha optado por separar completamente la interfaz gráfica (frontend) de la capa de negocio/acceso a datos (backend). El frontend se ha construido utilizando JavaScript, más concretamente utilizando el framework React [4].

Para diseñar el frontend se ha optado por utilizar la estructura de SPA (Single Page Application). Estas SPA son un tipo de página web que ejecuta todo su contenido en una única página. Funcionan cargando todo el contenido HTML, CSS y JavaScript al abrir la web, por lo que la carga entre vistas es extremadamente rápida. Al cambiar de una vista a otra, solo se carga el contenido nuevo, y esto se hace de forma dinámica, evitándonos cargar la página completa de nuevo. Todo esto mejora la navegación y los tiempos de respuesta, lo que a su vez conlleva una mejor experiencia de usuario.

Uno de los puntos más importantes en la creación de la interfaz de usuario es la organización en base a los tres roles principales. Por ello, se parte de una pantalla de inicio y un login común, pero después cada usuario tendrá visibilidad solo de su propio menú.

5.2.2 Capa de negocio

Esta es la capa que contiene la funcionalidad principal de la aplicación. Es la encargada de recibir y responder a las solicitudes recibidas de la capa de presentación, obteniendo los datos necesarios para procesar dichas solicitudes. Para ello, la capa de negocio se comunicará con la capa de acceso a datos y adquirirá, insertará o modificará información de la base de datos en función de la solicitud recibida. Esta capa está integrada en el backend de la aplicación, construido utilizando Java [2] con Spring [3]. En el siguiente apartado se explicarán algunos de los detalles más importantes de su implementación.

5.2.3 Capa de acceso a datos

Esta es la capa en donde residen los datos y la encargada de gestionar el acceso a los mismos. Está formada por una base de datos MySQL [1], así como por parte del backend.

6 Implementación

Retomando algunos de los puntos mencionados en el apartado anterior, se procede a detallar su implementación haciendo hincapié en los aspectos más importantes de cada capa.

6.1 Capa de presentación

Como se mencionó, esta capa es la que implementa la interfaz de usuario, en nuestro caso, el frontend utilizando React [4].

6.1.1 Implementación del frontend

React [4], como se menciona en los primeros apartados de este trabajo, no es más que una biblioteca de JavaScript utilizada para crear interfaces de usuario mediante la especificación y configuración de componentes. Para crear estos componentes se han utilizado varias librerías, como son Material-UI [22] o Axios [23] entre otras. Se ha optado por su uso debido al fácil aprendizaje y al conocimiento previo existente. El comienzo en el desarrollo es bastante sencillo, solo es necesario tener instalado en nuestra máquina las versiones de Node >= 14.0.0 y npm >= 5.6. Una vez con ellas instaladas, se procede a crear un primer esqueleto de aplicación con el comando: "*npx create-react-app frontend"*. Este comando nos crea una primera versión de aplicación bajo un nuevo directorio con el nombre indicado, que en este caso es "frontend".

Para arrancar la aplicación, se accede al nuevo directorio, y utilizando el comando "npm start", se arrancará un servidor de pruebas en local que alojará nuestra aplicación. Esta estará por defecto bajo la dirección localhost:3000. Una vez tenemos el entorno listo y una primera versión arrancada, puede comenzarse a desarrollar el esqueleto frontend.

La parte frontend de la aplicación se ha construido utilizando la librería Material-UI [22]. Material-UI [22] es una biblioteca de código abierto que implementa componentes y herramientas de diseño de Material Design de Google para hacer que las aplicaciones web sean más rápidas y vistosas con poco esfuerzo.

A la hora de estructurar el código, se ha optado por ordenarlo en base a directorios siguiendo la estructura de tres roles planteada. De esta forma se dispone de una pantalla principal y un login comunes, y después existen los tres tipos de menús. En la captura mostrada en la Ilustración 6 se muestra esta estructura de directorios.

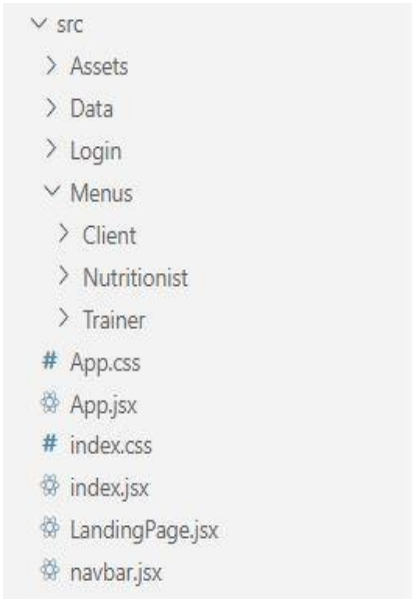

Ilustración 6: Estructura código frontend

Al igual que en la mayoría de las aplicaciones web, partimos de un fichero index.jsx. En nuestro caso, este fichero es el encargado de renderizar el componente App.jsx.

Por tanto, este componente App.jsx (Ilustración 7) será el primer componente de nuestra aplicación. En este componente, se definen las principales URLs del proyecto, así como, en caso de recibir cada una de ellas, a qué componente se ha de redireccionar.

```
9   function App() {
10    return (
11      <BrowserRouter>
12        <Routes>
13          <Route path="/" element={<LandingPage />}></Route>
14          <Route path="/menuCliente" element={<MuenuCliente />}></Route>
15          <Route path="/menuEntrenador" element={<MuenuEntrenador />}></Route>
16          <Route path="/menuNutricionista" element={<NutritionistMenu />}></Route>
17          <Route path="/login" element={<Login />}></Route>
18        </Routes>
19      </BrowserRouter>
20    );
21  }
22
23  export default App;
```

Ilustración 7: Componente frontend App.jsx

Como se observa en la Ilustración 7, por ejemplo, en el caso de acceder a localhost:3000/, se cargará el componente de la landing page. En cambio, al acceder a localhost:3000/menuCliente, se cargará el componente relativo al menú del cliente.

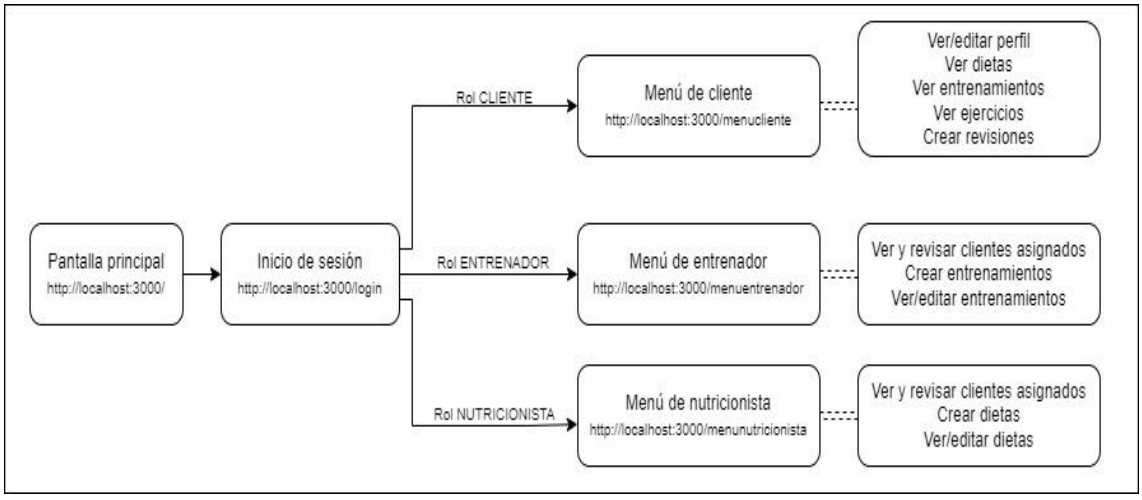

Ilustración 8: Diagrama de flujo y acciones de frontend en base a roles

Esto tiene una limitación, dado que todo usuario ha de haber iniciado sesión para poder acceder a un menú, y, además, un usuario logueado como cliente, no podrá acceder a los menús del personal y viceversa. En la Ilustración 8 se muestra como en función del tipo de rol, el usuario será redirigido a través de una URL u otra y, por tanto, el componente App.jsx mencionado en la Ilustración 7 cargará una vista u otra dependiendo de dicho rol.

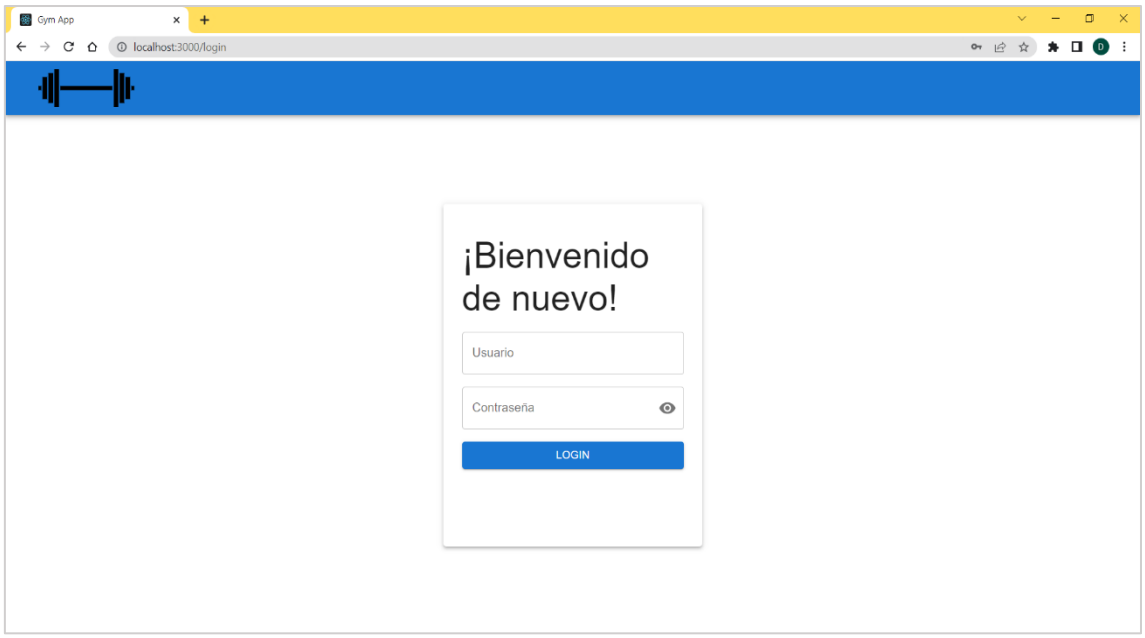

Ilustración 9: Vista de inicio de sesión

Desde la pantalla de login (Ilustración 9), simplemente se recogen las credenciales de inicio de sesión que introduce el usuario en el formulario y se realiza una llamada fetch de tipo post para autenticarlo (llamada http). La instrucción fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, tales como peticiones y respuestas. Éste, al ser uno de los primeros componentes desarrollados, se implementó utilizando fetch. Sin embargo, en el resto de los componentes se ha utilizado una librería llamada Axios [23], que nos permite elaborar peticiones http de forma mucho más sencilla.

Como cada usuario debe tener acceso únicamente al contenido habilitado para su rol y no a todos, se debe redirigir al usuario en función de su rol. Cuando el usuario inicia sesión satisfactoriamente, se genera un token JWT [21] que se almacena en una cookie llamada "loggedUser". Este token es el que luego se incluirá en la cabecera de las llamadas al backend para autenticarse. A nivel frontend, se comprueba en cada menú si el usuario está logueado y su rol.

```
59  useEffect(() => {
60    var loggedUser = window.localStorage.getItem("loggedUser");
61
62    if (loggedUser == null) {
63      navigate("/login");
64    } else {
65      loggedUser = jwt(loggedUser);
66      if (loggedUser.rol === "CLIENTE") {
67        navigate("/menuCliente");
68      }
69      if (loggedUser.rol === "ENTRENADOR") {
70        navigate("/menuEntrenador");
71      }
72    }
73
```

Ilustración 10: UseEffect menú de nutricionista

Como se observa en la Ilustración 10, que muestra el menú del nutricionista, la primera acción según se carga el componente es la comprobación del usuario en base a su rol. Si no se encuentra la información en la cookie (línea 62), implica que el usuario no está logueado, y por tanto se le redirige a la pantalla de inicio de sesión. En cambio, si existe la cookie, se comprueba el rol. Si el rol es el correcto, en este caso el de nutricionista, el componente se cargará de forma correcta. Si por ejemplo tiene otro rol, como puede ser el de Cliente, quiere decir que el usuario logueado es un cliente y se le redirigirá a su propio menú.

A partir de este punto, en cada uno de los menús, se cargan los datos del usuario correspondiente y de la información que se vaya a mostrar en su menú, por ejemplo, los datos de un cliente y un listado de los ejercicios. Esto se hace, como se comentaba en los párrafos anteriores, mediante una petición al backend utilizando la librería Axios [23].

```
191  // Load all daily trainings
192  const loadDailyTrainings = (e) => {
193    var token = window.localStorage.getItem("loggedUser");
194    token = JSON.parse(token);
195
196    const url = `http://localhost:8080/entremaniento-diario/get-all`;
197    axios
198      .get(url, {
199        headers: {
200          "Access-Control-Allow-Origin": "*",
201          Authorization: `Bearer ${token}`,
202        },
203      })
204      .then(
205        (response) => {
206          setDailyTrainingtList(response.data);
207          setDailyTrainingLoading(false);
208        },
209        (error) => {
210          console.log(error);
211        }
212      );
213  };
```

Ilustración 11: Método que realiza petición con cabeceras para obtener todos los entrenamientos diarios

Como se observa en la Ilustración 11, este método, en primer lugar, recoge la cookie, y además construye la URL sobre la que realizar la petición. Al realizar la petición con Axios [23], se indican tres elementos, por un lado, la URL de la petición, por otro las cabeceras, y finalmente el cuerpo. En este caso, al ser una petición de tipo “GET” solo se le indican los dos primeros elementos. La URL, y en las cabeceras, construimos el token concatenando el string “Bearer” con el token obtenido de la cookie.

Una vez se tiene la respuesta, se almacena en el estado del componente.

También se observa en la captura una llamada setDailyTrainingLoading a false. Este es un flag que se define en todos los componentes que realizan llamadas contra el backend. Este flag comienza siempre en estado “true”, indicando que la llamada aún está cargando, y se le da valor false cuando la llamada ha terminado. Es utilizado para mostrar un icono de loading mientras se está procesando la llamada, y dejar de mostrarlo cuando se han obtenido los datos de base de datos.

El resto de código empleado en el frontend es prácticamente estructural, empleado para representar los datos obtenidos de base de datos, organizar las vistas, introducir o dividir los datos. Por ejemplo, en la Ilustración 12, se observa un formulario, en el que el nutricionista puede imputar datos con el fin de crear una dieta diaria. Por añadir más ejemplos de otros roles, se muestra el menú de cliente. En la Ilustración 13 se observa la vista del perfil de cliente (que se carga según inicia sesión), en la que se muestra su información personal, permitiéndole visualizarla y editarla. Otro ejemplo es la vista de ejercicios (Ilustración 14), en la que el cliente podrá obtener información de todos los ejercicios disponibles, pero en este caso, al contrario que en el anterior, no podrá editar la información.

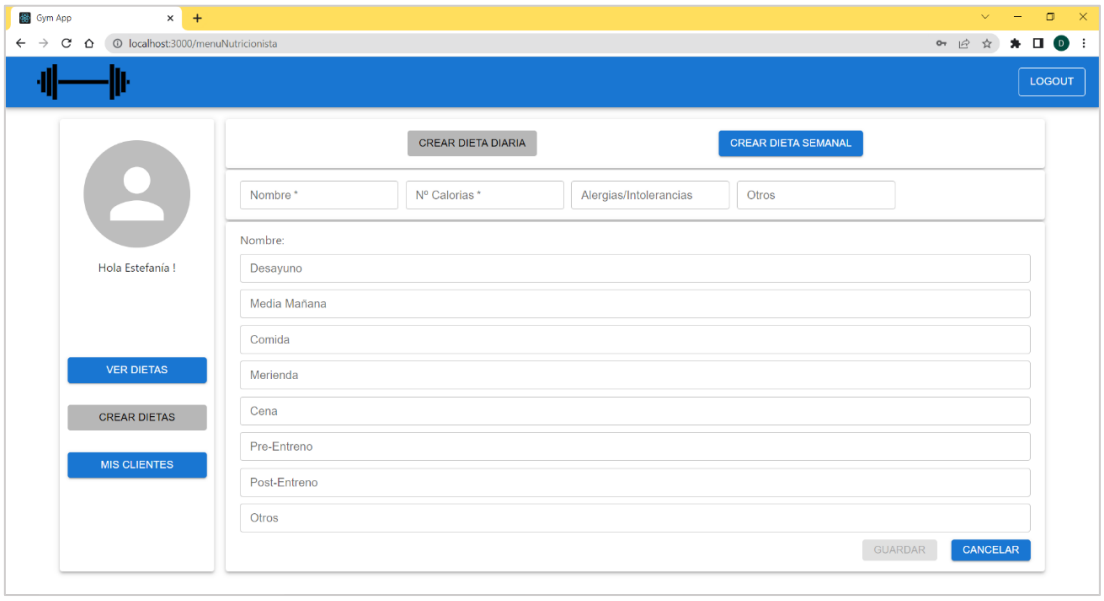

Ilustración 12: Creación de dietas en menú nutricionista

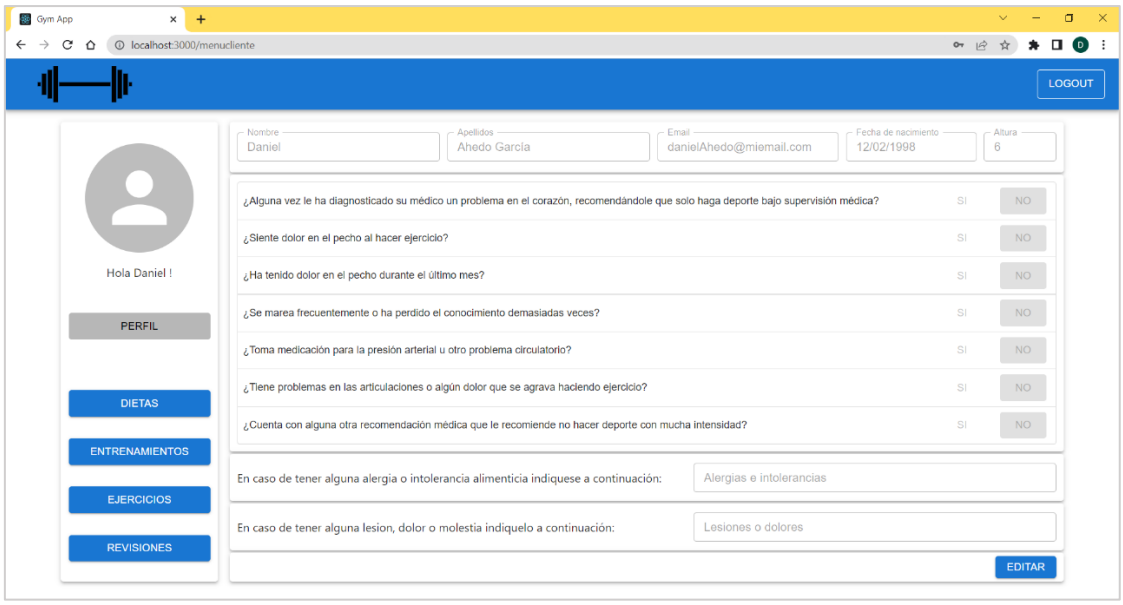

Ilustración 13: Perfil de cliente en menú de cliente

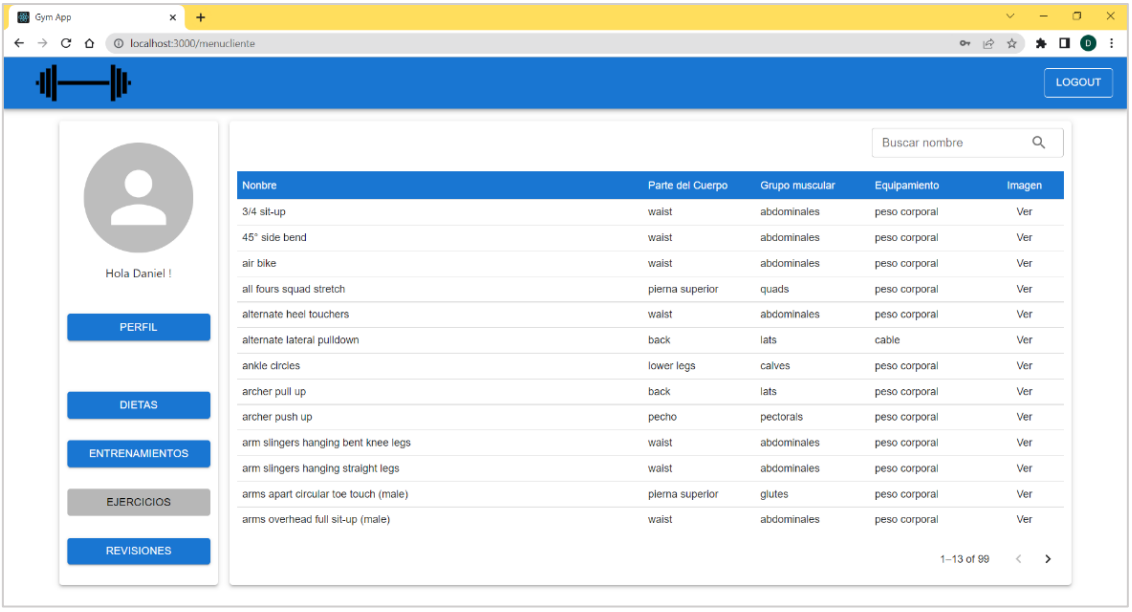

Ilustración 14: Vista de ejercicios en menú de cliente

6.1.2 Imágenes CC0

A pesar de que este no es un punto referente a la implementación como tal del propio frontend, es importante destacar el tipo de licencia de uso seleccionado para las imágenes utilizadas. Todas las imágenes de la interfaz de usuario ([17]–[19]), tanto las empleadas en la página principal, como en los gifs de los ejercicios, son de uso público con licencia CC0 ([20]).

6.2 Capa de negocio

La capa de negocio forma parte del backend de la aplicación. Está formado principalmente por los controladores y los servicios, descritos a continuación.

6.2.1 Controladores

Los controladores son la parte del sistema que interactúa directamente con el frontend. Son los encargados de recibir las peticiones (en el caso de esta aplicación, solamente GET y POST), y devolver una respuesta. Los controladores implementados en la aplicación realizan algunas funciones básicas como buscar un elemento en base de datos por su id, buscar todos los elementos de una entidad concreta, crear o actualizar información.

```java
@RestController
@RequestMapping("clientes")
public class ClienteRest {

    // Inyeccion de dependencia
    @Autowired
    private ClienteService clienteService;

    // Metodos de peticion HTTP

    // Get one by id
    @RequestMapping(value = "/{userId}", method = RequestMethod.GET)
    public Optional<Cliente> getById(@PathVariable long userId) {
        return Optional.ofNullable(clienteService.findById(userId));
    }

    // Create/update cliente
    @RequestMapping(value = "/create-update", method = RequestMethod.POST)
    public Optional<Cliente> createUpdateCliente(@RequestBody Cliente cliente) {
        return Optional.ofNullable(clienteService.createUpdateCliente(cliente));
    }
}

```

Ilustración 15: Controlador de cliente con métodos GET y POST

En la Ilustración 15 se muestra la definición de los controladores de la entidad cliente. Se dispone de un endpoint de tipo GET para buscar un cliente por un ID dado, y otro de tipo POST para crear o actualizar la información de un cliente.

6.2.2 Servicios

Los controladores una vez reciben una petición, llaman a los servicios, que serán los encargados de procesar la información recibida, y llevar a cabo las tareas necesarias. En el caso de esta aplicación, no hubiera sido necesario crear esta capa, ya que no se implementa ninguna lógica de negocio. Los controladores podrían haber llamado directamente a la capa de acceso a datos, pero finalmente se decidió implementarlo de cara a facilitar la realización de futuros cambios que puedan introducir complejidad adicional en la plataforma.

Siguiendo con el ejemplo del cliente que se mostraba en la Ilustración 15, en la Ilustración 16 se muestra cómo sería el servicio del cliente asociado que permite buscar y actualizar un cliente.

```java
@Service
@Transactional
public class ClienteServiceImpl implements ClienteService {

    @Autowired
    private ClienteDAO clienteDao;

    @Override
    public Cliente findById(long id) {
        return clienteDao.findById(id);
    }

    @Override
    public Cliente createUpdateCliente(Cliente cliente) {
        return clienteDao.createUpdateCliente(cliente);
    }

}

```

Ilustración 16: Implementación del servicio de cliente

6.2.3 JWT

Los tokens JWT [21] se utilizan para agregar autenticación y autorización a los enpoints de nuestra API REST [11]. Por una parte, la autenticación es el proceso de identificar y verificar la identidad de los usuarios que intenten iniciar sesión. Esto se hace mediante la comprobación de las credenciales (usuario y contraseña) introducidas en el sistema para iniciar sesión. Por otra parte, la autorización valida si el usuario realmente tiene los permisos necesarios para acceder a un recurso o realizar alguna función.

La aplicación desarrollada sigue la misma estructura que la definida en la Ilustración 17. Cuando el usuario inicia sesión introduciendo sus credenciales, el backend recibe dicha petición. Se buscan esas credenciales en base de datos, y si son correctas, se crea un token JWT [21] y se le devuelve. En dicho token, se devuelven además de los datos predefinidos (fecha creación, fecha expiración…) el id del usuario logueado y su rol, para utilizarlo a nivel frontend.

Ilustración 17: Flujo peticiones JWT [24]

Una vez el usuario inicia sesión, ya dispone de ese token y ahora se debe proceder a cargar su información mediante peticiones de tipo GET a contra la API REST [11]. En estas peticiones, se añade el token en las cabeceras para que el backend pueda identificar y autorizar al usuario. Por ejemplo, los clientes no tienen acceso a los endpoints de nutricionistas o entrenadores. Esto lo gestiona el backend en función del rol del usuario que hace la petición. Lo especificamos a nivel de endpoint, como se ve en la Ilustración 18. Este endpoint corresponde con la creación y actualización de dietas diarias. Como es lógico, solo podrán acceder a él los usuarios con rol *NUTRICIONISTA* y esto lo conseguimos utilizando la notación @PreAuthorize("hasRole('NUTRICIONISTA')").

```
@RequestMapping(value = "/create-update-alimentacion", method = RequestMethod.POST)
@PreAuthorize("hasRole('NUTRICIONISTA')")
public Optional<AlimentacionDiariaDieta> createUpdateAlimentacion(
        @RequestBody AlimentacionDiariaDieta alimentacionDiariaDieta) {
    return Optional.ofNullable(service.createUpdateAlimentacionDiaria(alimentacionDiariaDieta));
}
```

Ilustración 18: Endpoint con seguridad en base a rol

6.3 Capa de acceso a datos

La capa de acceso a datos o capa de persistencia es la encargada de almacenar u obtener del almacén de datos seleccionado, en este caso una base de datos relacional.

6.3.1 Base de datos

El gestor de base de datos seleccionado para este proyecto ha sido MySQL [1]. La única labor que se ha realizado a nivel de base de datos ha sido su creación. Mas concretamente, para crear la base de datos se ha utilizado el software DBeaver [9], el cual también se ha utilizado para cargar algunos datos de prueba manualmente, o para ejecutar consultas y comprobar los datos de salida de los controladores.

Todas las tablas y relaciones se han creado automáticamente a nivel de backend, mediante anotaciones JPA y el software Hibernate [25]. Por ejemplo, en la Ilustración 19 vemos la entidad de Ejercicio.

```
13 @Entity
14 public class Ejercicio {
15
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private Integer id;
19
20     @Column
21     private String nombre;
22
23     @Column
24     private String grupoMuscular;
25
26     @Column
27     private String parteCuerpo;
28
29     @Column
30     private String urlImagen;
31
32     @Column
33     private String equipamiento;
34
35     @OneToMany(mappedBy = "ejercicio", cascade = CascadeType.ALL)
36     private Set<EjercicioEntrenamientoDiario> ejerciciosEntrenamientos;
37
```

Ilustración 19: Definición de la entidad Ejercicio.

Haciendo uso de las anotaciones de JPA (Java Persistence API), con @Entity, definimos la clase como una entidad de BBDD. Con el uso de @ID y @GeneratedValue, indicamos que ese será un campo de primary key autogenerado y con @Column y @oneToMany generamos columnas y relaciones.

En otras clases podemos encontrar notaciones un poco menos habituales, como por ejemplo @PrimaryKeyJoinColumn(name = "user_id"). Esta se utiliza en las clases de cliente entrenador y nutricionista para que hereden el ID de la clase usuario.

Finalmente, una vez arrancamos el backend y se crean todas las clases, obtenemos la estructura base de datos de la Ilustración 20.

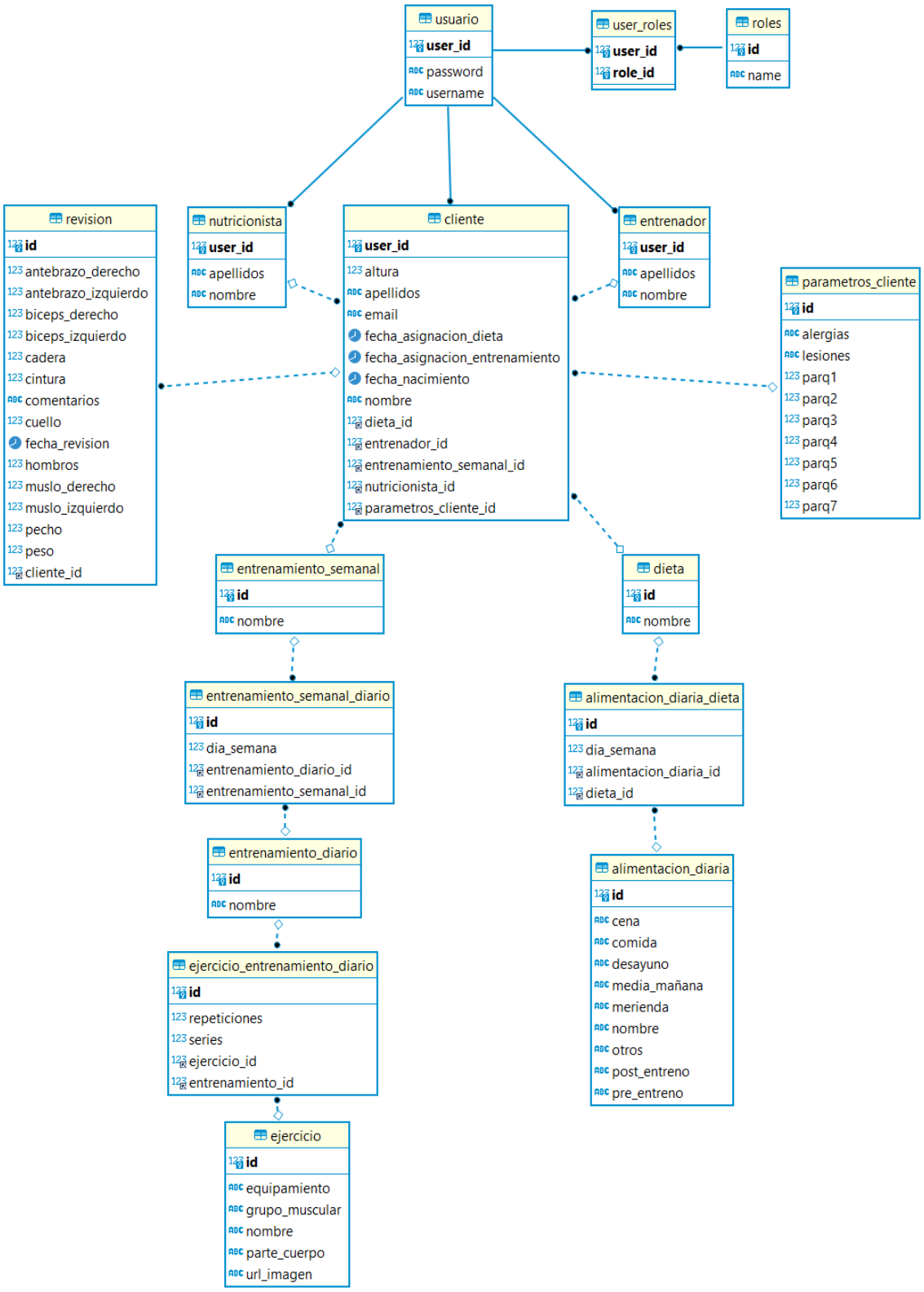

Ilustración 20: Esquema de BBDD

6.3.2 DAO

El último elemento que falta por mencionar son los DAO (Data Access Object). Estos son el paso intermedio entre los servicios y la base de datos. Cada entidad definida debe tener su propio DAO, el cual será el encargado de encapsular la información sobre la capa de persistencia y proveer de una interfaz CRUD a cada entidad. Un ejemplo del DAO del nutricionista lo tenemos en la Ilustración 21:

```
12 @Repository
13 public class NutricionistaImpl implements NutricionistaDAO {
14
15     @Autowired
16     private EntityManager entityManager;
17
18     @Override
19     public Nutricionista findById(long id) {
20
21         Session currentSession = entityManager.unwrap(Session.class);
22         Nutricionista nutricionista = currentSession.get(Nutricionista.class, id);
23         return nutricionista;
24     }
25
26 }
```

Ilustración 21: DAO Nutricionista

7 Validación y pruebas

Durante el desarrollo del proyecto se han ido realizando determinadas pruebas para comprobar el correcto funcionamiento del sistema, así como para detectar posibles errores.

Las pruebas realizadas han sido en su mayoría pruebas de integración, las cuales se detallan en la sección siguiente. No se han realizado pruebas unitarias ya que la funcionalidad implementada es bastante simple y directa, por lo que se prefirió invertir los esfuerzos en verificar el correcto funcionamiento de los distintos componentes de la plataforma.

7.1 Pruebas de integración

Mediante las pruebas de integración se comprueba que todos los elementos unitarios que componen el sistema software funcionan juntos correctamente probándolos al unísono. En este caso, las pruebas se realizaron en un punto intermedio, validando desde Postman [10] los distintos endpoints del backend, sin llegar a involucrar al frontend.

Estas comprobaciones se han realizado durante el propio desarrollo, de forma previa a la integración con el frontend y se han llevado a cabo mediante la herramienta Postman [10], comprobado el funcionamiento de cada uno de los endpoints que expone la API REST [11].

Posteriormente, a medida que se realizaron las vistas del frontend, se comprobó que el sistema completo funcionaba satisfactoriamente, gestionando adecuadamente las peticiones del frontend al backend y devolviendo la información correcta hasta el frontend.

Algunos de los endpoints se encargan de solicitar datos, otros de modificarlos o insertarlos. Por tanto, de forma previa a la realización de estas pruebas, y mediante el uso de un script, se ha poblado la base de datos con datos de ejemplo. En el entorno de Postman [10] y con el fin de facilitar la realización de test, se han configurado dos variables de entorno llamadas “BaseURL” y “Token”.

Ilustración 22: Variables de entorno Postman

El uso de estas variables (Ilustración 22) nos facilita la realización de diversas pruebas. Por ejemplo, en este caso, mediante la variable BaseURL, se configura la dirección y el puerto a la que apuntarán nuestras peticiones, en este caso apuntan a localhost:8080. En nuestro workspace de Postman [10] se dispone de más de 15 peticiones. En el caso de

desplegar la API REST [11] en otra maquina y tener que acceder desde otra ip:puerto, bastaría con cambiar el valor de la variable en lugar de modificar el valor en cada una de las 15 peticiones.
Por otro lado, la variable Token no tiene un valor por defecto, el valor de esta petición se recoge del resultado de realizar la llamada de login de un usuario.

En el siguiente ejemplo (Ilustración 23), en esta llamada de autenticación, o de login, se utiliza la variable mencionada en primer lugar para construir la URL. Para utilizarla, se referencia entre dos llaves, y adquiere el valor que tenga asignado. Además, en esta petición, se le pasa un body en formato JSON y nos retorna el token que se asignará a la segunda variable.

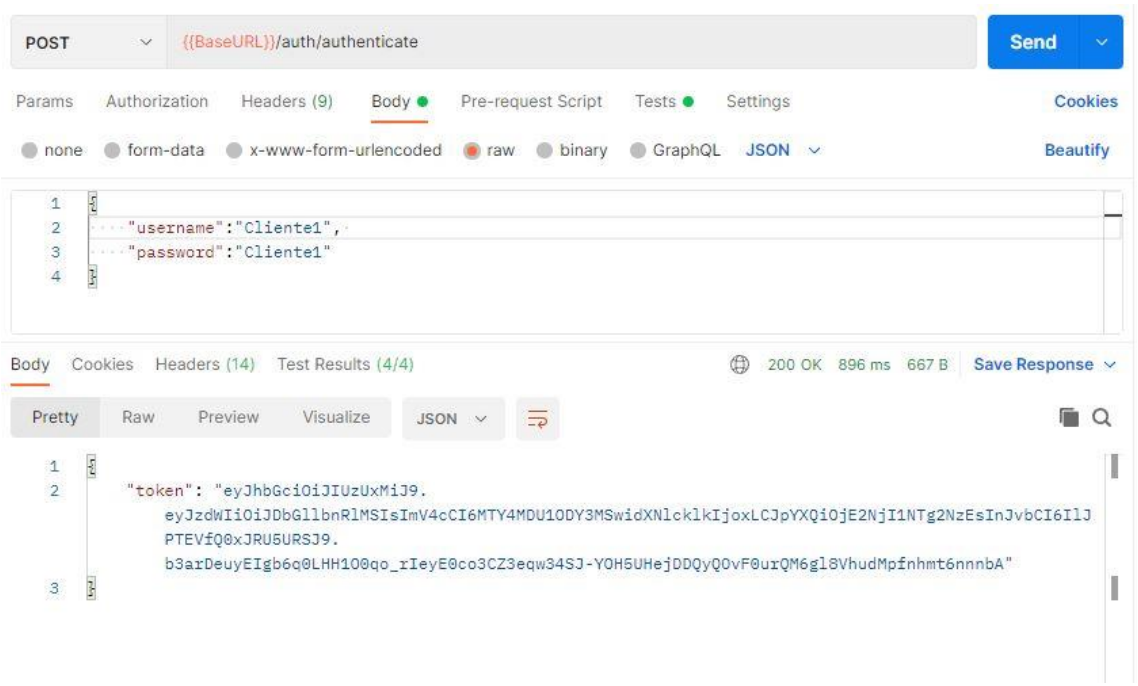

Ilustración 23: Petición de autenticación desde Postman

Esta llamada al ser la propia de la autenticación, no es necesario asignarle ningún token en las cabeceras. En cambio, para el resto de las peticiones si será necesario. Por ejemplo, en la petición encargada de buscar un cliente por su ID, si será necesario identificarse mediante el token que nos fue retornado por la autenticación.

Esto se hará en la pestaña “Autorización” de la llamada (ver Ilustración 24), indicando que se utilizará un token de tipo Bearer (es decir, de tipo JWT [21] con prefijo Bearer, el programa lo indica así porque por convención se usa Bearer como prefijo), e indicando el valor que se le quiere dar. En nuestro caso, al tener almacenado el valor en una variable, se la referencia entre llaves como en el caso anterior.

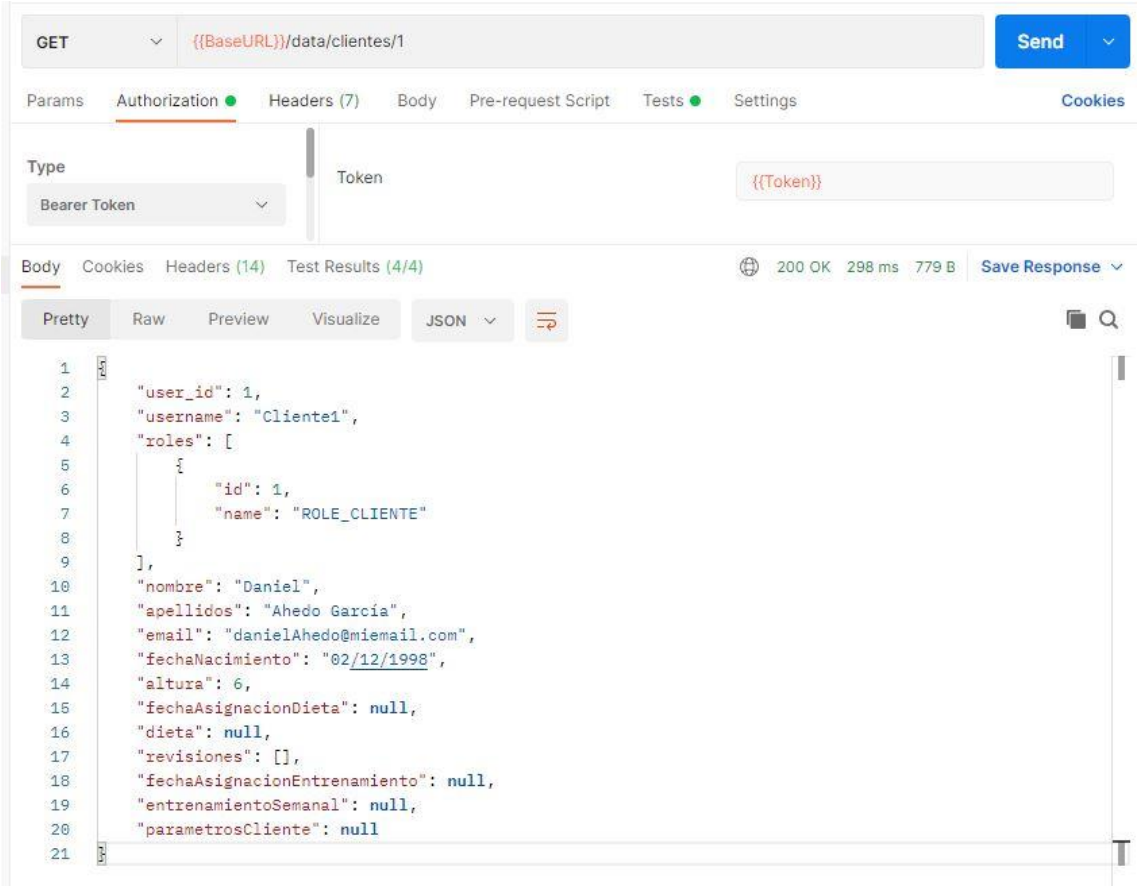

Ilustración 24: Petición Postman GET cliente por ID

Se podría parar aquí y comprobar manualmente cada vez que los datos obtenidos son los deseados. No obstante, como este proceso sería muy tedioso y no escalaría bien ante un aumento del número de pruebas, Postman [10] permite automatizar estas pruebas con una serie de configuraciones extra, como muestra la Ilustración 25.

Por ejemplo, algunos de los test que se han implementado en esta llamada son:

- Detectar que la llamada retorne un código 200(OK)
- Que la respuesta contenga un cuerpo o "body"
- Que este body sea en formato JSON.
- Que se obtenga la respuesta en un tiempo menor que uno dado.

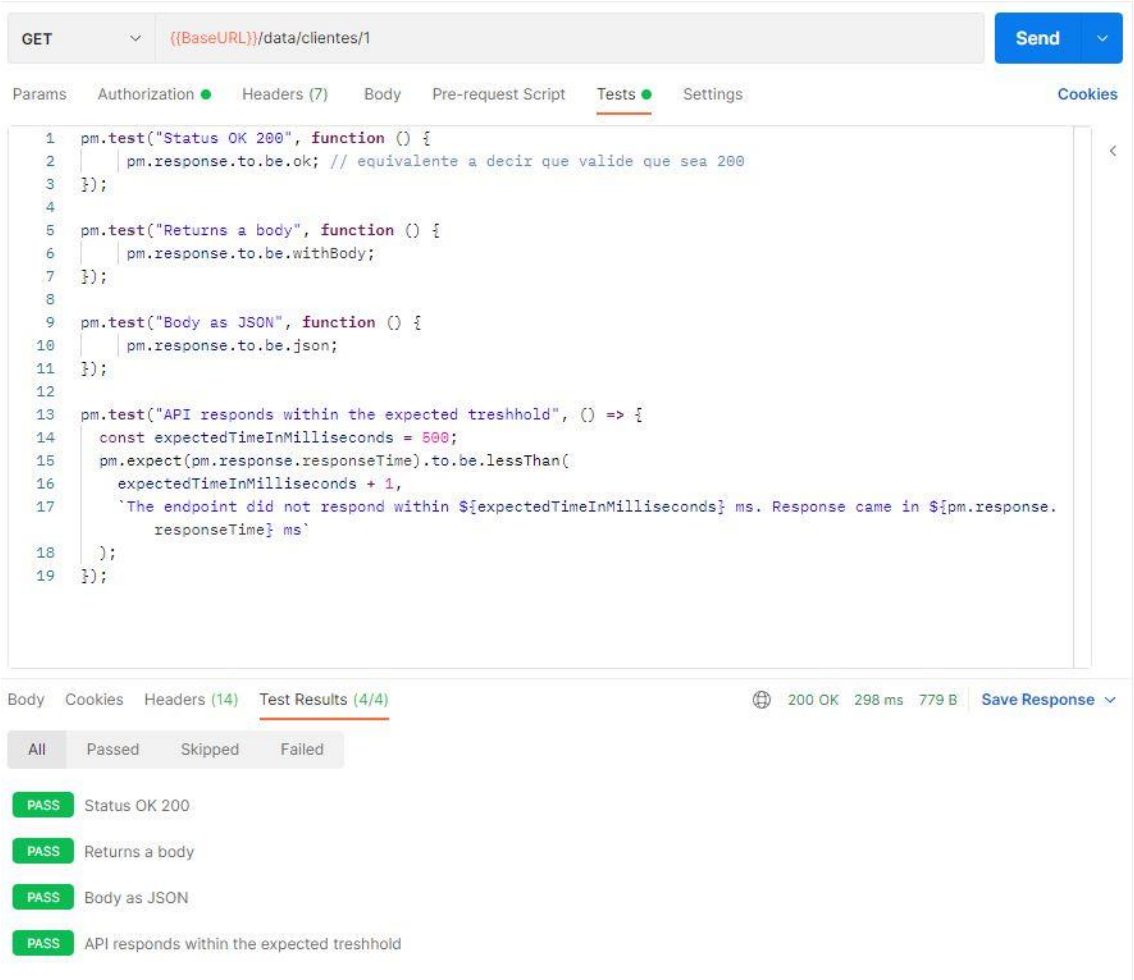

Ilustración 25: Test automatizados implementados en Postman

8 Conclusiones

El objetivo de este Trabajo Fin de Grado ha sido crear una aplicación web que brinde a los centros deportivos facilidades para ofrecer atenciones de dietética y de entrenamiento personalizadas a sus clientes. Se considera que se ha alcanzado dicho objetivo, ya que se dispone de una aplicación plenamente funcional y que cumple con los requisitos especificados. A lo largo de estos meses se ha desarrollado cada una de las partes de la aplicación, probándolas de forma individual, y todas ellas como parte de un único sistema, realizando durante el propio desarrollo pequeñas mejoras que pudieran afectar al rendimiento o a la experiencia de usuario.

Finalmente, tras todo este desarrollo se ha conseguido generar un primer MVP. Cuando se trabaja en agile, se describe un MVP como un Producto Mínimo Viable, es decir, una primera versión que incluye toda la funcionalidad y que podrá ser mostrada al público con el fin de obtener su feedback y mejorar el producto en próximas versiones.

Uno de los puntos más retadores de este proyecto, y del cual no partía con conocimiento previo ha sido la autenticación y autorización mediante tokens JWT [21]. Este punto, me ha hecho realizar una gran labor de investigación, viendo cómo funcionaba cada uno de los puntos y como se implementaban. He notado la falta de un equipo en el que apoyarme y con los que compartir conocimientos, y esto me ha hecho realizar muchas pruebas ensayo-error hasta conseguir comprender su funcionamiento y que funcionase correctamente.

Con el desarrollo de este proyecto he adquirido muchos conocimientos sobre el desarrollo web además de afianzar los conocimientos y que había adquirido en el pasado a través de mi experiencia laboral y académica.

Por todo ello, a título personal, me considero satisfecho con el trabajo realizado. Sin duda, este aprendizaje me será muy útil en el futuro de cara a afrontar nuevos proyectos.

9 Trabajos futuros

Como se comenta en el apartado anterior, esto no es más que un primer MVP, por lo que hay mucha capacidad de mejora. Como en la mayoría de proyectos software, la capacidad de mejora es prácticamente infinita, siempre se puede hacer la interfaz de usuario un poco más bonita, o un poco más amigable para el usuario, por no hablar del rendimiento.

En el caso concreto de este proyecto, durante el desarrollo se han identificado algunos puntos de mejora que se podrían incluir en futuras versiones con el objetivo de mejorar la plataforma.

Algunos de estos puntos de mejora son los siguientes:

- Implementar una funcionalidad de edición de dietas y entrenamientos genéricos. En la versión actual, tanto las dietas como los entrenamientos genéricos se encuentran en dos ficheros de formato JSON, por tanto, se dificulta su edición o la inserción de nuevos valores por parte del personal del gimnasio. Se podría crear un apartado en los menús de entrenador y de nutricionista que les permita modificar dichos ficheros de una forma sencilla.

- Parametrizar el valor que indica el periodo de revisiones. En la versión actual, los clientes pueden imputar revisiones en la plataforma cada 7 días. Este valor es fijo y no se puede cambiar. En futuras versiones, se podría parametrizar este valor, de forma que, si el usuario no tiene asignado ningún entrenador/nutricionista, se mantengan estos 7 días por defecto. En cambio, si tiene contratado un plan de entrenamiento/nutrición, el personal del gimnasio pueda modificar este valor con el fin de realizar un seguimiento más preciso de su evolución.

- A nivel backend, crear en base de datos de forma automática, tanto a nivel de dietas diarias como entrenamientos diarios, un registro correspondiente con el día libre. Este día libre implicara que ese día no se realizara entrenamiento o dieta. De esta forma, a nivel frontend, en los apartados del personal del gimnasio, más concretamente en los de creación de dietas/entrenamientos semanales se podrá incluir este valor por defecto en los 7 días de la semana. Este pequeño cambio a nivel de interfaz hará que en caso de que el personal del gimnasio no necesite rellenar todos los días de la semana, se ahorrará el trabajo de rellenar aquellos que pretende dejar libres.

- Adaptar la interfaz de usuario para su funcionamiento correcto en interfaces móviles. Para ello, a nivel frontend, se deberá modificar el diseño de los componentes de manera que sean “responsivos”, es decir, que se adapten a la altura y anchura de la pantalla del dispositivo.

- Añadir al usuario en base de datos un campo de foto de perfil que se pueda mostrar en su menú en la parte superior izquierda. Actualmente, a nivel frontend, hay definido un espacio para poner dicha foto, pero aún no está implementada su funcionalidad a nivel backend.

Bibliografía

[1] "MySQL." https://www.mysql.com/ (accessed Sep. 04, 2022).
[2] "Java." https://www.java.com/es/download/help/whatis_java.html (accessed Sep. 04, 2022).
[3] "Spring." https://spring.io/ (accessed Sep. 04, 2022).
[4] "React." https://es.reactjs.org/ (accessed Sep. 04, 2022).
[5] "Maven." https://maven.apache.org/ (accessed Sep. 04, 2022).
[6] "NPM." https://www.npmjs.com/ (accessed Sep. 04, 2022).
[7] "Git." https://git-scm.com/ (accessed Sep. 04, 2022).
[8] "GitKraken." https://www.gitkraken.com/ (accessed Sep. 04, 2022).
[9] "DBeaver Community." https://dbeaver.io/ (accessed Sep. 04, 2022).
[10] "Postman API Platform." https://www.postman.com/product/what-is-postman/ (accessed Sep. 04, 2022).
[11] "API REST." https://www.ibm.com/es-es/cloud/learn/rest-apis (accessed Sep. 07, 2022).
[12] "Spring Tool Suite." https://spring.io/tools (accessed Sep. 04, 2022).
[13] "Visual Studio Code." https://code.visualstudio.com/ (accessed Sep. 04, 2022).
[14] "Scrum." https://www.atlassian.com/es/agile/scrum (accessed Sep. 04, 2022).
[15] "Repositorio Aplicacion-TFG-Gimnasio." https://github.com/dahedo/Aplicacion-TFG-Gimnasio/tree/develop (accessed Sep. 06, 2022).
[16] "El cuestionario PAR-Q." https://saludable.unizar.es/el-cuestionario-par-q (accessed Sep. 04, 2022).
[17] Pixabay, "Yoga Pose Trikonasana Bikram." https://pixabay.com/vectors/yoga-yoga-pose-trikonasana-bikram-32124/ (accessed Sep. 04, 2022).
[18] samtuke, "Dumbbell weight - Openclipart," Sep. 15, 2015. https://openclipart.org/detail/227426/dumbbell-weight (accessed Sep. 04, 2022).
[19] E. Cantagallo, "Fitness Exercises | Kaggle." https://www.kaggle.com/datasets/edoardoba/fitness-exercises-with-animations (accessed Sep. 04, 2022).
[20] "Creative Commons — CC0." https://creativecommons.org/publicdomain/zero/1.0/ (accessed Sep. 06, 2022).
[21] "JWT." https://jwt.io/introduction (accessed Sep. 04, 2022).
[22] "MUI: React component library." https://mui.com/ (accessed Sep. 06, 2022).
[23] "Axios." https://axios-http.com/ (accessed Sep. 06, 2022).
[24] "Imagen de openwebinars flujo JWT." https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/ (accessed Sep. 06, 2022).
[25] "Hibernate." https://hibernate.org/ (accessed Sep. 07, 2022).