

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de
Telecomunicación

Aplicación web basada en microservicios para la
predicción de Peso en Rutinas de Ejercicios usando
Machine Learning

Autor: Julio Navarro Vázquez

Tutor: María Teresa Ariza Gómez

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024



Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación web basada en microservicios para la predicción de Peso en Rutinas de Ejercicios usando Machine Learning

Autor:

Julio Navarro Vázquez

Tutor:

María Teresa Ariza Gómez

Profesor titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Aplicación web basada en microservicios para la predicción de Peso en Rutinas de Ejercicios usando Machine Learning

Autor: Julio Navarro Vázquez

Tutor: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

*A toda la gente que me
acompaña día a día*

A mis maestros

Agradecimientos

A lo largo de la elaboración de este trabajo de fin de grado, y sobre todo de mi etapa universitaria, he tenido el placer de contar con el apoyo incondicional de varias personas sin las cuales todo lo que he conseguido no habría sido posible.

En primer lugar, quiero expresar mi más profundo agradecimiento a mi familia, quienes han sido un pilar constante de apoyo y motivación durante toda mi etapa universitaria. Su presencia y comprensión han sido fundamentales para mi desarrollo tanto personal como académico, creyendo en mí cuando a veces ni yo lo hacía.

A Adela, cuyo apoyo ha sido clave en la realización de este trabajo. Las muchas tardes que has dedicado a acompañarme, adelantando tus estudios solo para estar conmigo, han significado mucho para mí.

No puedo dejar de mencionar a los amigos que he hecho durante estos años. Parecía imposible que nos fuéramos a sacar la carrera, pero aquí estamos.

Especial mención a Alberto Chamorro, que ha sido mi compañero de trabajos durante toda la rama de telemática e incluso antes, creando hasta más de tres versiones de la aplicación LibrosTriana en variadas tecnologías. Hasta ha habido alguna que otra cosa en shell script.

Gracias a todos por creer en mí, por vuestro apoyo y por acompañarme en esta significativa etapa. Este trabajo no es solo un reflejo de mis esfuerzos, sino también del amor, la amistad y la colaboración de todas las personas que menciono aquí.

Julio Navarro Vázquez

Sevilla, 2024

Este proyecto consiste en la creación de una aplicación web llamada mytrAIning que proporciona rutinas de ejercicios y predice los pesos que el usuario tiene que ponerle a cada ejercicio según las características de este.

Comenzar a ir al gimnasio puede ser complicado para los principiantes. Existen muchas máquinas e infinidad de ejercicios. Uno de los mayores desafíos para quienes quieren ponerse en forma es saber qué ejercicios realizar, cómo realizarlos y cuánto peso usar en cada uno de ellos.

MytrAIning aborda estos desafíos proporcionando prescripciones de ejercicios físicos personalizados basados en el análisis de datos y el aprendizaje automático. Los usuarios deben seleccionar una rutina y rellenar información sobre sus características físicas y nivel de experiencia en el gimnasio. Utilizando esta información, la aplicación genera una rutina de ejercicios adaptada específicamente a las necesidades y capacidades del usuario.

Los usuarios pueden registrarse, iniciar sesión, realizar predicciones para características físicas determinadas y cerrar sesión.

La aplicación se divide en tres partes, una desarrollada en Java usando el framework Spring, que aborda el corazón de la aplicación, manejando las sesiones y la configuración de seguridad, entre otras funcionalidades; una de *frontend* desarrollada con el *framework* Bootstrap; y otra desarrollada en Python, en la que se ha creado toda la estructura del microservicio de predicción: desde el entrenamiento del modelo hasta el despliegue de la API Rest para ser accesible por la estructura Spring.

En la parte de Java se utilizan las siguientes tecnologías:

- **Java Spring:** Es la base del *backend*. Se encarga de la configuración y el despliegue de la aplicación. Con Spring Security se maneja toda la configuración de seguridad y la autorización de accesos a rutas determinadas.
- **Hibernate y JPA:** Se emplean para la capa de acceso a datos, permitiendo la integración con la base de datos MySQL y la gestión de la entidad de usuario de la aplicación.
- **JWT (JSON Web Tokens):** Se usan para manejar las sesiones de usuario de manera segura. Al iniciar sesión se crea este token y se mantiene en el navegador hasta expirar o hasta que el usuario decida cerrar sesión.

Por otro lado, en la parte de Python las tecnologías principales son:

- **Docker:** Utilizado para meter en un contenedor el microservicio de predicción, facilitando su portabilidad y despliegue.
- **Flask:** Se usa para crear una API en el microservicio de Python que permite la comunicación con Java Spring, enviando y recibiendo datos en formato JSON.
- **Pandas:** Librería utilizada para la creación y manejo del dataset que se utilizó para entrenar el modelo, así como para diferentes operaciones relativas a datos
- **Scikit-Learn:** Librería usada para construir el modelo de Machine Learning *Random Forest*, el cual predice las cargas en las rutinas de ejercicios.

This project involves the creation of a web application called mytrAIning, which provides exercise routines and predicts the weights that the user should use for each exercise based on its characteristics.

Starting to go to the gym can be complicated for beginners. There are many machines and countless exercises. One of the biggest challenges for those who want to get in shape is knowing which exercises to do, how to do them, and how much weight to use for each one.

MytrAIning addresses these challenges by providing personalized exercise prescriptions based on data analysis and machine learning. Users must select a routine and provide information about their physical characteristics and gym experience level. Using this information, the application generates a workout routine tailored specifically to the user's needs and capabilities.

Users can register, log in, make predictions for certain physical characteristics, and log out.

The application is divided into three parts: one developed in Java using the Spring framework, which addresses the core of the application, managing sessions and security settings, among other functionalities; a frontend part developed with the Bootstrap framework; and another developed in Python, where the entire structure of the prediction microservice has been created: from model training to the deployment of the Rest API to be accessible by the Spring structure.

In the Java part, the following technologies are used:

- **Java Spring:** It is the backbone of the backend. It is responsible for the configuration and deployment of the application. With Spring Security, all security configuration and access authorization to certain routes are handled.
- **Hibernate and JPA:** They are used for the data access layer, allowing integration with the MySQL database and management of the application's user entity.
- **JWT (JSON Web Tokens):** They are used to securely handle user sessions. When logging in, this token is created and kept in the browser until it expires or until the user decides to log out.

On the other hand, in the Python part, the main technologies are:

- **Docker:** It is used to containerize the prediction microservice, facilitating its portability and deployment.
- **Flask:** It is used to create an API in the Python microservice that allows communication with Java Spring, sending and receiving data in JSON format.
- **Pandas:** Library used for the creation and handling of the dataset used to train the model, as well as for different data-related operations.
- **Scikit-Learn:** Library used to build the Random Forest Machine Learning model, which predicts the loads in the exercise routines.

Agradecimientos	ix
Resumen	xi
Abstract	xiii
Índice	xiv
Índice de Figuras	xvi
1 Introducción	11
1.1 <i>Motivación</i>	12
1.2 <i>Objetivos</i>	12
1.3 <i>Antecedentes y otras líneas de trabajo</i>	13
1.3.1 Aplicación Android para la rehabilitación física de usuario usando MySql, PHP y JSON.	14
1.3.2 Personal Trainer, entrenador personal para Android.	15
1.3.3 Aplicación Android de rutinas de entrenamiento adaptadas al usuario usando SQLite y JSON	16
1.3.4 Desarrollo de servicio web RESTful para el acceso a base de datos de entrenamiento desde android.	17
1.3.5 Plataforma web para la creación de rutinas de entrenamiento para la app RutinaApp	18
1.3.6 Aplicación móvil de prescripción de ejercicio con acceso a servicio web Spring	19
1.3.7 Plataforma web de prescripción de ejercicios usando Spring	20
1.3.8 Aplicación móvil para la gestión de listas de ejercicios de entrenamiento en React Native	21
1.3.9 Actualización y mejora de la plataforma web de prescripción de ejercicios físicos MyTraining usando Spring y la integración de la pila ELK	22
1.4 <i>Arquitectura de la aplicación</i>	23
1.4.1 Componente Java Spring	24
1.4.2 Microservicio Flask	24
1.5 <i>Estructura de la memoria</i>	25
2 Tecnologías utilizadas	26
2.1 <i>Componente Java Spring – Backend</i>	26
2.1.1 API REST	26
2.1.2 Spring	27
2.1.3 Maven	29
2.1.4 JSON Web Token (JWT)	29
2.1.5 Swagger	30
2.1.6 Persistencia de datos	31
2.2 <i>Microservicio Python - Backend</i>	33
2.2.1 Flask	33
2.2.2 Scikit-Learn	33
2.2.3 Pandas	34
2.3 <i>Frontend</i>	35
2.3.1 Bootstrap	35
3 Herramientas utilizadas	37
3.1 <i>Google Chrome</i>	37
3.2 <i>Visual Studio Code</i>	37

3.3	<i>Docker</i>	39
3.4	<i>Postman</i>	40
3.5	<i>XAMPP Control Panel</i>	41
3.6	<i>GitHub</i>	42
4	Desarrollo del proyecto	44
4.1	<i>Componente Java Spring</i>	44
4.1.1	Controladores	46
4.1.2	Seguridad	49
4.1.3	Entidades	52
4.1.4	Dto	53
4.1.5	Repositorios	54
4.1.6	Servicios	54
4.1.7	Otros ficheros	55
4.2	<i>Microservicio Python</i>	56
4.2.1	Creación del conjunto de datos	57
4.2.2	Generación del modelo Random Forest	61
4.2.3	Entrenamiento del modelo con la librería Scikit-Learn	62
4.2.4	Implementación del microservicio	63
4.2.5	Proceso de Dockerización	64
4.3	<i>Frontend</i>	65
5	Manual de interfaz de usuario	66
6	Conclusiones y futuras líneas de trabajo	69
Anexo A: Instalación y despliegue de la aplicación		70
A)	<i>Instalación de Maven</i>	70
B)	<i>Instalación de Java JDK</i>	71
C)	<i>Instalación de XAMPP y configuración de phpMyAdmin</i>	72
D)	<i>Instalación de Visual Studio Code y de sus extensiones</i>	74
E)	<i>Instalación de Docker Desktop y puesta en funcionamiento del container</i>	77
Referencias		78

ÍNDICE DE FIGURAS

Figura 1: Aplicación de Antonio José Díaz Lora	14
Figura 2: Aplicación de Juan García Piosa	15
Figura 3: Aplicación de Mirian Franco Maireles	16
Figura 4: Aplicación de Jose María Valverde Baena	17
Figura 5: Aplicación de Francisco José Díaz Romero	18
Figura 6: Aplicación de Laura García Corredera	19
Figura 7: Aplicación de Pablo Carmona Rebollo	20
Figura 8: Aplicación de María de los Reyes Machuca Romero	21
Figura 10: Esquema de la arquitectura de la aplicación	23
Figura 11: Logo de REST	26
Figura 12: Logo de JSON	27
Figura 13: Logo de Spring	28
Figura 14: Logo de XML	29
Figura 15: Logo de Maven	29
Figura 16: Logo de JWT	29
Figura 17: Logo de Swagger	30
Figura 18: Interfaz gráfica de Swagger	30
Figura 19: Logo de JPA	31
Figura 20: Logo de Hibernate	31
Figura 21: Logo de MySQL	32
Figura 24: Logo de Scikit-Learn	33
Figura 25: Logo de Pandas	34
Figura 26: Logo de HTML5	35
Figura 27: Logo de CSS3	35
Figura 28: Logo de JavaScript	36
Figura 29: Logo de AJAX	36
Figura 30: Logo de Google Chrome	37
Figura 31: Logo de Visual Studio Code	38
Figura 32: Interfaz gráfica de VSCode	38
Figura 33: Logo de Docker	39
Figura 34: Interfaz gráfica de Docker Desktop	39
Figura 35: Logo Postman	40
Figura 36: Interfaz gráfica de Postman	40
Figura 37: Logo de XAMPP	41

Figura 38: Interfaz gráfica de XAMPP	41
Figura 39: Interfaz gráfica de phpMyAdmin	42
Figura 41: Página de repositorio de GitHub	43
Figura 42: Jerarquía de directorios del componente Java Spring	45
Figura 43: Documentación de los <i>endpoint</i> y las entidades de Swagger	45
Figura 44: AccessController.java	46
Figura 45: UserController.java	47
Figura 46: PredictionController.java	48
Figura 47: Bean securityFilterChain de WebSecurityConfig.java	49
Figura 48: Fragmento de código de JwtUtil.java	50
Figura 49: Método doFilterInternal de JWTCookieFilter	51
Figura 50: User.java	52
Figura 51: UserInput.java	53
Fig 52: CustomUserDetailsService.java	54
Figura 53: application.properties	55
Figura 54: Fragmento del fichero pom.xml	55
Figura 55: Jerarquía de ficheros del microservicio Python	56
Figura 56: crop_dataset.py	57
Figura 57: Conjunto de datos en formato csv tras haber eliminado las columnas	58
Figura 58: Diccionario de ponderaciones de ejercicios del script data_generate.py	58
Figura 59: Fragmento de código de la función calcular_peso_valoracion()	59
Figura 60: Fragmento de código de data_generate.py	59
Figura 61: Función asignar_rutina_y_ejercicios()	60
Figura 62: Versión final del <i>dataset</i> tras añadir el resto de las columnas	60
Figura 63: Fragmento del diccionario de rutinas de data_generate.py	61
Figura 64: Representación del algoritmo Random Forest	61
Figura 65: Fragmento del código del model_train.py	62
Figura 66: Fragmento de código de prediction_msv.py	63
Figura 67: Fichero Dockerfile	64
Figura 68: Fragmento de código Javascript de envío de datos JSON al <i>endpoint</i> prediction	65
Figura 69: Página de inicio de sesión	66
Figura 70: Página de registro de usuario	66
Figura 71: Página del menú principal	67
Figura 72: Página de predicción de pesos	68
Figura 73: Resultado de la predicción	68
Figura 74: Página oficial de Maven	70
Figura 75: Comprobación de instalación de Maven	70
Figura 76: Página oficial de Java JDK	71
Figura 77: Variables de entorno del sistema	71

Figura 79: Página principal de phpMyAdmin	72
Figura 80: Pestaña de privilegios de la página phpMyAdmin	73
Figura 81: Opción de abrir carpeta de Visual Studio Code	74
Figura 82: Apartado de extensiones en Visual Studio Code	75
Figura 83: Extension settings de la extensión “Maven For Java”	75
Figura 84: Extensión Database de Visual Studio Code	76
Figura 85: Extensión Spring Boot Dashboard de Visual Studio Code	76
Figura 86: Página oficial de Docker	77
Figura 87: Comandos para construir y desplegar el contenedor Docker	77

1 INTRODUCCIÓN

La tecnología es el motor de la innovación y tiene el poder de cambiar el mundo.

Tim Cook

En la actualidad, se ha normalizado la velocidad con la que la tecnología avanza. Hace 20 años prácticamente no existían los teléfonos móviles como los conocemos hoy en día, con miles de funcionalidades y cientos de características que nos hacen la vida mucho más fácil.

Aunque lleva algún tiempo en auge, no ha sido hasta hace aproximadamente un año cuando la Inteligencia Artificial ha llegado a los usuarios normales. Parecía magia las primeras veces que metías un *prompt* en chatGPT y te contestaba lo que querías en cuestión de segundos.

El campo del aprendizaje automático (*Machine Learning*) se ha posicionado como una fuerza transformadora en múltiples sectores. Esta rama de la inteligencia artificial no solo está reformando industrias enteras, sino que también está mejorando la calidad de vida de las personas al ofrecer soluciones totalmente personalizadas.

Por otro lado, una de las principales causas de mortalidad en el mundo son las enfermedades cardiovasculares, mayormente producidas por la ausencia de ejercicio físico y una mala alimentación.

En este contexto, la importancia del ejercicio físico no puede ser subestimada. El entrenamiento regular no solo ayuda a mantener un peso saludable y a reducir el riesgo de enfermedades, sino que también es crucial para fortalecer los músculos y mejorar la salud mental.

Sin embargo, el hecho de comenzar en el gimnasio puede llegar a ser abrumador. Muchas personas sienten incertidumbre sobre cómo comenzar, qué ejercicios realizar y cómo ajustar las cargas de peso de manera segura y efectiva. Aquí es donde el aprendizaje automático puede desempeñar un papel importante.

Surge así la idea de **mytrAIning**, diseñada para asistir a las personas que están comenzando en el gimnasio.

Esta aplicación utiliza un modelo de predicción de *Machine Learning* llamado **Random Forest**. Este modelo genera recomendaciones personalizadas sobre qué pesos hay que ponerle a cada ejercicio. Los usuarios simplemente ingresan sus características físicas —como edad, género, frecuencia de entrenamiento y peso corporal— y la aplicación les proporciona un conjunto de ejercicios adecuados y predice con ayuda de la inteligencia artificial los pesos recomendados para cada ejercicio. De esta forma, la aplicación no solo facilita un inicio menos complicado en el gimnasio, sino que también promueve un entrenamiento más seguro y efectivo, personalizado a las necesidades y capacidades de cada usuario.

1.1 Motivación

La motivación para desarrollar esta aplicación nace de una base ya establecida en un proyecto anterior, el cual se centraba en la prescripción de ejercicios físicos. A partir de esa idea inicial y buscando la innovación, decidí hacer uso de la inteligencia artificial para ofrecer una asistencia más personalizada a los usuarios del gimnasio.

Mi interés personal y profesional en *Machine Learning* ha sido el impulso para dar vida a este proyecto. A través de él, busco tener un primer contacto con esta tecnología, ya que tiene un potencial enorme para transformar prácticamente cualquier industria.

El aprendizaje automático ofrece infinitas posibilidades, y su aplicación en el ámbito de la salud física es muy importante.

Frente a los crecientes problemas de salud asociados con el sedentarismo, como la obesidad y las enfermedades cardiovasculares, incentivar ir al gimnasio y la actividad física es más importante que nunca. Al eliminar barreras de entrada para los principiantes y proporcionar una guía amigable y personalizada, la aplicación busca convertirse en un recurso valioso para aquellos que desean mejorar su salud física y bienestar general de una manera eficaz.

1.2 Objetivos

Los objetivos de este proyecto se pueden categorizar en personales y técnicos:

- **Introducción al *Machine Learning*:** Uno de los principales objetivos personales era obtener una introducción práctica al mundo del *Machine Learning*. En este proyecto quería aplicar y entender cómo funcionaban los modelos básicos de *Machine Learning* para poder introducirme en este campo y tener así una primera toma de contacto con esta tecnología. Tras investigar cuál era el mejor modelo predictivo para la idea de mytrAIning, opté por el modelo *Random Forest*.
- **Aprender sobre la arquitectura de aplicaciones completas:** Este proyecto ha servido para poder conocer cómo funciona una aplicación al completo:
 - **Patrón de diseño Modelo-Vista-Controlador:** Uso de arquitectura tradicional de una aplicación como es el patrón MVC para garantizar una estructura de capas clara y escalable.
 - **Uso de *frameworks* y tecnologías:** Uso de Spring para el *backend* y herramientas como Bootstrap y jQuery para el *frontend*, entre muchas otras.
 - **Bases de datos:** Implementación de MySQL junto con JPA e Hibernate para gestionar la persistencia de datos de manera eficiente.
 - **Configuración de seguridad y autenticación:** Establecimiento de medidas de seguridad con Spring Security y *tokens* JWT para la gestión de sesiones y datos de usuarios, implementando sistemas de autenticación y autorización.
 - **Uso de *APIs* Rest para la conexión de los servicios Java Spring y Python:** Implementación de comunicaciones entre el microservicio principal desarrollado en Spring y otro microservicio secundario desarrollado en Python, utilizando Flask. Esta integración demuestra la capacidad de trabajar con arquitecturas basadas en microservicios y la flexibilidad de REST para la comunicación entre diferentes tecnologías.
 - **Implementación de contenedores con Docker:** Aprendizaje y aplicación de contenedores Docker para el despliegue de microservicios.

1.3 Antecedentes y otras líneas de trabajo

El proyecto de prescripción de ejercicio físico mytrAIning tiene muchas versiones a su espalda. Algunas han seguido el flujo natural de mejora con respecto a la anterior y otras, como esta, han cogido la idea base y seguido una línea de trabajo diferente.

Los proyectos mencionados, en orden cronológico, son los siguientes:

- Aplicación Android para la rehabilitación física de usuario usando MySQL, PHP y JSON. [1]
- Personal Trainer, entrenador personal para Android. [2]
- Aplicación Android de rutinas de entrenamiento adaptadas al usuario usando SQLite y Json. [3]
- Desarrollo de Servicio Web RESTful para el Acceso a Base de Datos de Entrenamiento desde Android. [4]
- Plataforma Web para la creación de rutinas de entrenamiento para la App RutinaApp. [5]
- Aplicación móvil de prescripción de ejercicio con acceso a servicio web Spring. [6]
- Plataforma web de prescripción de ejercicios usando Spring. [7]
- Aplicación móvil para la gestión de listas de ejercicios de entrenamiento en React Native. [8]
- Actualización y mejora de la plataforma web de prescripción de ejercicios físicos MyTraining usando Spring y la integración de la pila ELK. [9]

1.3.1 Aplicación Android para la rehabilitación física de usuario usando MySql, PHP y JSON.

Este proyecto fue elaborado por Antonio José Díaz Lora en el año 2014.

La aplicación se centra en la idea de que los usuarios accedan a rutinas de entrenamiento personalizadas asignadas remotamente por un especialista. Estas rutinas incluyen una variedad de ejercicios que los usuarios pueden realizar en sus domicilios, cada uno acompañado de información detallada para garantizar una ejecución adecuada.

Tal y como se ve en la Fig. 1, entre estos recursos se encuentran explicaciones escritas, imágenes de los músculos implicados, un número específico de repeticiones, detalles visuales del ejercicio y videos explicativos. La característica clave de este proyecto es la capacidad para llevar a cabo los ejercicios de manera independiente, sin la necesidad de la presencia física de un entrenador, médico o fisioterapeuta.

Es una aplicación Android donde el cliente se pone en contacto con un servidor web, donde varios scripts PHP son los encargados de realizar todas las funciones ofrecidas.



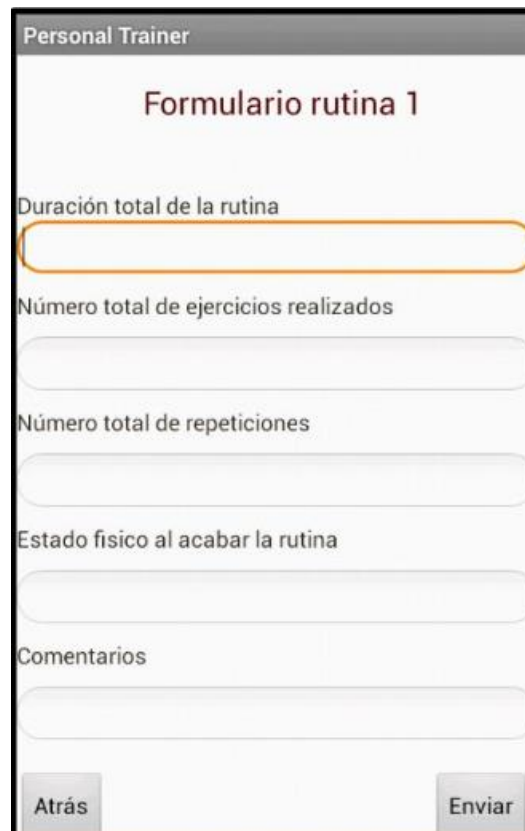
Figura 1: Aplicación de Antonio José Díaz Lora

1.3.2 Personal Trainer, entrenador personal para Android.

Este proyecto fue elaborado por Juan García Piosa en el año 2014.

Este proyecto fue desarrollado en paralelo al de Antonio José Díaz Lora. Por un lado, Antonio José realizó la parte del usuario y por otro, Juan desarrolló la parte del entrenador, pudiendo añadir rutinas a los pacientes, como se puede ver en la Fig. 2.

Su funcionamiento se basa en la gestión de los entrenamientos, rutinas y ejercicios por parte del entrenador en una aplicación Android y codificada en su mayoría con el lenguaje PHP.



The image shows a screenshot of an Android application titled "Personal Trainer". The screen displays a form titled "Formulario rutina 1". The form contains several input fields and buttons. The fields are labeled: "Duración total de la rutina", "Número total de ejercicios realizados", "Número total de repeticiones", "Estado físico al acabar la rutina", and "Comentarios". At the bottom of the form, there are two buttons: "Atrás" (Back) on the left and "Enviar" (Send) on the right. The "Duración total de la rutina" field is highlighted with a yellow border.

Figura 2: Aplicación de Juan García Piosa

1.3.3 Aplicación Android de rutinas de entrenamiento adaptadas al usuario usando SQLite y JSON

Este proyecto fue elaborado por Mirian Franco Maireles en 2015.

La aplicación está diseñada para permitir a los usuarios realizar auto-entrenamientos en cualquier lugar y momento, dando instrucciones para compensar la ausencia de un profesional supervisando la actividad.

En la Fig. 3 se observa que cada usuario tiene una lista de rutinas a las que se puede acceder a los distintos ejercicios que la componen.

Los usuarios pueden crear perfiles ajustados a su nivel de forma física: bajo, medio o alto. Basándose en esta selección, la aplicación muestra los vídeos de rutinas adecuados para el nivel de condición física del usuario. La aplicación también incluye una función de seguimiento del tiempo dedicado a la práctica de ejercicios, representado visualmente en una gráfica.

La aplicación utiliza SQLite para la gestión de datos, almacenando localmente toda la información del usuario en la base de datos del dispositivo. Esto permite que varios usuarios puedan acceder a la aplicación en un mismo dispositivo de forma segura y personalizada.



Figura 3: Aplicación de Mirian Franco Maireles

1.3.4 Desarrollo de servicio web RESTful para el acceso a base de datos de entrenamiento desde android.

Este proyecto fue desarrollado por Jose María Valverde Baena en 2015.

Como se ve en la Fig. 4, la aplicación proporciona una plataforma para el acceso a rutinas de ejercicios, diseñada tanto para usuarios registrados como no registrados. Los usuarios registrados tienen acceso completo a todas las rutinas y ejercicios disponibles, mientras que los no registrados tienen un acceso más limitado.

Dentro de cada rutina, los usuarios pueden optar por un acceso detallado, que incluye descripciones completas, duraciones y recomendaciones específicas de los ejercicios, o un acceso rápido a través de una lista de vídeos.

El proyecto se basa en servicio web RESTful desplegado en un servidor Apache, que maneja las peticiones HTTP de los usuarios, la mayoría codificadas en JSON. Este servicio es el encargado de realizar todas las operaciones de comunicación con la base de datos MySQL, procesando y respondiendo con datos en JSON.

También la aplicación utiliza el servidor externo Imgur para el alojamiento y manejo de imágenes.



Figura 4: Aplicación de Jose María Valverde Baena

1.3.5 Plataforma web para la creación de rutinas de entrenamiento para la app RutinaApp

Este proyecto fue realizado por Francisco José Díaz Romero en 2017.

Esta aplicación tiene parte de aplicación web y parte de aplicación móvil. La aplicación web permite a los usuarios crear y gestionar rutinas de auto-entrenamiento, así como descargarlas para su uso en la aplicación móvil asociada, Rutina App.

Como se puede observar en la Fig. 5, los usuarios tienen acceso a rutinas, que son conjuntos de ejercicios o a ejercicios por separado.

Los usuarios también pueden realizar todas las tareas de autenticación y acceder a un menú extenso que incluye la creación, modificación y listado de rutinas y ejercicios. Se pueden asociar ejercicios existentes a sus rutinas, descargar rutinas y subir vídeos para ejercicios específicos. La plataforma ofrece aparte soporte en línea y permite a los usuarios buscar, listar y descargar rutinas y ejercicios públicos de otros usuarios, con opciones de filtrado avanzado.

La interacción usuario-servidor se maneja a través de un navegador que recibe datos del servicio web REST implementado con el *framework* Java Spring. Este servicio gestiona la autenticación, la seguridad de los datos y las operaciones CRUD sobre los datos de usuarios, rutinas, ejercicios y vídeos.

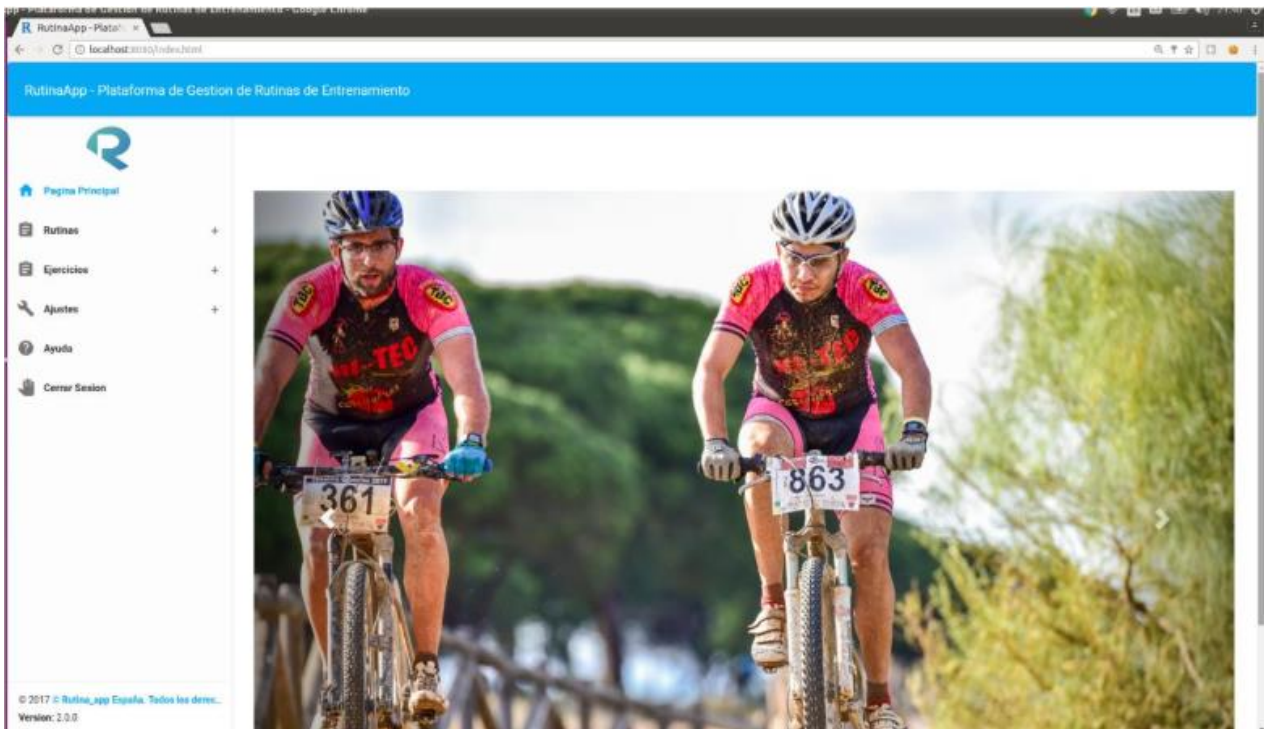


Figura 5: Aplicación de Francisco José Díaz Romero

1.3.6 Aplicación móvil de prescripción de ejercicio con acceso a servicio web Spring

Este proyecto fue realizado por Laura García Corredera en 2018.

La aplicación móvil MyTraining está diseñada para integrar la actividad física en la rutina diaria del usuario, ofreciendo total libertad en cuanto al lugar y momento de realizar el ejercicio.

Se ve en la Fig. 6 que la aplicación permite a los usuarios realizar entrenamientos libres explorando todos los ejercicios disponibles, seguir rutinas específicas asignadas por profesionales, y utilizar un mapa interactivo para localizar centros deportivos, gimnasios o parques cercanos. Por otro lado, los usuarios pueden visualizar listados de rutinas personalizadas y ejercicios clasificados por objetivos, con detalles y vídeos explicativos de cada uno.

Los usuarios interactúan con la aplicación a través de sus dispositivos móviles, obteniendo los datos solicitados del servicio web. La información entre el servidor y la aplicación se transmite utilizando peticiones HTTP codificadas en objetos JSON



Figura 6: Aplicación de Laura García Corredera

1.3.7 Plataforma web de prescripción de ejercicios usando Spring

Este proyecto fue realizado por Pablo Carmona Rebollo en 2018.

La aplicación está diseñada para facilitar la gestión y seguimiento del entrenamiento de pacientes por parte de especialistas. Tiene dos tipos de usuarios: especialistas y pacientes.

Como se ve en la Fig. 7, los especialistas tienen la posibilidad de registrar a los pacientes en la aplicación (Crear usuario) y gestionar su entrenamiento, incluyendo la creación, modificación, y eliminación de rutinas, ejercicios, y vídeos. Cada paciente solo tiene acceso a las rutinas y ejercicios asignados por su especialista. Los especialistas pueden visualizar y modificar no solo sus propias rutinas sino también las creadas por otros especialistas.

La plataforma también permite la descarga de rutinas en formato JSON para su uso en aplicaciones móviles complementarias.

La aplicación se basa en un servicio web REST implementado con el *framework* Spring. Las operaciones de datos, como la creación, lectura, actualización y eliminación de información se manejan mediante SQL y HTTP, utilizando JDBC para interactuar con la base de datos MySQL.

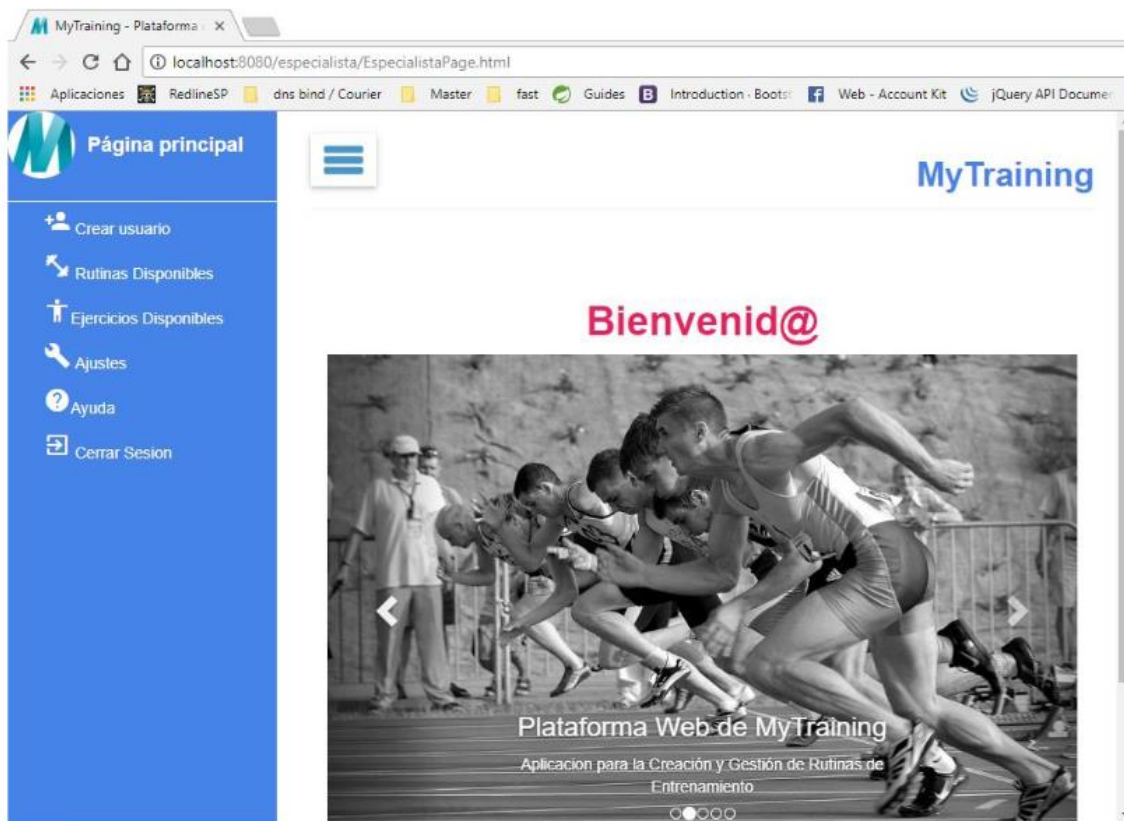


Figura 7: Aplicación de Pablo Carmona Rebollo

1.3.8 Aplicación móvil para la gestión de listas de ejercicios de entrenamiento en React Native

Este proyecto fue desarrollado por María de los Reyes Machuca Romero en 2022.

La principal función de esta aplicación es maximizar el nivel de actividad física del usuario mediante el seguimiento periódico de su rendimiento, ayudando a alcanzar objetivos físicos y promoviendo un estilo de vida saludable.

En la Fig. 8 aparece cómo los usuarios pueden crear y eliminar listas de ejercicios que incluyen enlaces a vídeos, información detallada sobre los ejercicios, y un registro del historial de entrenamiento.

La aplicación también permite enviar estas listas por correo electrónico o Bluetooth en un archivo .TXT en formato JSON y configurar notificaciones locales para recordatorios de entrenamientos programados.



Figura 8: Aplicación de María de los Reyes Machuca Romero

1.3.9 Actualización y mejora de la plataforma web de prescripción de ejercicios físicos MyTraining usando Spring y la integración de la pila ELK

Este proyecto fue elaborado por José Antonio Rufo López en 2023. Es el que más se ha tenido en cuenta a la hora de pensar la idea para mytrAIning.

El proyecto tiene como base la aplicación de Pablo Carmona. Como se puede observar en la vista del especialista (Fig. 9), el especialista puede crear usuarios y configurar las rutinas y los ejercicios disponibles.

Al proyecto base se le implementaron las siguientes mejoras:

- Adición de JWT (Json Web Token) para la autenticación.
- Integración de Hibernate y JPA en la aplicación para simplificar y optimizar la persistencia de datos.
- Incorporación de Elasticsearch, Logstash y Kibana. Esto permite recopilar, almacenar y analizar los registros generados por la aplicación, proporcionando herramientas visuales para interpretar los datos.
- Inclusión de Swagger para la documentación de la API.
- Creación de un nuevo rol "ADMIN".

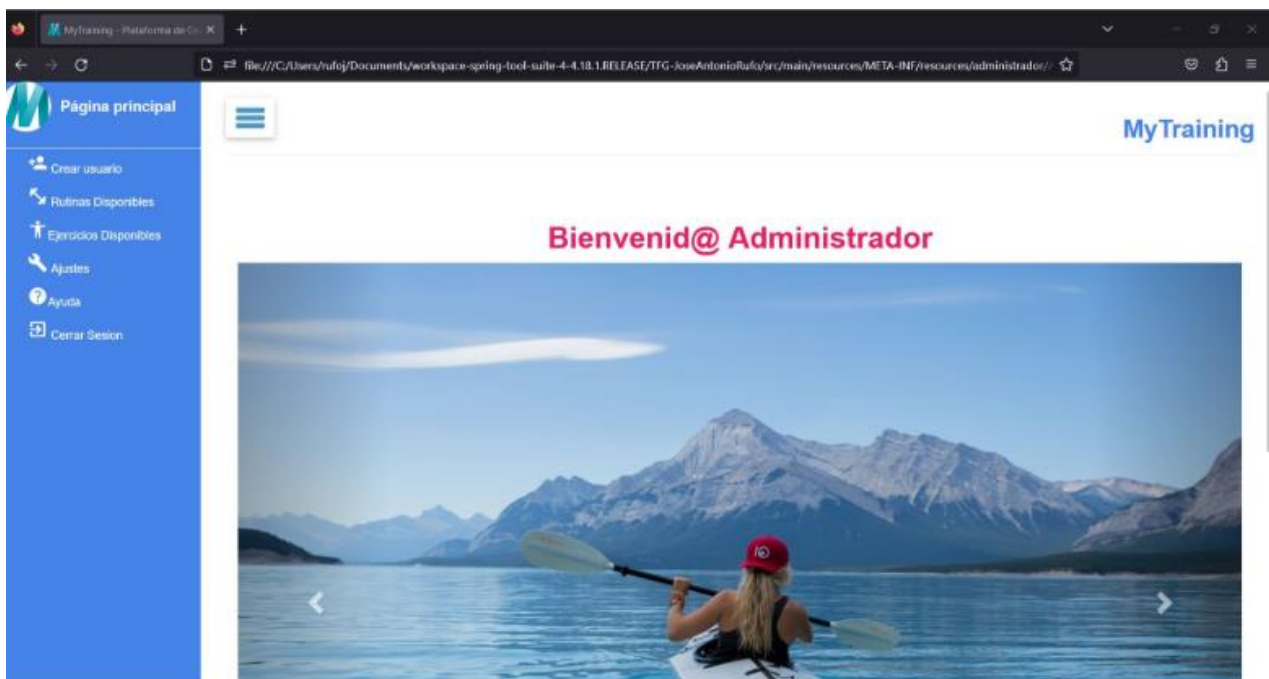


Figura 9: Aplicación de José Antonio Rufo López

1.4 Arquitectura de la aplicación

La aplicación se estructura en dos componentes principales: un *backend* compuesto por un componente Java Spring y un microservicio en Python utilizando Flask, dedicado al *Machine Learning*.

Como se puede observar en la Fig. 10, la interacción entre el *frontend* y el *backend* de Java se gestiona mediante llamadas HTTP al igual que la comunicación entre Java y Python, que se realiza a través de solicitudes POST que transportan datos en formato JSON.

En cuanto a la seguridad, se implementa a través de Spring Security, que gestiona un Json Web Token como método de autenticación. Esta configuración proporciona una capa de seguridad adicional, ya que los tokens JWT son difíciles de falsificar y no requieren que el servidor mantenga un estado de sesión, reduciendo así la carga en el servidor.

Aparte, Spring Security configura los accesos a los *endpoints* permitiendo accesos totales solo a las rutas de inicio de sesión y registro, y a los recursos estáticos del *frontend*; dejando el resto de rutas protegidas con la autenticación del *token*.

La base de datos utilizada es MySQL, que se encarga de almacenar y gestionar todos los datos de usuario y autenticación.

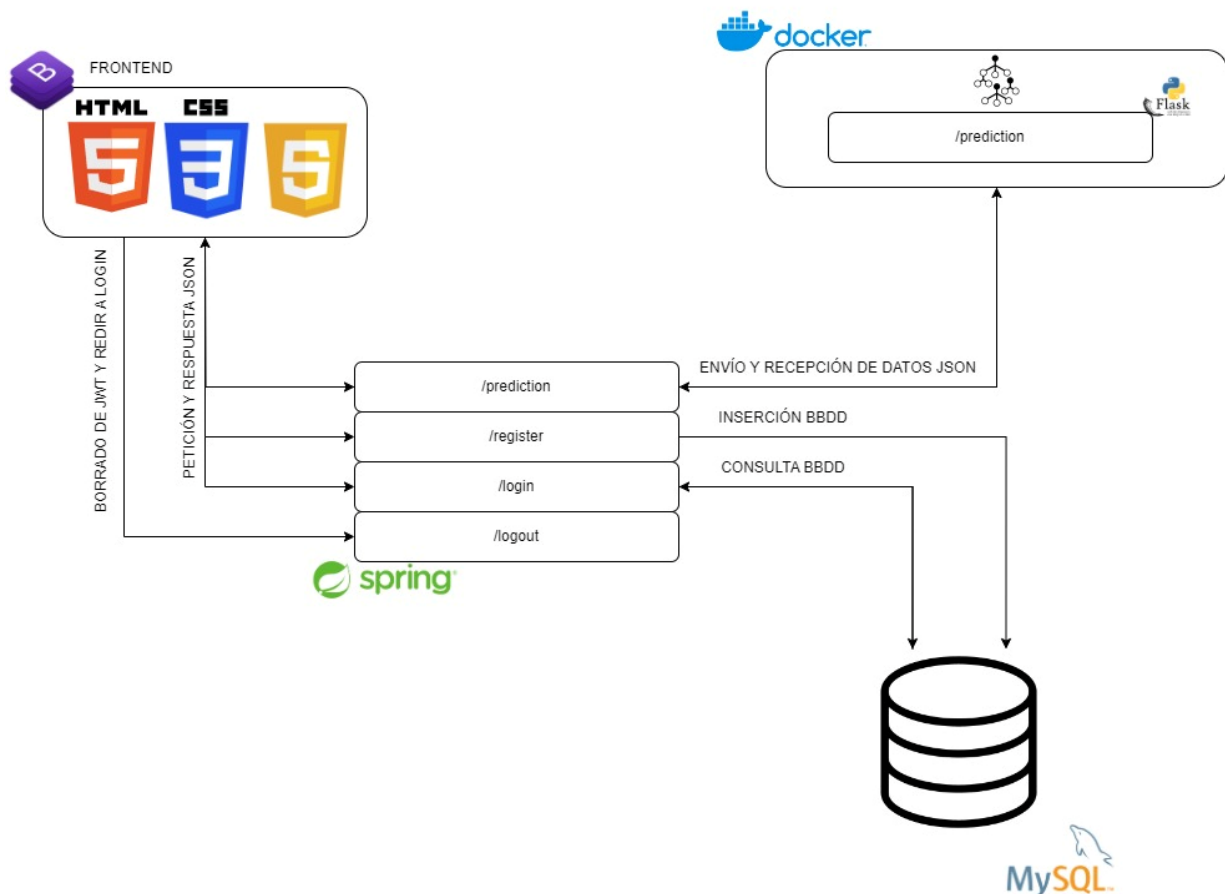


Figura 10: Esquema de la arquitectura de la aplicación

1.4.1 Componente Java Spring

El *backend* en Java Spring maneja la autenticación y la gestión de sesiones del usuario, así como la interacción con la base de datos MySQL. Incluye los siguientes *endpoints*:

- **/register**: Registra nuevos usuarios en la base de datos. Al recibir la solicitud, el sistema verifica si el correo ya existe y, en caso negativo, cifra la contraseña y almacena el nuevo usuario.
- **/login**: Autentica usuarios verificando el correo electrónico y la contraseña con los registros de la base de datos. Si las credenciales son correctas, genera un token JWT (Json Web Token) y lo almacena en una *cookie* en el navegador del usuario. Este token es *STATELESS*, lo que significa que no requiere almacenamiento de sesión en el servidor y contiene toda la información necesaria para validar la sesión del usuario.
- **/logout**: Elimina el token JWT del navegador del usuario y limpia el contexto de seguridad, asegurando que la sesión se cierre correctamente.
- **/javaprediction**: Actúa como un puente entre el *frontend* y el microservicio de Python. Este *endpoint* recibe datos del usuario en formato JSON, los cuales son transformados a un DTO (Data Transfer Object) para convertirlos en campos legibles por el modelo de predicción en Python.

Luego, envía los datos del usuario (edad, género, peso corporal y rutina de ejercicios seleccionada) al microservicio Python y este le devuelve los ejercicios de la rutina y los pesos de cada ejercicio que han sido predichos por el modelo.

1.4.2 Microservicio Flask

Este microservicio utiliza un modelo *Random Forest* integrado en la librería Scikit-Learn para predecir los pesos adecuados para los ejercicios de una rutina basada en las características del usuario. Al recibir datos desde el *endpoint* /prediction de Java, el microservicio realiza las siguientes acciones:

- Procesa la información utilizando la librería Pandas para preparar los datos, recibidos desde el componente Java, en el formato necesario para introducirlos en el modelo.
- Utiliza un modelo de *Machine Learning* pre-entrenado y realiza las predicciones de pesos para los ejercicios de la rutina escogida por el usuario.
- Devuelve los resultados en formato JSON al servicio Java Spring, que a su vez los envía de vuelta al *frontend*.

Todas estas acciones se realizan dentro de un contenedor Docker, lo cual garantiza el aislamiento y la portabilidad del microservicio, asegurando así que se mantenga consistente y eficiente en diferentes entornos de despliegue.

1.5 Estructura de la memoria

La memoria de este proyecto se divide en los siguientes capítulos:

- **Introducción:** Establece el contexto del proyecto, explicando la motivación detrás del desarrollo de la aplicación, los objetivos específicos que busca alcanzar y los antecedentes relevantes que inspiraron su creación.
- **Tecnologías utilizadas:** Detalla todas las tecnologías empleadas en el proyecto, divididas en tecnologías usadas en el componente Java Spring, en el microservicio Python, así como las utilizadas en el *frontend*.
- **Herramientas utilizadas:** Se describen las herramientas de software utilizadas para el desarrollo, prueba y despliegue de la aplicación.
- **Desarrollo del proyecto:** Este capítulo es el núcleo de la memoria, donde se describe detalladamente el proceso de desarrollo de la aplicación, incluyendo la configuración de los componentes del *backend* en Java y Python, la construcción del *frontend* y cómo cada parte fue diseñada e implementada para cumplir con los requisitos del proyecto.
- **Manual de interfaz de usuario:** Proporciona una guía detallada sobre cómo interactuar con la interfaz gráfica de la aplicación.
- **Conclusiones y futuras líneas de trabajo:** Reflexiona sobre los logros del proyecto y posibles mejoras que podrían aumentar la funcionalidad de la aplicación.
- **Anexos:** Contiene información sobre la instalación y configuración de la aplicación.

2 TECNOLOGÍAS UTILIZADAS

En este proyecto se han integrado una serie de tecnologías y *frameworks*. Para una mejor organización, las tecnologías se han organizado en tres bloques: Componente Java Spring, microservicio Python y *frontend*.

A continuación, se detalla el uso de las distintas tecnologías implementadas en cada parte del proyecto:

2.1 Componente Java Spring – Backend

2.1.1 API REST

REST, o *Representational State Transfer*, fue introducido por Roy Fielding en 2000 como un estilo arquitectónico para sistemas hipermedia distribuidos. No es un protocolo ni un estándar, sino más bien un conjunto de principios que guían el desarrollo de *APIs* basadas en web. REST ha ganado una amplia adopción debido a su flexibilidad en la implementación. Para que una interfaz de servicio sea considerada RESTful, debe adherirse a ciertos principios y restricciones delineados por Fielding.

En decir, en el estilo arquitectónico REST, los datos y la funcionalidad se consideran recursos y se acceden utilizando Identificadores Uniformes de Recursos (*URIs*, por sus siglas en inglés *Uniform Resource Identifiers*).

Los recursos son actuados mediante un conjunto de operaciones simples y bien definidas. Además, los recursos deben estar desacoplados de su representación para que los clientes puedan acceder al contenido en varios formatos, como HTML, XML, texto sin formato, PDF, JPEG, JSON, entre otros.

Los clientes y servidores intercambian representaciones de recursos utilizando una interfaz y protocolo estandarizados. Típicamente, HTTP es el protocolo más utilizado, pero REST no lo exige.

Los metadatos sobre el recurso están disponibles y se utilizan para controlar el almacenamiento en caché, detectar errores de transmisión, negociar el formato de representación adecuado y realizar autenticación o control de acceso.



Figura 11: Logo de REST

2.1.1.1 Intercambio de datos: JSON

JSON (*JavaScript Object Notation* - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Se usa durante todo el proyecto para el intercambio de datos entre servicios.

Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.



Figura 12: Logo de JSON

2.1.2 Spring

El *framework* Spring proporciona un modelo completo de programación y configuración para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de implementación.

Un elemento clave de Spring es el soporte infraestructural a nivel de aplicación: Spring se centra en la estructura de las aplicaciones empresariales para que los equipos puedan concentrarse en la lógica empresarial a nivel de aplicación, sin ataduras innecesarias a entornos de implementación específicos.

2.1.2.1 Spring Boot

Spring Boot facilita la creación de Aplicaciones Spring autónomas y de calidad.

Spring Boot proporciona la herramienta para desplegar la plataforma Spring junto a librerías de terceros o dependencias, para añadir más funcionalidades. La mayoría de las aplicaciones de Spring Boot necesitan una configuración básica de Spring para ser ejecutadas.

2.1.2.2 Spring Data

La misión de Spring Data es proporcionar un modelo de programación familiar y consistente basado en Spring para el acceso a datos.

Facilita el uso de tecnologías de acceso a datos, bases de datos relacionales y no relacionales, marcos de *map-reduce* y servicios de datos basados en la nube.

2.1.2.3 Spring Security

Spring Security es un marco de autenticación y control de acceso potente y altamente personalizable. Es el estándar de facto para asegurar aplicaciones basadas en Spring.

Spring Security es un marco que se centra en proporcionar tanto autenticación como autorización a aplicaciones Java. La característica principal de Spring Security es la sencillez de su implementación.

2.1.2.4 Spring Session

Spring Session facilita el soporte para sesiones en clústeres sin estar vinculado a una solución específica del contenedor de aplicaciones. También proporciona integración transparente con:

- **HttpSession:** Permite reemplazar HttpSession de una manera neutral para el contenedor de aplicaciones (es decir, Tomcat), con soporte para proporcionar IDs de sesión en encabezados para trabajar con APIs RESTful
- **WebSocket:** Proporciona la capacidad de mantener viva la HttpSession al recibir mensajes WebSocket
- **WebSession:** Permite reemplazar WebSession de Spring WebFlux de una manera neutral para el contenedor de aplicaciones

2.1.2.5 Spring Rest Docs

Spring REST Docs ayuda a documentar servicios RESTful

Combina documentación escrita a mano con AsciiDoctor y fragmentos autogenerados producidos con Spring MVC Test. Este enfoque libera de las limitaciones de la documentación producida por otras herramientas .

Ayuda a producir documentación precisa, concisa y bien estructurada. Esta documentación permite a tus usuarios obtener la información que necesitan con un mínimo de complicaciones.



Figura 13: Logo de Spring

2.1.3 Maven

Maven es la base del proyecto mytrAIning. Esta tecnología busca una forma estandarizada de construir los proyectos, una definición clara de en qué consiste el proyecto, una manera fácil de publicar la información del proyecto y una forma de compartir JARs entre varios proyectos.

El resultado es una herramienta que se puede utilizar para construir y gestionar cualquier proyecto basado en Java.

Maven hace uso del lenguaje XML para definir el archivo pom.xml (Fig. 54), el cual es el encargado de importar librerías de terceros al proyecto Spring.



Figura 14: Logo de XML

El objetivo principal de Maven es permitir que un desarrollador comprenda el estado completo de un proyecto en el menor tiempo posible. Para lograr este objetivo, Maven trata varios aspectos:

- Hacer que el proceso de construcción sea fácil
- Proporcionar un sistema de construcción uniforme
- Proporcionar información de proyecto de calidad
- Fomentar mejores prácticas de desarrollo



Figura 15: Logo de Maven

2.1.4 JSON Web Token (JWT)

JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autocontenida de transmitir información de forma segura entre partes como un objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente. Los JWT pueden ser firmados utilizando un secreto (con el algoritmo HMAC) o un par de claves pública/privada utilizando RSA o ECDSA.

Aunque los JWT pueden ser cifrados para proporcionar también secreto entre las partes, se centra en los *tokens* firmados. Los *tokens* firmados pueden verificar la integridad de las afirmaciones contenidas en él, mientras que los *tokens* cifrados ocultan esas afirmaciones a otras partes. Cuando los *tokens* son firmados utilizando pares de claves pública/privada, la firma también certifica que solo la parte que posee la clave privada es la que lo firmó.

En este proyecto es la base de la autenticación del usuario.



Figura 16: Logo de JWT

2.1.5 Swagger

Swagger trabaja junto a Spring Doc para permitir describir la estructura de las *APIs* para que las máquinas puedan leerlas. La capacidad de las *APIs* para describir su propia estructura es la raíz de Swagger.

Al leer la estructura de una *API*, puede construir automáticamente una documentación estética e interactiva, como la de la Fig. 18, que es la documentación del proyecto.

También puede generar automáticamente bibliotecas de cliente para una *API* en muchos lenguajes y explorar otras posibilidades como pruebas automatizadas. Swagger hace esto al pedirle a una *API* que devuelva un YAML o JSON que contenga una descripción detallada de toda la *API*.



Figura 17: Logo de Swagger



Figura 18: Interfaz gráfica de Swagger

2.1.6 Persistencia de datos

2.1.6.1 JPA

JPA es una especificación de Java que define una *API* para el mapeo objeto-relacional y el manejo de la persistencia en aplicaciones Java. Proporciona una interfaz de programación común y una serie de anotaciones para mapear clases Java a tablas de bases de datos y realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de manera independiente de la implementación subyacente.

Junto con Hibernate y MySQL, crean toda la estructura de persistencia de datos de los usuarios de la aplicación.



Figura 19: Logo de JPA

2.1.6.2 Hibernate

Hibernate es un *framework* de mapeo objeto-relacional (ORM) que simplifica la interacción entre una base de datos relacional y una aplicación Java. Como un marco de mapeo objeto-relacional (ORM), Hibernate se preocupa por la persistencia de datos en lo que se aplica a bases de datos relacionales (a través de JDBC).

Además de su propia API "nativa", Hibernate también es una implementación de la especificación Java Persistence API (JPA). Como tal, puede ser fácilmente utilizado en cualquier entorno que admita JPA, incluyendo aplicaciones Java SE, servidores de aplicaciones Java EE, contenedores Enterprise OSGi, etc.



Figura 20: Logo de Hibernate

2.1.6.3 MySQL

MySQL, el sistema de gestión de bases de datos SQL de código abierto más popular. Está desarrollado, distribuido y respaldado por Oracle Corporation.

Se usa para guardar la información de inicio de sesión de los usuarios.



Figura 21: Logo de MySQL

2.2 Microservicio Python - Backend

2.2.1 Flask

Flask es un marco de aplicación web escrito en Python. Fue desarrollado por Armin Ronacher, quien lideró un equipo de entusiastas internacionales de Python llamado Pocco. Flask se basa en la caja de herramientas WSGI Werkzeug y el motor de plantillas Jinja2. Ambos son proyectos de Pocco.

- **WSGI:** La Interfaz de Puerta de Enlace del Servidor Web (*Web Server Gateway Interface*, WSGI) se ha utilizado como estándar para el desarrollo de aplicaciones web en Python. WSGI es la especificación de una interfaz común entre los servidores web y las aplicaciones web.
- **Werkzeug:** Werkzeug es una *toolbox* WSGI que implementa objetos de solicitud, respuesta y funciones de utilidad. Esto permite construir un marco web sobre ella. El marco Flask utiliza Werkzeug como una de sus bases.
- **jinja2:** jinja2 es un motor de plantillas popular para Python. Un sistema de plantillas web combina una plantilla con una fuente de datos específica para renderizar una página web dinámica.

Flask se describe como un *microframework*. Está diseñado para mantener el núcleo de la aplicación simple y escalable. En lugar de una capa de abstracción para el soporte de base de datos, Flask admite extensiones para agregar tales capacidades a la aplicación.



Figura 23: Logo de Flask

2.2.2 Scikit-Learn

Scikit-learn es una librería de aprendizaje automático de código abierto que admite aprendizaje supervisado y no supervisado. También proporciona varias herramientas para el ajuste del modelo, preprocesamiento de datos, selección de modelos, evaluación de modelos y muchas otras utilidades.

Scikit-learn proporciona docenas de algoritmos y modelos de aprendizaje automático integrados, llamados estimadores. Cada estimador puede ajustarse a algunos datos utilizando su método de ajuste.



Figura 24: Logo de Scikit-Learn

2.2.3 Pandas

Pandas es una librería de Python utilizada para trabajar con conjuntos de datos. Tiene funciones para analizar, limpiar, explorar y manipular datos.

El nombre "Pandas" hace referencia tanto a "Panel Data" como a "*Python Data Analysis*" y fue creado por Wes McKinney en 2008.

Pandas permite analizar grandes conjuntos de datos y sacar conclusiones basadas en teorías estadísticas.

Pandas puede limpiar conjuntos de datos desordenados y hacerlos legibles y relevantes. Los datos relevantes son muy importantes en la ciencia de datos.



Figura 25: Logo de Pandas

2.3 Frontend

2.3.1 Bootstrap

Bootstrap es el marco CSS más popular para desarrollar sitios web responsivos y diseñados primero para dispositivos móviles. Está formado por:

2.3.1.1 HTML

HTML es el lenguaje de marcado principal de la World Wide Web. Originalmente, HTML fue diseñado principalmente como un lenguaje para describir semánticamente documentos científicos. Sin embargo, su diseño general ha permitido que se adapte, a lo largo de los años siguientes, para describir una serie de otros tipos de documentos e incluso aplicaciones.



Figura 26: Logo de HTML5

2.3.1.2 CSS3

CSS es el lenguaje que se usa para dar estilo a una página web. CSS significa Hojas de Estilo en Cascada.

Describe cómo se deben mostrar los elementos HTML en la pantalla.

Se utiliza para añadir estética a la interfaz gráfica del usuario.



Figura 27: Logo de CSS3

2.3.1.3 JavaScript

JavaScript (JS) es un lenguaje de programación ligero interpretado (o compilado en tiempo real) con funciones de primera clase. Si bien es más conocido como el lenguaje de *scripting* para páginas web, muchos entornos no relacionados con el navegador también lo utilizan, como Node.js, Apache CouchDB y Adobe Acrobat. JavaScript es un lenguaje dinámico basado en prototipos, de múltiples paradigmas, de un solo hilo, que admite estilos de programación orientados a objetos, imperativos y declarativos (por ejemplo, programación funcional).

JavaScript



Figura 28: Logo de JavaScript

JavaScript hace uso de la librería jQuery:

El propósito de jQuery es hacer mucho más fácil usar JavaScript en un sitio web. jQuery toma muchas tareas comunes que requieren muchas líneas de código JavaScript para realizarlas, y las envuelve en métodos que puedes llamar con una sola línea de código.

jQuery también simplifica muchas de las cosas complicadas de JavaScript, como las llamadas AJAX y la manipulación del DOM.

La biblioteca jQuery contiene las siguientes características:

- Manipulación HTML/DOM
- Manipulación de CSS
- Métodos de eventos HTML
- Efectos y animaciones
- AJAX

Como bien se menciona, JavaScript y jQuery hacen uso de Ajax:

AJAX, que significa Asynchronous JavaScript And XML, no es un lenguaje de programación en sí mismo. Más bien, es una técnica que hace uso de una combinación de componentes:

- El objeto XMLHttpRequest integrado en los navegadores (para solicitar datos desde un servidor web).
- JavaScript y el Modelo de Objetos del Documento (DOM) de HTML (para mostrar o utilizar los datos).

La principal ventaja de AJAX radica en su capacidad para actualizar páginas web de forma asíncrona, intercambiando datos con un servidor web en segundo plano. Esto significa que es posible actualizar partes específicas de una página web sin tener que volver a cargar toda la página.



Figura 29: Logo de AJAX

3 HERRAMIENTAS UTILIZADAS

Para el desarrollo y despliegue del proyecto se han utilizado diversas herramientas:

3.1 Google Chrome

El navegador Google Chrome ha sido elegido para mostrar la aplicación debido a su excelente rendimiento con el *frontend*. Su capacidad para interpretar y renderizar de manera eficiente el diseño y las funcionalidades de la interfaz gráfica de la aplicación lo convierten en la opción ideal para ofrecer una experiencia óptima a los usuarios.



Figura 30: Logo de Google Chrome

3.2 Visual Studio Code

Visual Studio Code es un editor de código fuente que ofrece un amplio soporte para múltiples lenguajes de programación. Con características como resaltado de sintaxis, coincidencia de corchetes, auto-indentación y selección de texto, facilita la tarea de escribir y editar código.

VSCoode cuenta con atajos de teclado intuitivos y opciones de personalización para adaptarse a las preferencias de cada usuario. Para proyectos más complejos, Visual Studio Code incluye herramientas integradas como IntelliSense para completado de código, entendimiento semántico del código y refactorización. También se integra con herramientas de compilación y control de versiones como Git, lo que permite realizar tareas comunes de desarrollo de manera más eficiente y trabajar con control de origen directamente desde el editor.

La ventaja que tiene VSCoode es que tiene disponibles una gran cantidad de extensiones. Tal y como se ve en la Fig. 32, estas son accesibles en todo momento gracias a una barra vertical que está fija a la izquierda y se puede ver desde cualquier parte de la aplicación.

Algunas de estas extensiones han sido determinantes para la realización del proyecto:

- **Database:** La extensión Database, ha servido como interfaz gráfica para manejar la base de datos de la aplicación, pudiendo hacer todo tipo de operaciones sobre la BBDD sin necesidad de utilizar comandos.

- **Spring Boot Dashboard:** Sirve para controlar en tiempo real todos los elementos que componen el framework de Spring: las aplicaciones (en este caso mytrAIning), los Beans y los *endpoints*. También sirve para desplegar la aplicación.
- **Python:** La extensión oficial de Python ha sido utilizada para crear y manejar los distintos entornos y librerías que se han usado para el desarrollo del microservicio Python.



Figura 31: Logo de Visual Studio Code

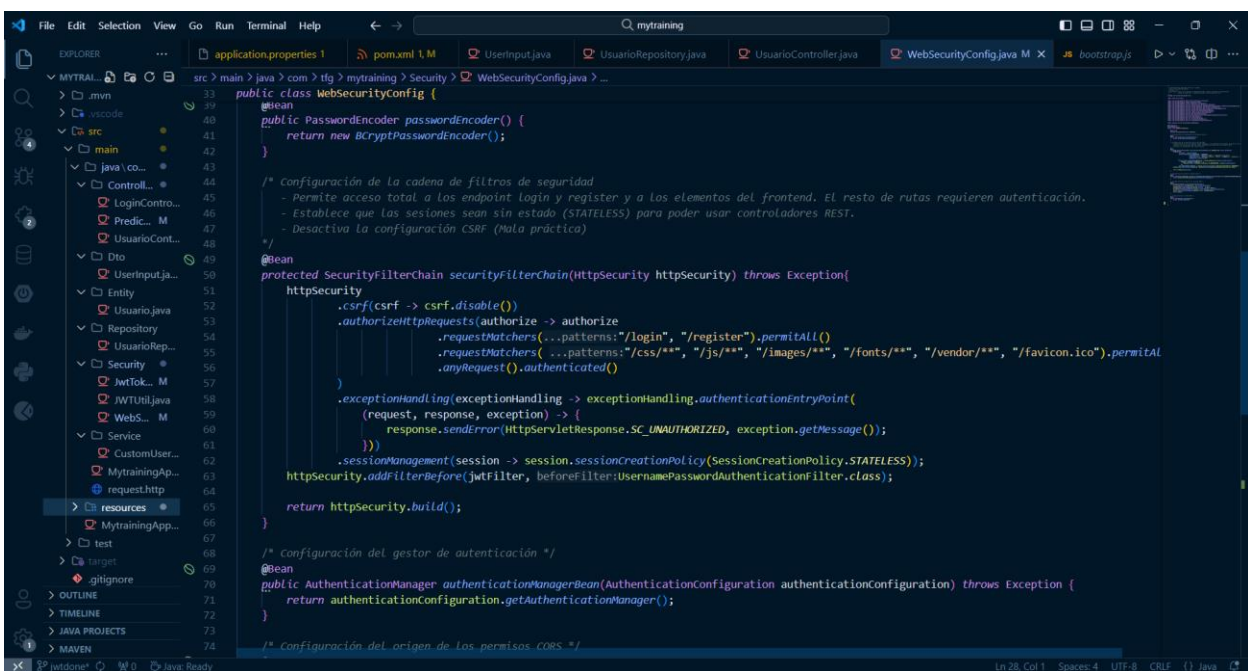


Figura 32: Interfaz gráfica de VSCode

3.3 Docker

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones. Con Docker, se puede gestionar la infraestructura de la misma manera que se gestionan las aplicaciones. Al aprovechar las metodologías de Docker para enviar, probar e implementar código, se puede reducir significativamente el tiempo entre escribir el código y ejecutarlo en producción.

Docker proporciona la capacidad de empaquetar y ejecutar una aplicación en un entorno ligeramente aislado llamado contenedor. El aislamiento y la seguridad permiten ejecutar muchos contenedores simultáneamente en un host dado. Los contenedores son ligeros y contienen todo lo necesario para ejecutar la aplicación, por lo que no se necesita depender de lo que está instalado en el host. Se pueden compartir contenedores mientras se trabaja y asegurarse de que todos con quienes se comparte obtengan el mismo contenedor que funciona de la misma manera.

Gracias a su aplicación de escritorio Docker Desktop (Fig. 34), podemos gestionar todas las imágenes y contenedores, así como desplegarlos, pausarlos o detenerlos, entre otras características.

Docker proporciona herramientas y una plataforma para gestionar el ciclo de vida de los contenedores:

- Se desarrolla la aplicación y sus componentes de soporte utilizando contenedores.
- El contenedor se convierte en la unidad para distribuir y probar la aplicación.
- Cuando se está listo, se implementa la aplicación en el entorno de producción, ya sea como un contenedor o un servicio orquestado. Esto funciona de la misma manera, independientemente de si el entorno de producción es un centro de datos local, un proveedor de servicios en la nube o una combinación de ambos.

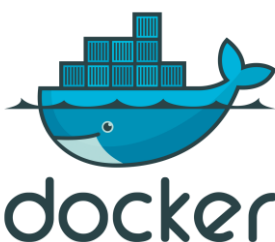


Figura 33: Logo de Docker

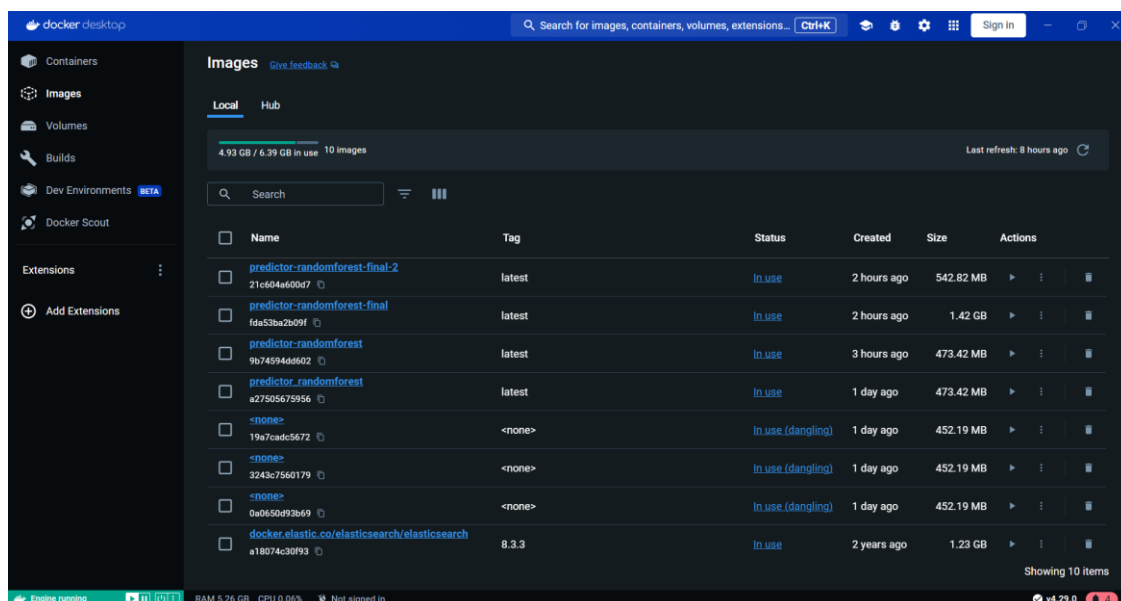


Figura 34: Interfaz gráfica de Docker Desktop

3.4 Postman

Postman es una herramienta de desarrollo de *API* (Interfaz de Programación de Aplicaciones) que ayuda a construir, probar y modificar *APIs*. Casi cualquier funcionalidad que pueda necesitar cualquier desarrollador está encapsulada en esta herramienta. Es utilizada por más de 5 millones de desarrolladores cada mes para hacer que su desarrollo de *API* sea fácil y simple. Como se ve en la Fig. 36, Postman tiene una interfaz simple y con la capacidad de realizar varios tipos de solicitudes HTTP (GET, POST, PUT, PATCH), guardar entornos para su uso posterior, y convertir la *API* a código para varios lenguajes (como JavaScript y Python).

En resumen, Postman transforma el panorama del desarrollo de *API* al combinar versatilidad, flexibilidad, simplicidad y eficiencia. Ya sea interactuando con *APIs*, manejando autenticación, organizando pruebas o generando documentación, Postman ofrece una suite completa de herramientas diseñadas para satisfacer las demandas del desarrollo de software moderno.



Figura 35: Logo Postman

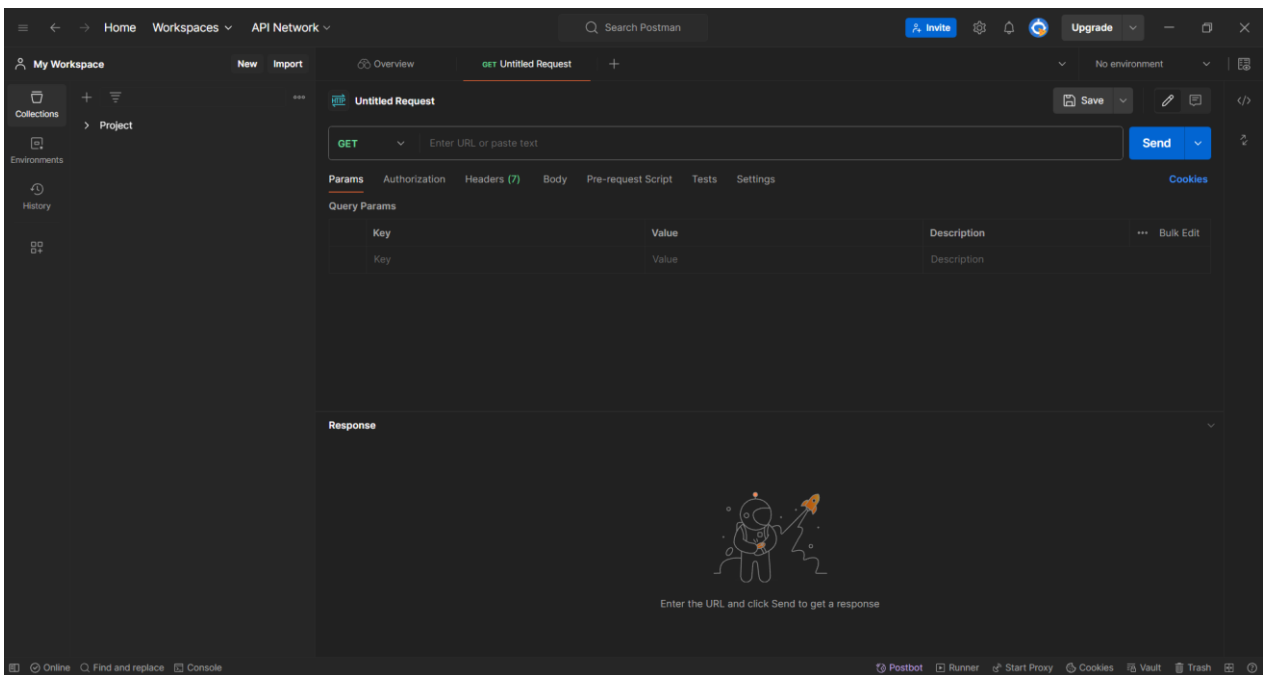


Figura 36: Interfaz gráfica de Postman

3.5 XAMPP Control Panel

XAMPP, que significa Cross-Platform, Apache, MySQL, PHP y Perl, es una plataforma gratuita que permite a los desarrolladores probar su código localmente en sus propios ordenadores. Esta plataforma proporciona la experiencia de tener tu propio mini servidor web en casa, compatible tanto con entornos Windows (WAMP) como Linux (LAMP).

Como se puede observar en la Fig. 38, contiene los módulos de Apache, MySQL, FileZilla, Mercury y Tomcat.

Con XAMPP también viene incluido el gestor de bases de datos phpMyAdmin, cuya interfaz se puede ver en la Fig. 39, con el que se ha creado la base de datos de la aplicación.

En el contexto de mytrAIning, ha servido para desplegar MySQL, Apache y Tomcat.



Figura 37: Logo de XAMPP

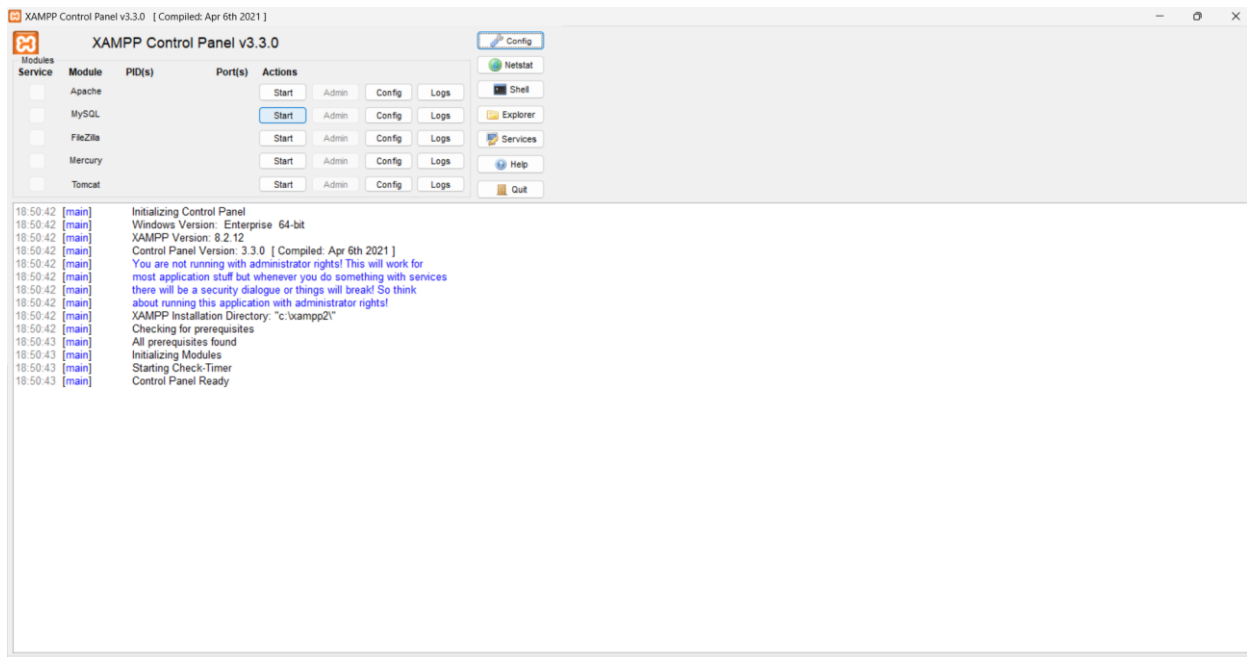


Figura 38: Interfaz gráfica de XAMPP

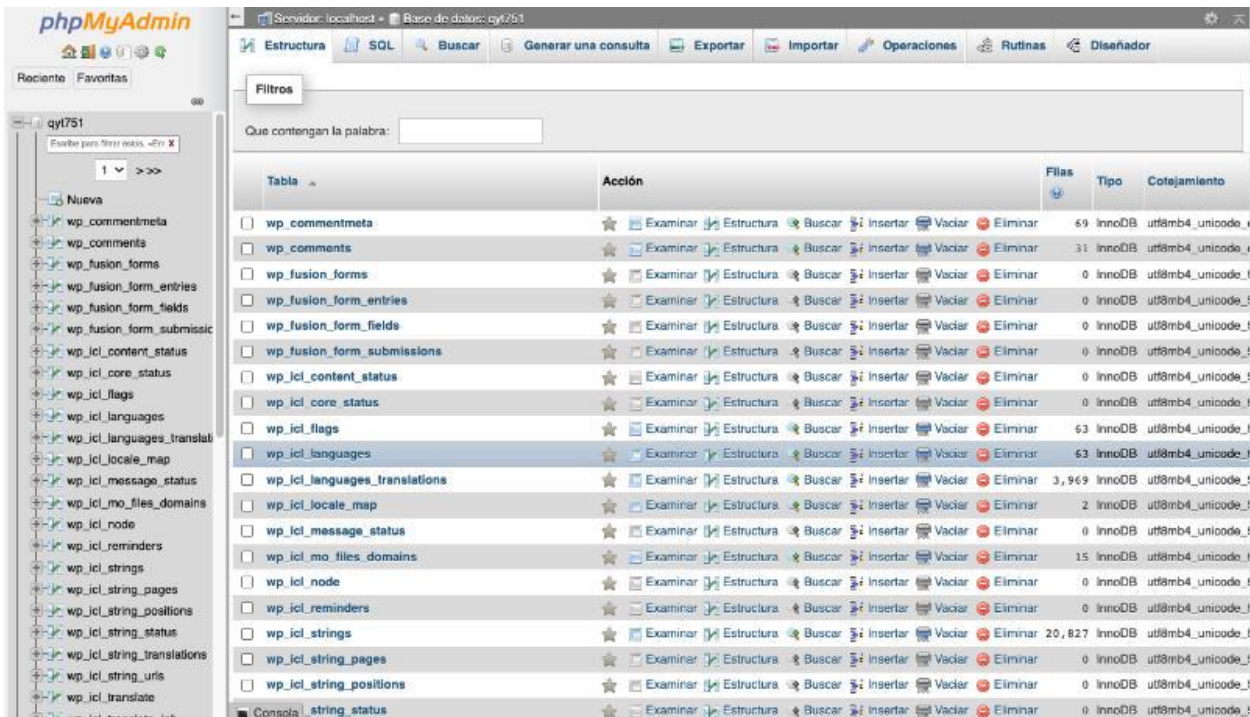


Figura 39: Interfaz gráfica de phpMyAdmin

3.6 GitHub

GitHub es una plataforma de desarrollo de software en línea. Se utiliza para almacenar, rastrear y colaborar en proyectos de software.

En la Fig. 41 podemos ver un ejemplo de un repositorio de GitHub, donde aparece la estructura de directorios, así como información relevante sobre el desarrollo del mismo.

Facilita a los desarrolladores compartir archivos de código y colaborar con otros desarrolladores en proyectos de código abierto. GitHub también funciona como una red social donde los desarrolladores pueden conectarse abiertamente, colaborar y presentar su trabajo.



Figura 40: Logo de GitHub

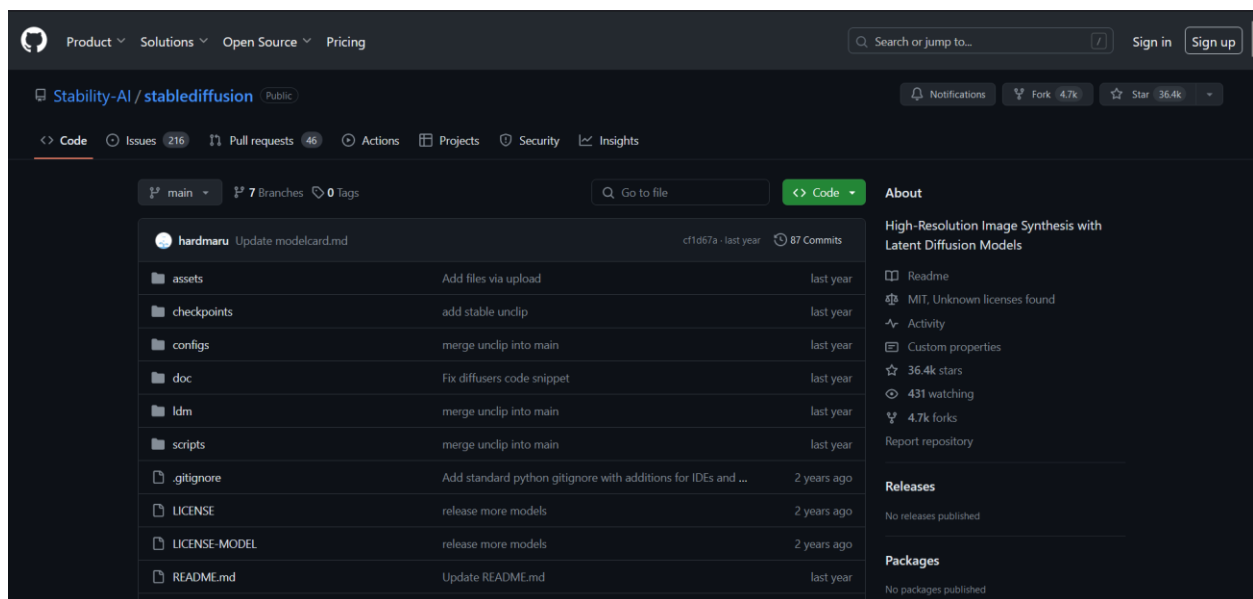


Figura 41: Página de repositorio de GitHub

4 DESARROLLO DEL PROYECTO

En esta sección se detalla el proceso seguido para la implementación de mytrAIning, describiendo paso a paso qué se ha implementado y cómo se ha hecho.

El proyecto se divide en dos principales componentes: el componente Java, usando el *framework* Spring, y el microservicio Python, diseñado para la predicción del peso que se debe aplicar a cada ejercicio.

También hay una sección dedicada al *frontend* donde se explica cómo se ha realizado el manejo y envío de datos en formato JSON desde los formularios a los *endpoints* REST y viceversa.

4.1 Componente Java Spring

El proyecto ha sido desarrollado con el patrón de diseño Modelo-Vista-Controlador. Esta arquitectura consiste en separar la aplicación en tres componentes lógicos: Modelo, Vista y Controlador.

Cada componente está diseñado para manejar aspectos específicos del desarrollo de una aplicación, aislando la lógica de negocio de la capa de presentación.

Tradicionalmente, se usaba para interfaces gráficas de usuario (GUIs) de escritorio. Hoy en día, MVC es uno de los modelos de desarrollo web más utilizados en la industria para crear proyectos escalables y extensibles.

Su estructura es la siguiente:

- **Modelo:** Conformar toda la lógica relacionada con los datos con los que trabaja el usuario. Esto puede representar tanto los datos que se transfieren entre los componentes Vista y Controlador como cualquier otro dato relacionado con la lógica de negocio.
- **Vista:** Se utiliza para toda la lógica de interfaz de usuario de la aplicación. Genera una interfaz para los usuarios finales. Las vistas se crean a partir de los datos recopilados por el componente Modelo, pero estos datos no se toman directamente, sino a través del controlador.
- **Controlador:** Hace de intermediario entre las vistas y el modelo. Procesa toda la lógica de negocio y las solicitudes entrantes, manipula los datos utilizando el componente Modelo e interactúa con la Vista para renderizar la salida final.

En el proyecto, el modelo está formado por los directorios Dto, Entity, Repository y Service que son los encargados de realizar la persistencia de datos con la BBDD; los controladores, que están claramente definidos en la estructura de directorios, y la vista, que comprende toda la estructura del *frontend*.

Por otro lado, tenemos la configuración de seguridad realizada con Spring Security, que permite configurar distintos parámetros de la seguridad de la aplicación web, así como controlar el acceso a las diferentes rutas.

La jerarquía de directorios en el componente Java Spring se ha organizado de la forma en la que se ve en la Fig. 42.

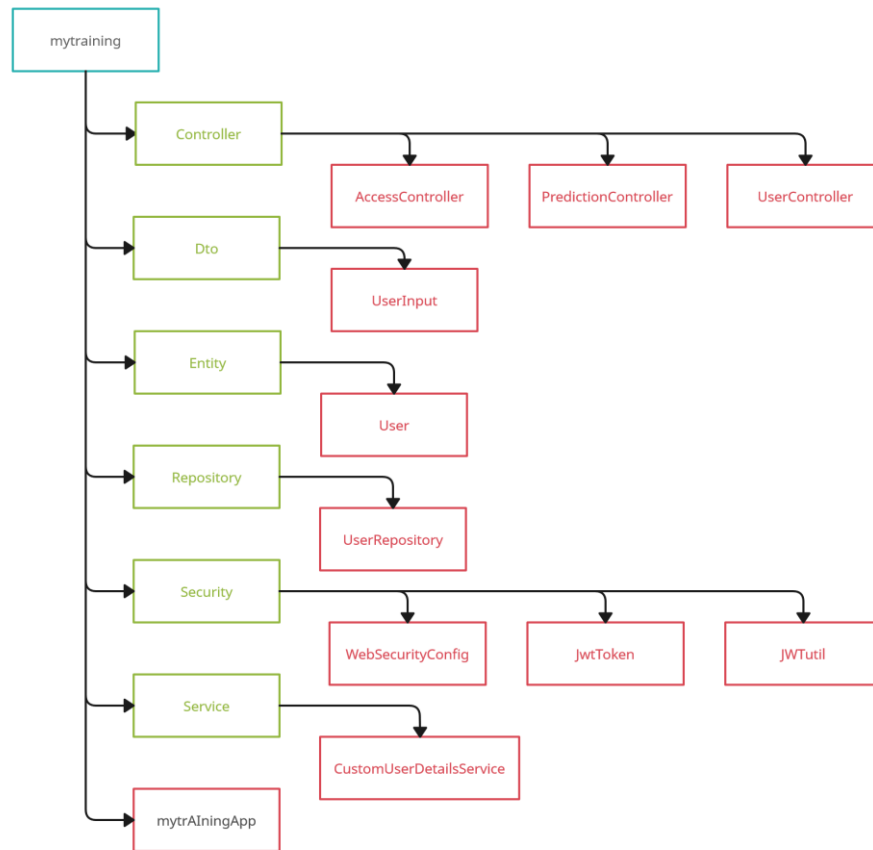


Figura 42: Jerarquía de directorios del componente Java Spring

access-controller

- GET /register Devuelve la página de registro de usuarios.
- GET /login Devuelve la página de inicio de sesión.
- GET /predictor Devuelve la página de predicción donde los usuarios pueden realizar predicciones.
- GET /home Devuelve la página principal del sitio web.

user-controller

- POST /register Registra un nuevo usuario con contraseña cifrada y lo guarda en la base de datos.
- POST /logout Desconecta al usuario eliminando la cookie JWT y limpiando el contexto de seguridad.
- POST /login Autentica al usuario, genera un token JWT y lo almacena en una cookie.

prediction-controller

- POST /prediction Convierte la entrada del usuario a JSON y envía una solicitud POST al microservicio Python, que devuelve una rutina personalizada.

Schemas

- User >
- Userinput >

Figura 43: Documentación de los *endpoint* y las entidades de Swagger

4.1.1 Controladores

Tal y como aparece en la Fig.43, existen tres controladores que se encargan de manejar las distintas rutas que acceden a todos los REST *endpoint* de la aplicación:

4.1.1.1 AccessController

El AccessController se encarga de manejar las solicitudes web hacia las diferentes vistas de la aplicación.

Está etiquetado con la anotación `@Controller` ya que realiza la función clásica de un controlador, que es devolver vistas.

Como se observa en la Fig. 44, el fichero tiene cuatro *endpoints* principales:

- **/login:** Devuelve la página de inicio de sesión ("login.html"). Este *endpoint* se activa al iniciar la aplicación, ya que es la ruta por defecto.
- **/home:** Este *endpoint* redirige a la página principal del sitio ("index.html") y se accede a través de una solicitud GET. Es la vista a la que se llega después de iniciar sesión.
- **/predictor:** Devuelve la página de predicción ("predictor.html"), donde se rellena el formulario con la rutina seleccionada y los datos del usuario para obtener los ejercicios con sus respectivos pesos personalizados.
- **/register:** Devuelve la página de registro ("register.html") accesible desde el inicio de sesión. Este *endpoint* facilita el acceso a la vista donde los usuarios pueden crear una nueva cuenta.

```
@Controller
public class AccessController {

    @Operation(summary = "Devuelve la página de inicio de sesión.")
    @ApiResponse(responseCode = "200", description = "La página de inicio de sesión se carga correctamente.")
    @GetMapping("/login")
    public String loginPage() {
        return "login";
    }

    @Operation(summary = "Devuelve la página principal del sitio web.")
    @ApiResponse(responseCode = "200", description = "La página principal se carga correctamente.")
    @GetMapping("/home")
    public String menuPage() {
        return "index";
    }

    @Operation(summary = "Devuelve la página de predicción donde los usuarios pueden realizar predicciones.")
    @ApiResponse(responseCode = "200", description = "La página de predicción se carga correctamente.")
    @GetMapping("/predictor")
    public String predictorPage() {
        return "predictor";
    }

    @Operation(summary = "Devuelve la página de registro de usuarios.")
    @ApiResponse(responseCode = "200", description = "La página de registro se carga correctamente.")
    @GetMapping("/register")
    public String registerPage() {
        return "register";
    }
}
```

Figura 44: AccessController.java

4.1.1.2 UserController

UserController es el controlador REST que realiza las operaciones relacionadas con los usuarios de la aplicación: el registro, inicio de sesión y el cierre de sesión. Para documentar las respuestas esperadas en cada llamada, se utiliza Spring Doc junto a Swagger, detallando las posibles respuestas HTTP que cada *endpoint* puede emitir.

Como vemos en la Fig. 45, estos son los *endpoints* definidos:

- **/register:** Permite el registro de nuevos usuarios. Verifica si el usuario ya está registrado y, en caso negativo, cifra su contraseña con BcryptPasswordEncoder y lo guarda en la base de datos.
- **/login:** Administra el inicio de sesión comprobando que existan las credenciales del usuario en la BBDD. Si son correctas, genera un token JWT y lo guarda en una cookie en el navegador.
- **/logout:** Maneja el cierre de sesión eliminando la cookie JWT y limpiando el contexto de seguridad. Una vez borrado el *token*, la aplicación redirige al usuario a la vista de login.

```
@PostMapping("/register")
public ResponseEntity<User> registrarUsuario(@RequestBody User nuevoUsuario) {
    if (usuarioRepository.existsById(nuevoUsuario.getEmail())) {
        return ResponseEntity.badRequest().build();
    }
    nuevoUsuario.setPassword(passwordEncoder.encode(nuevoUsuario.getPassword()));
    User usuarioGuardado = usuarioRepository.save(nuevoUsuario);
    return ResponseEntity.ok(usuarioGuardado);
}

@Operation(summary = "Autentica al usuario, genera un token JWT y lo almacena en una cookie.")
@ApiResponse(responseCode = "200", description = "Usuario autenticado y token generado correctamente")
@ApiResponse(responseCode = "400", description = "Credenciales incorrectas", content = @Content(mediaType = "application/json"))
@PostMapping("/login")
public ResponseEntity<> Login(@RequestBody Map<String, String> requestMap, HttpServletResponse response) {
    try {
        String email = requestMap.get("email");
        String password = requestMap.get("password");
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(email, password)
        );

        if (authentication.isAuthenticated()) {
            UserDetails userDetails = (UserDetails) authentication.getPrincipal();
            String token = jwtUtil.generateToken(userDetails.getUsername(), userDetails.getPassword());
            Cookie jwtCookie = new Cookie("JWT_TOKEN", token);
            jwtCookie.setHttpOnly(true);
            jwtCookie.setPath("/");
            response.addCookie(jwtCookie);

            return ResponseEntity.ok().build();
        }
    } catch (Exception exception) {
    }

    return new ResponseEntity<String>{
        "{ \"mensaje\": \"\" + \"Credenciales incorrectas\" + \"\" }",
        HttpStatus.BAD_REQUEST);
}

@Operation(summary = "Desconecta al usuario eliminando la cookie JWT y limpiando el contexto de seguridad.")
@ApiResponse(responseCode = "302", description = "Redirección al login después del logout")
@ApiResponse(responseCode = "500", description = "Error interno del servidor durante el logout", content = @Content(mediaType = "application/json"))
@PostMapping("/logout")
public ResponseEntity<> Logout(HttpServletRequest request, HttpServletResponse response) {
    try {
        Cookie jwtCookie = WebUtils.getCookie(request, "JWT_TOKEN");
        if (jwtCookie != null) {
            jwtCookie.setValue(null);
            jwtCookie.setHttpOnly(true);
            jwtCookie.setPath("/");
            jwtCookie.setMaxAge(0);
            jwtCookie.setSecure(true);
            response.addCookie(jwtCookie);
        }

        SecurityContextHolder.clearContext();
        return ResponseEntity.status(HttpStatus.FOUND).header("Location", "/login").build();
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error durante el logout");
    }
}
```

Figura 45: UserController.java

4.1.1.3 PredictionController

Es el controlador que hace de puente entre el microservicio Python y el componente Java Spring.

Si se observa la Fig. 46, se puede ver que el único *endpoint* es `/prediction`. Este es llamado cuando se envía el formulario de selección de rutina y características físicas del usuario.

Cuando se pulsa el botón de continuar desde el apartado de predictor en la aplicación, se realiza una petición POST al controlador. En este POST se envían los datos del formulario en formato JSON.

Una vez realizada esta petición, se realizan las siguientes acciones dentro del controlador:

- El controlador, apoyándose en el DTO `UserInput`, convierte los datos recibidos del formulario en un formato aceptado por el modelo entrenado de Machine Learning.
- Una vez preparados los datos, se prepara una entidad HTTP que encapsula tanto el JSON con los datos como las cabeceras necesarias.
- Luego con `RestTemplate` se envía la solicitud POST al microservicio Python. `RestTemplate` es una clase de Spring que permite realizar peticiones HTTP a un servicio web y procesar la información recibida. Es necesario para recibir el resultado del modelo Random Forest
- Si la respuesta es exitosa, se devuelven en formato JSON los ejercicios de la rutina seleccionada y el peso propuesto para cada uno de ellos.

```
@PostMapping("/prediction")
public ResponseEntity<?> predictExerciseWeight(@RequestBody UserInput userInput) {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    ObjectMapper objectMapper = new ObjectMapper();
    String userJson;

    try {
        userJson = objectMapper.writeValueAsString(userInput);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al convertir UserInput a JSON");
    }

    HttpEntity<String> request = new HttpEntity<>(userJson, headers);
    String pythonServiceUrl = "http://192.168.1.17:5000/predict";
    RestTemplate restTemplate = new RestTemplate();

    try {
        ResponseEntity<String> response = restTemplate.postForEntity(pythonServiceUrl, request, String.class);
        return ResponseEntity.ok(response.getBody());
    } catch (HttpClientErrorException e) {
        e.printStackTrace();
        return ResponseEntity.status(e.getStatusCode()).body(e.getResponseBodyAsString());
    }
}
```

Figura 46: PredictionController.java

4.1.2 Seguridad

4.1.2.1 WebSecurityConfig

Una de las clases que garantiza la seguridad de la aplicación es **WebSecurityConfig**. Es una clase estándar de Spring Security que, con métodos predefinidos, permite configurar distintos aspectos relativos a la seguridad como pueden ser: Autenticación, accesos a rutas o configuración de distintos parámetros como la protección frente a ataques CSRF (Cross Site Request Forgery).

Para proteger la aplicación, se utilizan varios Beans, que son componentes clave del *framework Spring*. La principal particularidad de los Beans es que se pueden reutilizar en distintas partes del código.

En este fichero existen los siguientes:

- **Codificador de Contraseñas Bcrypt:** Este Bean configura el PasswordEncoder usando Bcrypt, un algoritmo hash para contraseñas.
- **Gestor de Autenticación:** Hay un Bean para AuthenticationManager que es el que maneja la autenticación. Este gestor procesa las credenciales de autenticación proporcionadas y valida si son correctas.
- **CorsConfigurationSource:** El Bean CorsConfigurationSource establece las políticas de CORS, permitiendo solicitudes de cualquier origen y especificando los métodos HTTP permitidos.
- **RestTemplate:** El Bean RestTemplate facilita la comunicación con otros servicios a través de HTTP.
- **SecurityFilterChain:** Este Bean, cuya implementación se ve en la Fig. 47, es el núcleo de la configuración de la estructura Spring Security. Realiza las siguientes acciones:
 - **CSRF:** La protección contra CSRF está activada por defecto en Spring Security. Aunque es una mala práctica desactivarla, se ha desactivado debido a que se utiliza el JWT. JWT no maneja sesiones, sino que simplemente tiene un tiempo de vida, por lo que desactivar esta característica simplifica bastante la comunicación con el cliente.
 - **Control de Accesos:** Permite acceder a todos los usuarios a las rutas de /login, /register y a los recursos gráficos del frontend, pero exige un JWT para navegar por el resto de *endpoints*.
 - **Política de Sesiones:** Debido a que se están utilizando servicios REST, es necesario establecer la política de creación de sesiones a STATELESS, evitando así que el servidor mantenga el estado de la sesión, ya que al tener varios servicios sería complicado de manejar el uso de estos con sesiones activas.

```
@Bean
protected SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception{
    httpSecurity
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(authorize -> authorize
            .requestMatchers("/login", "/register").permitAll()
            .requestMatchers("/css/**", "/js/**", "/images/**", "/fonts/**", "/vendor/**", "/favicon.ico").permitAll()
            .anyRequest().authenticated()
        )
        .exceptionHandling(exceptionHandling -> exceptionHandling.authenticationEntryPoint(
            (request, response, exception) -> {
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, exception.getMessage());
            }
        ))
        .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
    httpSecurity.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

    return httpSecurity.build();
}
```

Figura 47: Bean securityFilterChain de WebSecurityConfig.java

Los otros dos ficheros que componen el directorio de Seguridad están relacionados con la creación y manipulación del JSON Web Token, el cual permite a los usuarios registrados acceder a recursos protegidos a los que los usuarios no autenticados no pueden acceder.

4.1.2.2 JwtUtil

La clase `JWTUtil` se compone de una serie de métodos necesarios para la creación y el manejo del JWT.

La implementación de algunos de estos métodos se puede observar en la Fig. 48.

Sus funciones son las siguientes:

- **Generación de JWT:** La función `generateToken` crea un nuevo token para un usuario basado en su email y contraseña. Para ello, llama a la función `createToken` que establece distintos parámetros en el token como el tiempo de expiración.
- **Extracción y Validación de Reclamaciones:** `extractClaims` y `extractAllClaims` permiten obtener datos de un token existente, como el nombre de usuario o la fecha de expiración. Estas funciones la utilizamos para eliminar el token al cerrar sesión.
- **Verificación de Expiración:** `isTokenExpired` comprueba si la fecha de expiración del token es anterior a la fecha actual.
- **Validación del Token:** `validateToken` compara el nombre de usuario extraído del token con el nombre de usuario que ha iniciado sesión.
- **Autenticación a partir del Token:** El método `getAuthentication` genera un objeto `Authentication` utilizando los detalles del usuario obtenidos a partir del token pasado por parámetro. Este objeto se utiliza con Spring Security para establecer el contexto de seguridad del usuario en la sesión.

```

// Verifica si el token JWT ha expirado
public Boolean isTokenExpired(String token){
    return extractExpiration(token).before(new Date());
}

// Genera un token JWT para el email y la contraseña proporcionados
public String generateToken(String email, String password){
    Map<String, Object> claims = new HashMap<>();
    claims.put("password", password);
    return createToken(claims, email);
}

// Crea un token JWT con las reclamaciones y el sujeto proporcionados
private String createToken(Map<String, Object> claims, String subject){
    return Jwts.builder()
        .setClaims(claims)
        .setSubject(subject)
        .setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() + 1000*60*60*10)) // 10 horas de duración
        .signWith(SignatureAlgorithm.HS256, secret).compact();
}

// Valida el token JWT frente a los detalles del usuario
public Boolean validateToken(String token, UserDetails userDetails){
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
}

// Obtiene la autenticación a partir del token JWT
public Authentication getAuthentication(String token) {
    String username = extractUsername(token);
    UserDetails userDetails = customUserDetailsService.loadUserByUsername(username);
    return new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
}

```

Figura 48: Fragmento de código de `JwtUtil.java`

4.1.2.3 JwtCookieFilter

Esta clase es esencial para el manejo de las *cookies* que almacenan los JWT. Este método se ejecuta cada vez que se realiza una solicitud HTTP, consiguiendo así mantener la seguridad de la aplicación en todo momento.

Está compuesta por dos métodos:

- **doFilterInternal:** La implementación de este método se puede encontrar en la Fig. 49.

Realiza las siguientes acciones:

- **Extracción del Token:** Primero, extrae la *cookie* “JWT_Token” de la solicitud. JWT_Token es el nombre definido para la *cookie* que se genera al iniciar sesión en la aplicación.
 - **Validación y autenticación:** Si el *token* es válido y no ha expirado, se obtiene la autenticación del *token*. En caso de éxito, se establece el contexto de seguridad de Spring Security con el objeto Authentication obtenido, permitiendo al usuario acceder a las rutas protegidas. En caso de que el *token* no sea válido o haya expirado, se elimina el *token* y se redirige al usuario al inicio de sesión.
 - **Cierre de sesión:** Si la ruta solicitada es /login?logout (contiene una consulta de cierre de sesión) se limpia la autenticación y se redirige al usuario a la página de inicio de sesión, eliminando la *cookie* JWT. Esto se tiene que hacer desde el *backend*, ya que desde el *frontend* no se puede gestionar la eliminación del *token* para así evitar brechas de seguridad en la aplicación, por lo que la única forma que hay de cerrar sesión es mandar el *token* como parámetro desde el *doFilterInternal* y poner su fecha de expiración a 0.
 - **Cadena de filtros:** Una vez realizadas todas las acciones, el filtro pasa la solicitud y la respuesta a la cadena de filtros de Spring Security para que el resto de las opciones de seguridad se ejecuten.
- **clearAuthenticationAndRedirect:** Este método es llamado desde *doFilterInternal* para ejecutar el cierre de sesión: se elimina el token, se limpia el contexto de seguridad de Spring Security y se redirecciona al usuario a la página de inicio de sesión.

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

    Cookie jwtCookie = WebUtils.getCookie(request, "JWT_TOKEN");

    String requestURI = request.getRequestURI();
    String queryString = request.getQueryString();

    if (requestURI.equals("/login") && queryString != null && queryString.equals("logout")) {
        clearAuthenticationAndRedirect(response, jwtCookie);
        return;
    }

    if (jwtCookie != null) {
        String token = jwtCookie.getValue();
        try {
            if (token != null && !jwtUtil.isTokenExpired(token)) {
                Authentication authentication = jwtUtil.getAuthentication(token);
                if (authentication != null) {
                    SecurityContextHolder.getContext().setAuthentication(authentication);
                }
            } else {
                throw new ServletException("Token expirado");
            }
        } catch (Exception e) {
            clearAuthenticationAndRedirect(response, jwtCookie);
            return;
        }
    }

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

Figura 49: Método doFilterInternal de JWTCookieFilter

4.1.3 Entidades

4.1.3.1 User

La entidad **User** representa a los usuarios de la aplicación.

Esta aplicación está pensada para seguir un modelo de suscripción, creando la necesidad de autenticar a los usuarios para acceder a la predicción de pesos.

La seguridad de la App se maneja a través de *tokens* JWT, por lo que es necesario que se haya iniciado sesión para que se le permita al usuario navegar por las diferentes vistas y funcionalidades disponibles.

En la Fig. 50 se puede observar que la entidad consta de 4 campos:

- **Email:** Es el identificador único del usuario en el sistema. Se usa para el proceso de inicio de sesión y para garantizar que cada cuenta de usuario sea única.
- **Nombre:** Este campo almacena el nombre del usuario.
- **Contraseña:** Contiene la contraseña del usuario, la cual se almacena encriptada mediante el codificador Bcrypt.
- **Rol:** Define el rol del usuario. Este campo serviría para la gestión de accesos mediante JWT, permitiendo diferenciar entre distintos niveles de acceso dentro de la aplicación. Debido a que no existe un usuario administrador no está en uso, pero se implementa para añadirlo en futuras versiones.

```
@ToString
@Setter
@EqualsAndHashCode
@Getter
@Entity

@Schema(description = "Entidad que representa a un usuario en el sistema.")
public class User {

    @Id
    @Schema(description = "Correo electrónico del usuario, usado como identificador único.", example = "usuario@example.com")
    private String email;

    @Schema(description = "Nombre del usuario.", example = "Juan Pérez")
    private String nombre;

    @Schema(description = "Contraseña cifrada del usuario.", example = "*****")
    private String password;

    @Schema(description = "Rol del usuario dentro del sistema. Se utiliza para manejar el token JWT.", example = "USER")
    private String rol;

    /* Constructor para inicializar un objeto Usuario con los atributos proporcionados. */
    public User(String email, String nombre, String password, String rol) {
        this.email = email;
        this.nombre = nombre;
        this.password = password;
        this.rol = rol;
    }

    /* Constructor vacío requerido por JPA. */
    public User() {
        super();
    }
}
```

Figura 50: User.java

4.1.4 Dto

4.1.4.1 UserInput

El DTO (Data Transfer Object) **UserInput** es la clase que encapsula los datos que serán enviados al microservicio Python.

Cabe destacar en la Fig. 51 el uso de la anotación `@JsonProperty("value")`, que sirve para hacer coincidir los nombres de los campos con los que el modelo *Random Forest* espera recibir.

Tiene los siguientes campos:

- **Gender**
- **Age**
- **Weight**
- **FAF (Frecuencia de Actividad Física):**
- **rutinaID**
- **Valoración:** Este campo se fija en 2, ya que sería la valoración ideal siguiendo el formato del conjunto de datos que se ha usado para entrenar el modelo de *Machine Learning*.

```
public class UserInput {

    @Schema(description = "Género del usuario, representado como un entero.", example = "1")
    @JsonProperty("Gender")
    private int Gender;

    @Schema(description = "Edad del usuario.", example = "30")
    @JsonProperty("Age")
    private int Age;

    @Schema(description = "Peso del usuario en kilogramos.", example = "70.5")
    @JsonProperty("Weight")
    private double Weight;

    @Schema(description = "Frecuencia de Actividad Física del usuario.", example = "1.2")
    @JsonProperty("FAF")
    private double FAF;

    @Schema(description = "Identificador de la rutina de ejercicios seleccionada por el usuario.", example = "Rutina001")
    @JsonProperty("rutinaID")
    private String rutinaID;

    @Schema(description = "Valoración predeterminada usada en el cálculo de la rutina.", example = "2")
    @JsonProperty("Valoracion")
    private double Valoracion = 2;

    public UserInput() {}

    public UserInput(int Gender, int Age, double Weight, double FAF, String rutinaID) {
        this.Gender = Gender;
        this.Age = Age;
        this.Weight = Weight;
        this.FAF = FAF;
        this.rutinaID = rutinaID;
    }
}
```

Figura 51: UserInput.java

4.1.5 Repositorios

4.1.5.1 UserRepository

El `UserRepository` facilita la interacción con la base de datos para realizar operaciones relacionadas con la entidad `User`.

Como se observa en la Fig. 52, son tres los métodos que componen este repositorio:

- **findByEmail(String email):** Este método devuelve el usuario asociado al correo electrónico pasado por parámetro. Se usa `Optional <User>` como tipo de return para evitar el uso de nulls en caso de que no devuelva ningún usuario.
- **existsByEmail(String email):** Verifica si existe un usuario con el correo electrónico pasado por parámetro. Este método se utiliza para comprobar si existe un usuario cuando se registra.
- **findByNombre(String nombre):** Al igual que `findByEmail`, este método devuelve el usuario con el nombre especificado.

4.1.6 Servicios

4.1.6.1 CustomUserDetailsService

Este servicio se encarga de recuperar el usuario de la base de datos a partir de su correo electrónico y es necesario para el funcionamiento de la autenticación en el sistema con JWT.

Su único método, `loadUserByUsername(String email)` busca un usuario por su correo electrónico. Si lo encuentra, se construye un objeto `UserDetails` de Spring Security que contiene la información necesaria para la autenticación.

```
@Service
public class CustomUserDetailsService implements UserDetailsService {

    /* Repositorio de usuarios para acceder a la base de datos */
    @Autowired
    private UserRepository usuarioRepository;

    /* Usuario actual obtenido durante la autenticación */
    private User user;

    /* Método para cargar los detalles del usuario a partir del correo electrónico */
    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        user = usuarioRepository.findByEmail(email)
            .orElseThrow(() -> new UsernameNotFoundException("User not found with username: " + email));

        List<SimpleGrantedAuthority> authorities = Collections.singletonList(new SimpleGrantedAuthority("ROLE_" + user.getRol()));

        return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), authorities);
    }
}
```

Figura 52: CustomUserDetailsService.java

4.1.7 Otros ficheros

4.1.7.1 application.properties

Este fichero, reflejado en la Fig. 53, es el principal fichero de configuración del componente Java. En él, se almacenan distintas configuraciones necesarias para que la aplicación funcione correctamente.

La configuración más importante es la de la conexión con MySQL. También sirve para configurar el nivel de los logs de la aplicación.

```
spring.datasource.username=conexionTFG
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.datasource.url=jdbc:mysql://localhost:3306/mytraining

#logging.level.org.springframework.security=DEBUG
#logging.level.org.springframework.web=DEBUG
#logging.level.root=DEBUG
#logging.level.com.tfg.mytraining=DEBUG

jwt.secret=1234
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.cache=false
```

Figura 53: application.properties

4.1.7.2 pom.xml

El archivo pom.xml mostrado en la Fig. 54 es un fichero esencial en todos los proyectos basados en Maven. Su función principal es la gestión de dependencias, que no es más que la “importación” de las bibliotecas externas que se han utilizado en el proyecto, como Spring o MySQL.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.3</version>
    <relativePath/>
  </parent>
  <groupId>com.tfg</groupId>
  <artifactId>mytraining</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>mytraining</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>21</java.version>
  </properties>
  <dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
  </dependency>
  </dependencies>
</project>
```

Figura 54: Fragmento del fichero pom.xml

4.2 Microservicio Python

Este microservicio está compuesto por cinco archivos que trabajan conjuntamente para desplegar el servicio y usar el modelo *Random Forest* previamente entrenado para predecir los pesos de los ejercicios.

Su estructura está encapsulada dentro de un contenedor Docker para facilitar su despliegue en cualquier entorno.

Esta es la estructura de directorios:

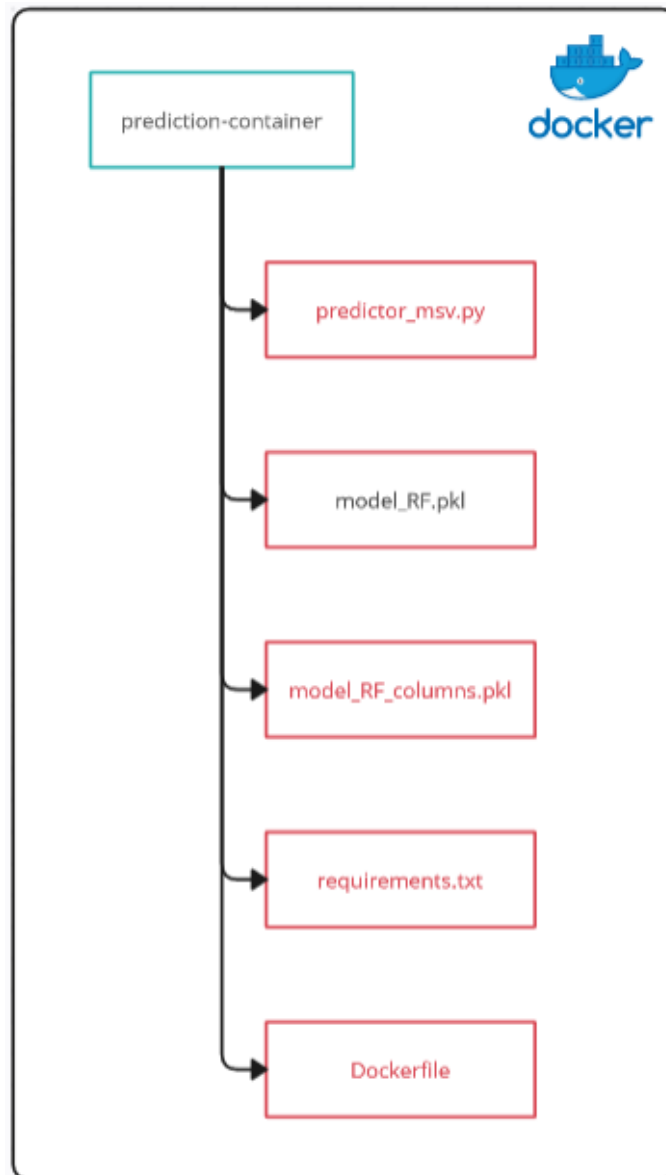


Figura 55: Jerarquía de ficheros del microservicio Python

El entrenamiento y posterior despliegue de un modelo de *Machine Learning* como *Random Forest* conlleva seguir una serie de pasos que se detallan a continuación.

4.2.1 Creación del conjunto de datos

Entrenar un modelo de *Machine Learning* requiere disponer de un conjunto de datos. Con este *dataset*, el modelo aprende y extrae patrones para poder realizar las predicciones.

La selección y preparación de un conjunto de datos es un proceso crucial para el desarrollo, ya que este es el principal responsable de los resultados que se obtienen.

Para este proyecto, hacía falta un conjunto de datos que tuviera unas columnas muy concretas:

- Edad
- Género
- Peso corporal
- Frecuencia de entrenamiento
- Ejercicio
- Peso del ejercicio
- Valoración del peso del ejercicio

No existe un *dataset* con datos reales de estos campos.

Para realizarlo de la manera más real posible, se ha utilizado uno de la plataforma Kaggle que tenía los campos edad, género, peso corporal y frecuencia de entrenamiento, entre otros datos de personas reales.

Con esto, se ha conseguido tener más de 3.000 entradas de datos personas reales, a los que hay que añadir el campo de nombre de ejercicio, peso y valoración de ese peso.

El primer paso que se realizó sobre el *dataset* original fue eliminar las columnas no necesarias con un script de Python llamado `crop_dataset.py`, el cual se puede ver en la Fig. 56.

El resultado de la utilización de este script se muestra en la Fig. 57.

```
# Paso 1: Lee el archivo CSV
df = pd.read_csv('C:\\Users\\jnv77\\Documents\\MICROSERVICIOS TFG\\dataset.csv')

# Paso 2: Eliminar columnas
columnas_a_borrar = ["Column1", "Column2", "Column3"]
df = df.drop(columnas_a_borrar, axis=1)

# Paso 3: Guarda el DataFrame modificado a un nuevo archivo CSV
df.to_csv('dataset_test.csv', index=False)
```

Figura 56: `crop_dataset.py`

```

Gender, Age, Height, Weight, FAF
Female, 21.0, 1.62, 64.0, 0.0
Female, 21.0, 1.52, 56.0, 3.0
Male, 23.0, 1.8, 77.0, 2.0
Male, 27.0, 1.8, 87.0, 2.0
Male, 22.0, 1.78, 89.8, 0.0
Male, 29.0, 1.62, 53.0, 0.0
Female, 23.0, 1.5, 55.0, 1.0
Male, 22.0, 1.64, 53.0, 3.0
Male, 24.0, 1.78, 64.0, 1.0
Male, 22.0, 1.72, 68.0, 1.0
Male, 26.0, 1.85, 105.0, 2.0
Female, 21.0, 1.72, 80.0, 2.0
Male, 22.0, 1.65, 56.0, 2.0
Male, 41.0, 1.8, 99.0, 2.0
Male, 23.0, 1.77, 60.0, 1.0

```

Figura 57: Conjunto de datos en formato csv tras haber eliminado las columnas

Posteriormente, se ha creado un segundo *script* llamado **data_generate.py**. Este añade el resto de las columnas necesarias al *dataset* recortado que aparece en la Fig. 57. Se exponen a continuación sus funciones más importantes:

4.2.1.1 Función de cálculo de peso y valoración de los ejercicios

La función **calcular_peso_valoracion()** calcula el peso que se debe aplicar al ejercicio y la valoración ficticia que el usuario le ha puesto a este peso:

El *script* `data_generate.py` comienza definiendo ponderaciones específicas para cada ejercicio en un diccionario que se puede ver en la Fig. 58. Debido a que el factor más influyente a la hora de predecir los pesos es el peso corporal del usuario, se ha creado un diccionario para multiplicar este por una ponderación según el ejercicio que se haya escogido.

```

# Ponderacion de cada ejercicio
factores_ajuste = {
    "Sentadilla": 1,
    "Press de banca": 0.7,
    "Press de banca inclinado con barra": 0.6,
    "Pullover con barra": 0.4,
    "Remo con barra": 0.2,
    "Peso muerto": 0.7,
    "Good mornings": 0.1,
    "Press militar": 0.5,
    "Remo al cuello con barra": 0.4,
    "Press de banca con agarre cerrado": 0.3,
    "Curl de bíceps con barra": 0.2,
    "Curl de bíceps con agarre inverso": 0.2,
    "Press francés con barra": 0.2,
    "Prensa de piernas": 1.4
}

```

Figura 58: Diccionario de ponderaciones de ejercicios del script `data_generate.py`

Después, como se puede observar en la Fig.59, se multiplica el peso corporal del usuario por la ponderación del ejercicio seleccionado, obteniendo así un peso base para cada ejercicio. Este peso base se irá modificando según las características del usuario. Este valor se asegura con la función max() para que no sea menor que uno evitando valores muy bajos.

```
factor = factores_ajuste.get(ejercicio_nombre, 0.5)
peso_ajustado = max(1, peso_usuario * factor)
valoracion = 2
```

Figura 59: Fragmento de código de la función calcular_peso_valoracion()

La valoración, como se ve en la Fig. 59, empieza con el valor 2 para servir como un punto de partida neutral antes de aplicar los ajustes según las características del usuario.

Tras esto, teniendo en cuenta la edad, el género y la frecuencia de entrenamiento del usuario se reduce el peso y la valoración para cada ejercicio siguiendo las reglas que se pueden encontrar en la Fig. 60:

- **Género:** Para las mujeres, se reduce un 20% el peso y la valoración para reflejar las diferencias en la masa muscular y la fuerza entre géneros.
- **Edad:** Se aplican reducciones en el peso y la valoración para reflejar la disminución en la fuerza y la masa muscular con el aumento de edad del usuario
- **FAF (Frecuencia de entrenamiento):** La frecuencia de entrenamiento junto al peso corporal del usuario han sido los factores más determinantes para asociar el peso y la valoración, llegando a reducir hasta la mitad el peso sugerido. Este factor es muy importante ya que el hecho de entrenar regularmente te permite manejar pesos mucho más elevados en los ejercicios.

```
if genero == "Femenino":
    peso_ajustado *= 0.8
    valoracion *= 0.8

if edad > 70:
    peso_ajustado *= 0.3
    valoracion *= 0.3
elif edad > 60:
    peso_ajustado *= 0.4
    valoracion *= 0.4
elif edad > 50:
    peso_ajustado *= 0.5
    valoracion *= 0.5
elif edad > 35:
    peso_ajustado *= 0.8
    valoracion *= 0.8
elif edad > 25:
    peso_ajustado *= 0.9
    valoracion *= 0.9

if 0<=faf<=1:
    peso_ajustado *= 0.5
    valoracion *= 0.5
elif 1<faf<=2:
    peso_ajustado *=0.7
    valoracion *=0.7
elif 2<faf<=3:
    peso_ajustado *= 1
    valoracion *= 1
```

Figura 60: Fragmento de código de data_generate.py

Finalmente, la función devuelve ambos valores que posteriormente serán introducidos en cada fila del conjunto de datos como se puede ver en la Fig. 62.

4.2.1.2 Función que asigna rutinas y ejercicios

La función `asignar_rutina_ejercicios()`, cuya implementación se puede ver en la Fig. 61 es la encargada de asignar la rutina a cada usuario, llamar a la función que calcula el peso y la valoración; y crear el nuevo *dataset* contemplado en la Fig. 62.

Esta función realiza las siguientes acciones:

- Primero, itera sobre cada fila (usuario) en el *dataset* recortado de la Fig. 57, seleccionando una rutina al azar del diccionario de rutinas reflejado en la Fig. 63.
- Después, se llama a la función `calcular_peso_valoración()` pasándole los parámetros género, peso corporal, frecuencia de entrenamiento y edad existentes en *dataset* (Fig. 57), que es al que le queremos añadir el resto de columnas.
- `calcular_peso_valoración()` devuelve el peso recomendado y la valoración para ese ejercicio, calculado con las distintas ponderaciones que usa esta función.
- Por último, se genera el *dataset* final (Fig. 62) que es el que se usa para entrenar el modelo Random Forest.

```
def asignar_rutina_ejercicios(df, rutinas):
    resultados = []
    for i, row in df.iterrows():
        rutina_key = str(random.choice(list(rutinas.keys())))
        rutina = rutinas[rutina_key]
        for ejercicio_key, ejercicio_nombre in rutina['ejercicios'].items():
            peso, valoracion = calcular_peso_valoracion(row['Weight'], row['FAF'], ejercicio_nombre, row['Gender'], row['Age'])
            resultados.append({
                'Gender': genero_numerico.get(row['Gender'], -1),
                'Age': row['Age'],
                'Height': row['Height'],
                'Weight': row['Weight'],
                'FAF': row['FAF'],
                'Ejercicio_ID': ejercicio_key,
                'Peso Ejercicio': peso,
                'Valoracion': valoracion,
                'Rutina_ID': rutina_key
            })
    return pd.DataFrame(resultados)
```

Figura 61: Función `asignar_rutina_y_ejercicios()`

```
Gender, Age, Height, Weight, FAF, Ejercicio_ID, Peso Ejercicio, Valoracion, Rutina_ID, Ejercicio, Rutina
Female, 21.0, 1.62, 64.0, 0.0, 5.1, 20.0, 1.0, 5, Sentadilla, Rutina Pierna
Female, 21.0, 1.62, 64.0, 0.0, 5.2, 18.0, 1.0, 5, Peso muerto, Rutina Pierna
Female, 21.0, 1.62, 64.0, 0.0, 5.3, 36.0, 1.0, 5, Prensa de piernas, Rutina Pierna
Female, 21.0, 1.52, 56.0, 3.0, 3.1, 28.0, 3.0, 3, Press militar, Rutina Hombros
Female, 21.0, 1.52, 56.0, 3.0, 3.2, 22.0, 3.0, 3, Remo al cuello con barra, Rutina Hombros
Female, 21.0, 1.52, 56.0, 3.0, 3.3, 17.0, 3.0, 3, Press de banca con agarre cerrado, Rutina Hombros
Male, 23.0, 1.8, 77.0, 2.0, 3.1, 29.0, 2.0, 3, Press militar, Rutina Hombros
Male, 23.0, 1.8, 77.0, 2.0, 3.2, 23.0, 2.0, 3, Remo al cuello con barra, Rutina Hombros
Male, 23.0, 1.8, 77.0, 2.0, 3.3, 17.0, 2.0, 3, Press de banca con agarre cerrado, Rutina Hombros
```

Figura 62: Versión final del *dataset* tras añadir el resto de las columnas

```

rutinas = {
  "0": {
    "nombre": "Rutina Full Body",
    "ejercicios": {
      "0.1": "Sentadilla",
      "0.2": "Press de banca",
      "0.3": "Peso muerto",
      "0.4": "Press militar",
      "0.5": "Curl de bíceps con barra",
      "0.6": "Press francés con barra"
    }
  },
  "1": {
    "nombre": "Rutina Espalda",
    "ejercicios": {
      "1.1": "Remo con barra",
      "1.2": "Peso muerto",
      "1.3": "Good mornings"
    }
  }
}

```

Figura 63: Fragmento del diccionario de rutinas de data_generate.py

4.2.2 Generación del modelo Random Forest

El algoritmo de Random Forest utiliza múltiples árboles de decisión para realizar predicciones precisas y efectivas. Este método combina el poder de varios árboles de decisión para llegar a un único resultado.

Un árbol de decisión es un método de aprendizaje automático que está compuesto por varios nodos. Estos nodos son condiciones que se generan a partir de los valores procedentes del conjunto de datos proporcionado para su entrenamiento. Por ejemplo, un nodo de decisión es si la persona tiene más de 25 años, otro si la frecuencia de entrenamiento está entre 1 y 2.

Estas decisiones o nodos, tal y como se ve en la Fig. 64, se ejecutan hasta llegar a los nodos finales, llamados *leafs*, ya que representan la parte final del árbol, que es la hoja.

El algoritmo de Random Forest utiliza varios árboles de decisión, dividiendo el conjunto de datos entre ellos. Por ejemplo, si hay 10.000 filas en el *dataset* que entrena el modelo y el número de árboles que componen el Random Forest son 5, se dividirá en 2.000 filas para cada árbol.

Una vez que cada árbol ha tomado una decisión por cada rama, en este caso sobre el peso, se decide el valor que más haya coincidido entre los resultados de los árboles. Por ejemplo, si hay 100 árboles y 40 han dado la predicción de 10kg para el ejercicio de sentadilla y 60 han dado como resultado 12kg, saldrá 12kg como el peso recomendado.

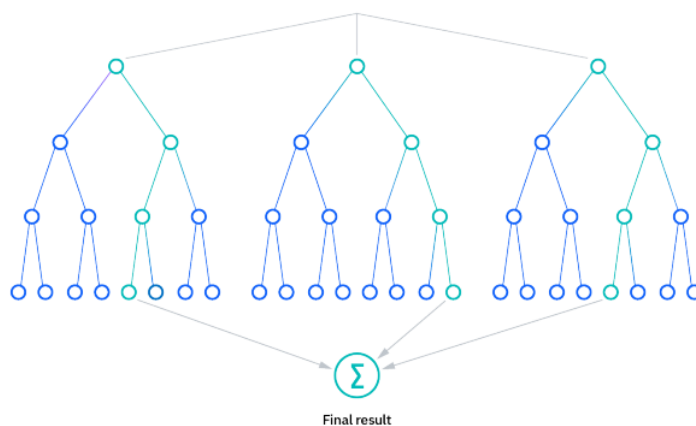


Figura 64: Representación del algoritmo *Random Forest*

4.2.3 Entrenamiento del modelo con la librería Scikit-Learn

Para entrenar el modelo que predecirá los pesos se ha desarrollado el *script* `model_train.py` (Fig. 65), que utiliza la librería Scikit-Learn, que es la que implementa todas las herramientas necesarias para el entrenamiento del *Random Forest*.

Este *script* realiza las siguientes acciones:

- Primero se carga el *dataset* preparado en el paso anterior (Fig. 62).
- Se seleccionan las columnas relevantes para incluirlas en el modelo: 'Gender', 'Age', 'Weight', 'FAF', 'Ejercicio_ID', y 'Valoracion'. Hay algunas columnas más en el conjunto de datos que no se tienen en cuenta para el entrenamiento, como puede ser la altura de la persona.
- El *dataset* se divide en conjuntos de entrenamiento y prueba. En este caso se ha utilizado la proporción de 80% del *dataset* destinado al entrenamiento y 20% para *testing*.
- Después se crea el modelo con la función `RandomForestRegressor`, con 100 árboles de decisión. Tras la creación, se entrena introduciéndole las columnas relevantes del *dataset* que se definieron en los pasos previos.
- Finalmente, se genera un archivo de extensión `.pkl` que contiene el modelo entrenado. Este ejecutable se cargará en el servidor Flask y no necesitará el *dataset* para funcionar, porque es el modelo ya entrenado.

```
df['Ejercicio_ID'] = df['Ejercicio_ID'].astype(float)

features = ['Gender', 'Age', 'Weight', 'FAF', 'Ejercicio_ID', 'Valoracion']
X = df[features]
y = df['Peso Ejercicio']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

joblib.dump(X_train.columns, 'model_columnsv03.pkl')
joblib.dump(rf, 'modelo_entrenado_v03.pkl')
```

Figura 65: Fragmento del código del `model_train.py`

4.2.4 Implementación del microservicio

Finalmente, tras el proceso de entrenamiento del modelo se implementa el microservicio que realiza las predicciones de peso para los ejercicios que componen una rutina (Fig. 66).

Estas son las acciones que realiza:

- Primero, se crea una instancia de Flask y se carga el modelo y las columnas del modelo con los archivos con extensión pkl previamente generados.
- Se vuelve a definir el diccionario de rutinas exactamente igual al de la Fig. 63.
- Con `app.route` se define el *endpoint* que se llama desde el componente Java. Dentro de este *endpoint*, lo primero que se hace es procesar los datos que llegan desde Spring, que son los datos que el usuario ha rellenado en el formulario de la aplicación.
- Después, se obtienen los ejercicios pertenecientes a la rutina que el usuario ha escogido y se realiza la predicción usando el modelo de *Random Forest*. El resultado obtenido, el peso del ejercicio, es agregado al cuerpo de la respuesta.
- Cuando se han incluido todos los nombres de los ejercicios con el peso predicho de cada uno, se almacenan en un diccionario y se devuelven en formato JSON al componente Spring, el cual será el encargado de mandarlo al *frontend* y se muestre al usuario.

```
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    print("Datos recibidos:", data)

    rutina_id = data['rutinaID']
    ejercicios_rutina = rutinas.get(rutina_id, {}).get("ejercicios", {})
    print("Ejercicios de la rutina:", ejercicios_rutina)

    predictions = []
    for ejercicio_id, ejercicio_nombre in ejercicios_rutina.items():
        print("Prediciendo para el ejercicio:", ejercicio_nombre)
        input_data = pd.DataFrame({key: [value] for key, value in data.items() if key != 'Ejercicio_ID'})
        input_data['Ejercicio_ID'] = ejercicio_id
        print("Datos de entrada para el ejercicio:", input_data)

        # Asegura que el DataFrame tenga las mismas columnas que el modelo
        for col in model_columns:
            if col not in input_data.columns:
                input_data[col] = 0
        input_data = input_data.reindex(columns=model_columns, fill_value=0)
        print("Datos de entrada después de reindexar:", input_data)

        # Predecir el peso para el ejercicio
        predicted_weight = model.predict(input_data)[0]
        print("Peso predicho para el ejercicio:", predicted_weight)
        predictions.append({
            'rutina_id': rutina_id,
            'exercise_id': ejercicio_id,
            'exercise_name': ejercicio_nombre,
            'predicted_weight': predicted_weight
        })

    return jsonify(predictions)
```

Figura 66: Fragmento de código de `prediction_msv.py`

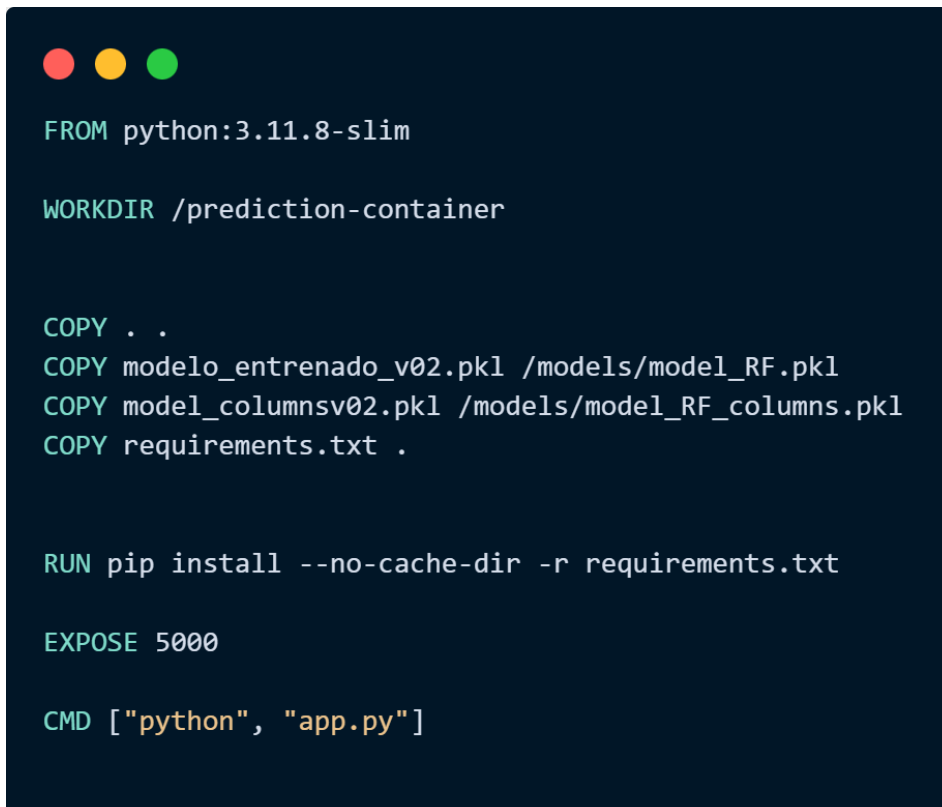
4.2.5 Proceso de Dockerización

El último paso en la configuración del microservicio Python, es la *dockerización* del servidor.

Este proceso implica encapsular el entorno del servidor Flask en un contenedor Docker, lo que facilita la implementación y garantiza el despliegue del entorno de ejecución en diferentes plataformas. La Fig. 67 muestra el contenido del Dockerfile, que define cómo se construye el contenedor Docker:

- Primero, se especifica la imagen de Python que el contenedor utiliza, que en este caso es python:3.11.8-slim.
- Se establece /prediction-container como el directorio de trabajo dentro del contenedor, donde se copiarán todos los archivos necesarios para ejecutar la aplicación.
- El script de Python que ejecuta el servidor y los modelos de predicción (model_RF.pkl y model_RF_columns.pkl) se copian en el contenedor, así como el requirements.txt, que lista todas las dependencias necesarias. El comando RUN pip install --no-cache-dir -r requirements.txt se utiliza para instalar estas dependencias.
- El puerto 5000 se expone mediante EXPOSE 5000, configurando el contenedor para aceptar conexiones entrantes en este puerto, que es el utilizado por el servidor Flask.

Por último, desde la línea de comandos se construye y se despliega este contenedor, siendo accesible al componente Java Spring en el puerto 5000 y en la IP interna de Docker, que hay que consultar en la carpeta /etc/host del sistema operativo Windows.



```
FROM python:3.11.8-slim

WORKDIR /prediction-container

COPY . .
COPY modelo_entrenado_v02.pkl /models/model_RF.pkl
COPY model_columnsv02.pkl /models/model_RF_columns.pkl
COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 5000

CMD ["python", "app.py"]
```

Figura 67: Fichero Dockerfile

4.3 Frontend

El desarrollo del *frontend* para el proyecto se ha abordado utilizando las tecnologías HTML, CSS y JavaScript, tal y como se detalla en el apartado 2 de esta memoria.

Los ficheros del *frontend* se han tomado de plantillas disponibles en internet, las cuales han sido modificadas para cumplir con los requisitos del proyecto. Estas modificaciones incluyen la implementación de funcionalidades para enviar datos en formato JSON a los *endpoints* del *backend*, así como la inclusión de animaciones para mejorar la estética de la interfaz gráfica, como la visualización de los resultados de las predicciones.

La modificación más relevante del *frontend* es la creación de la función JavaScript que maneja la captura y el envío de los datos del formulario a los *endpoints* del *backend* mediante solicitudes AJAX.

Se puede ver un ejemplo de esta función en la Fig. 68, la cual corresponde a la función que envía los datos del formulario al *endpoint* /prediction de Spring, que es el que hace de pasarela entre el *frontend* y el microservicio Python.

La función realiza las siguientes acciones:

- Primero se recopilan los valores de los campos del formulario, asegurando que el formato de estos datos es el que espera recibir el *endpoint*.
- Después, se envía una solicitud POST al *endpoint*, utilizando el formato JSON para los datos.
- Si el envío es exitoso, se recibe una respuesta por parte del *endpoint*. Esta se procesa para mostrarla adecuadamente en la interfaz gráfica con la función `displayResults()`

```
var formData = {
  Gender: genderVal ? parseInt(genderVal, 10) : null,
  Age: $('input[name="Age"]').val() ? parseInt($('input[name="Age"]').val(), 10) : null,
  Weight: $('input[name="Weight"]').val() ? parseFloat($('input[name="Weight"]').val()) : null,
  FAF: FAFVal ? parseInt(FAFVal, 10) : null,
  rutinaID: $('input[name="rutinaID"]:checked').val(),
  Valoracion: 2
};

// Realiza una solicitud AJAX al endpoint de Java
$.ajax({
  url: '/prediction',
  type: 'POST',
  contentType: 'application/json',
  data: JSON.stringify(formData),
  success: function(response) {

    var data = typeof response === 'string' ? JSON.parse(response) : response;
    displayResults(data);
  },
  error: function(xhr, status, error) {
    console.error("Error: ", error);
  }
});
});
```

Figura 68: Fragmento de código Javascript de envío de datos JSON al *endpoint* prediction

5 MANUAL DE INTERFAZ DE USUARIO

En esta sección se realiza una guía de uso de la aplicación para el usuario.

Al entrar en la aplicación web, aparecerá la pantalla de inicio de sesión (Fig. 69), donde el usuario tiene que meter sus credenciales y pulsar el botón “Login”, el cual redirigirá al usuario al menú principal de la aplicación en caso de que los datos de inicio de sesión sean correctos.

Si es la primera vez que el usuario utiliza la aplicación, tendrá que registrarse previamente. Para entrar en la página de registro hay que pulsar en el texto “Regístrate aquí”.

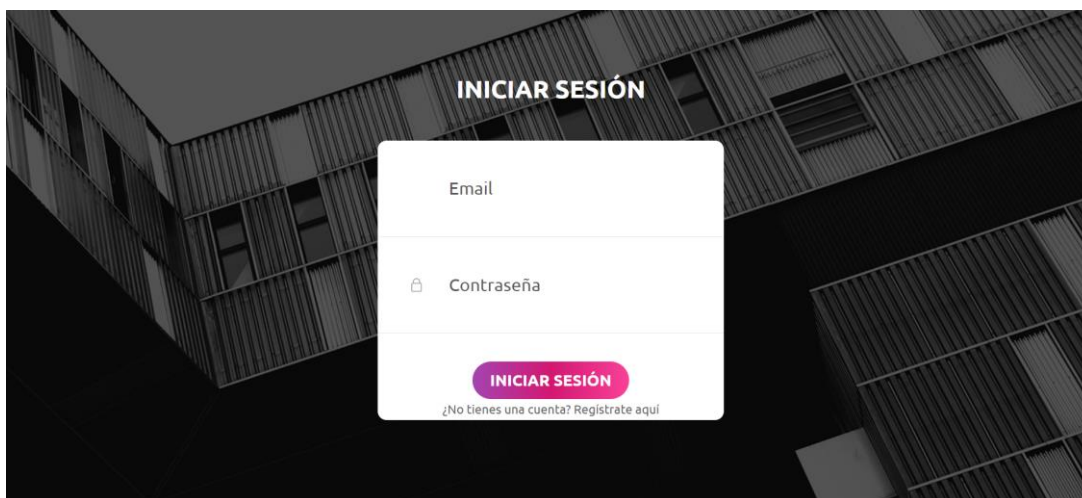


Figura 69: Página de inicio de sesión

Al pulsar en el texto para registrarse, aparece una pantalla como la de la Fig. 70.

Una vez rellenados los datos y pulsado sobre el botón de “Registrarse”, si el email no está ya registrado en la base de datos de la aplicación, se realizará con éxito el registro, redirigiendo al usuario a la pantalla de inicio de sesión.

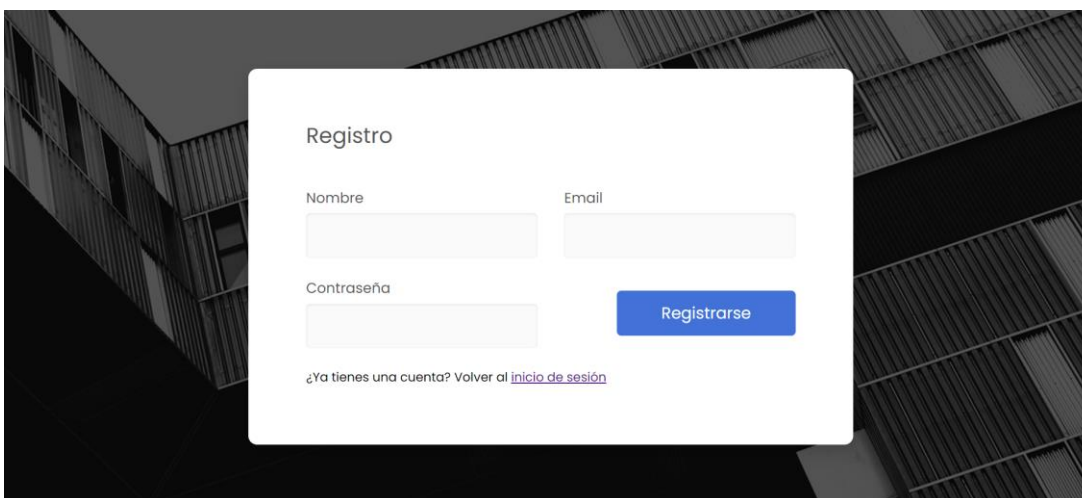


Figura 70: Página de registro de usuario

Una vez iniciada la sesión, aparecerá el menú principal de la aplicación (Fig. 71), donde aparece información sobre aspectos relevantes de salud y sobre la funcionalidad de la aplicación.

Esta pantalla tiene una barra superior con 4 botones remarcados en naranja en la Fig. 72.

Estos botones realizan las siguientes funciones:

- **Botón “MYTRAINING”**: Redirige al usuario al menú principal desde la pestaña de predictor.
- **Botón “INICIO”**: Redirige al usuario al menú principal desde la pestaña de predictor.
- **Botón “Predictor”**: Redirige al usuario a la pantalla para obtener las rutinas con el peso recomendado para cada ejercicio.
- **Botón “Cerrar sesión”**: Cierra la sesión del usuario y lo redirige a la pantalla de inicio de sesión.



Figura 71: Página del menú principal

Al pulsar sobre el botón “Predictor”, aparece la pantalla de la Fig.72, en la cual el usuario puede seleccionar una rutina entre las seis rutinas disponibles e introducir su género, frecuencia de entrenamiento, edad y peso corporal.

Una vez introducidos estos datos, al pulsar sobre el botón “Continuar” aparecerán los ejercicios de la rutina seleccionada junto a sus respectivos pesos, tal y como aparece reflejado en la Fig.73.

ELIGE TU RUTINA

Completa
Rutina de ejercicios para fortalecer todos los grupos musculares

Espalda
Rutina de ejercicios para ejercitar los músculos de la espalda

Pecho
Rutina de ejercicios para ejercitar los músculos del pecho

Hombro
Rutina de ejercicios para ejercitar los músculos del hombro

Brazo
Rutina de ejercicios para ejercitar los músculos del brazo

Pierna
Rutina de ejercicios para ejercitar los músculos de las piernas

INFORMACIÓN ADICIONAL

Género

¿Con qué frecuencia entrenas?

Edad

Peso(Números redondos)

Continuar

Figura 72: Página de predicción de pesos

70

Continuar

- Sentadilla [31.35 kg]
- Press de banca [21.37 kg]
- Peso muerto [21.34 kg]
- Press militar [16.03 kg]
- Curl de bíceps con barra [6.16 kg]
- Press francés con barra [6.16 kg]

Figura 73: Resultado de la predicción

6 CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

Para cerrar esta memoria, es necesario destacar tanto las conclusiones personales como las técnicas obtenidas a lo largo del desarrollo de este proyecto.

A pesar de los desafíos enfrentados, este período me ha servido de aprendizaje para crecer tanto personal como profesionalmente.

El hecho de introducir la tecnología *Machine Learning* en el proyecto ha representado un gran reto debido a su complejidad y también a pensar la idea de qué función podía realizar en la aplicación. Una de las cosas más difíciles fue generar el *dataset* tan particular que hacía falta para entrenar al modelo, ya que este no existía y había que crearlo prácticamente desde cero.

Sin embargo, la satisfacción de superar estas dificultades y lograr predicciones coherentes ha sido muy gratificante.

Desde el punto de vista técnico, la implementación de Spring para gestionar los *endpoints* REST y la conexión con el microservicio Python destacan entre los logros del proyecto. La interconexión de servicios, incluso utilizando diferentes lenguajes de programación, ha demostrado ser una estrategia eficaz para solucionar problemas complejos, ya que el tratamiento de datos y el uso de Machine Learning es mucho más eficaz en el lenguaje Python y, por otra parte, Java tiene muchos más recursos para servir como estructura de una aplicación completa con *frameworks* como Spring.

Para finalizar, en cuanto a líneas futuras de mejora, la creación de un *dataset* real mediante la recopilación de datos de personas reales sería la línea de mejora más evidente, ya que permitiría afinar aún más el modelo de *Machine Learning*.

Adicionalmente, incrementar las rutinas y ejercicios disponibles en la aplicación enriquecería la experiencia del usuario, ofreciendo un mayor número de opciones a elegir. La implementación de rutinas semanales o mensuales que incluyan mesociclos y diferentes recursos para entrenar ayudaría mucho a los usuarios que van al gimnasio.

ANEXO A: INSTALACIÓN Y DESPLIEGUE DE LA APLICACIÓN

En este anexo se va a detallar cómo instalar todos los componentes necesarios para el despliegue de la aplicación en el sistema operativo Windows 11.

A) Instalación de Maven

Se puede descargar desde la [página oficial de Maven](#). La opción que hay que descargar es la de “Source zip archive”, señalada en naranja en la Fig. 74. En el instalador habrá una casilla preguntando si añadir la variable de entorno de Maven a nuestro sistema que habrá que señalar.

The screenshot shows the Apache Maven website's 'Files' section. It includes a table with columns for Link, Checksums, and Signature. The 'Source zip archive' row is highlighted with an orange box around the link 'apache-maven-3.9.6-src.zip'.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.9.6-bin.tar.gz	apache-maven-3.9.6-bin.tar.gz.sha512	apache-maven-3.9.6-bin.tar.gz.asc
Binary zip archive	apache-maven-3.9.6-bin.zip	apache-maven-3.9.6-bin.zip.sha512	apache-maven-3.9.6-bin.zip.asc
Source tar.gz archive	apache-maven-3.9.6-src.tar.gz	apache-maven-3.9.6-src.tar.gz.sha512	apache-maven-3.9.6-src.tar.gz.asc
Source zip archive	apache-maven-3.9.6-src.zip	apache-maven-3.9.6-src.zip.sha512	apache-maven-3.9.6-src.zip.asc

Below the table, there are links for '3.9.6 Release Notes and Release Reference Documentation', 'latest source code from source repository', and 'Distributed under the Apache License, version 2.0'. There is also a section for 'Other Releases' with a recommendation to use the latest version and a link to 'Maven Releases History'.

Figura 74: Página oficial de Maven

Una vez esté instalado Maven, habrá que instalarlo en la terminal ejecutando el comando “mvn install”.

Se puede comprobar que todo ha salido bien ejecutando el comando en la terminal “mvn -version”, tal y como aparece en la Fig. 75.

```
Windows PowerShell
PS C:\Users\jnv77> mvn -version
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: C:\Apache\apache-maven-3.9.6-bin\apache-maven-3.9.6
Java version: 21.0.1, vendor: Eclipse Adoptium, runtime: C:\Program Files\Eclipse Adoptium\jdk-21.0.1.12-hotspot
Default locale: es_ES, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\jnv77>
```

Figura 75: Comprobación de instalación de Maven

B) Instalación de Java JDK

Se puede instalar desde [su página oficial](#). Para este proyecto se ha escogido el JDK 21, por lo que habría que descargar el instalador MSI del sistema operativo Windows, señalado en naranja en la Fig. 76.

JDK Development Kit 21.0.3 downloads

JDK 21 binaries are free to use in production and free to redistribute, at no cost, under the [Oracle No-Fee Terms and Conditions](#) (NFTC).

JDK 21 will receive updates under the NFTC, until September 2026, a year after the release of the next LTS. Subsequent JDK 21 updates will be licensed under the [Java SE OTN License](#) (OTN) and production use beyond the [limited free grants](#) of the OTN license will [require a fee](#).

Linux macOS Windows

Product/file description	File size	Download
x64 Compressed Archive	185.78 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.zip (sha256)
x64 Installer	164.16 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe (sha256)
x64 MSI Installer	162.91 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.msi (sha256)

Documentation Download

Release information

- [Online Documentation](#)

Escritorio 2

Figura 76: Página oficial de Java JDK

Una vez instalado, hay que añadir la variable de Entorno “JAVA_HOME”. Hay que pulsar la tecla Windows y escribir “Editar las variables de entorno del sistema”, pulsar sobre el acceso directo y seleccionar el botón que está en la parte inferior derecha que pone “Variables de entorno”. Ahí, se abre una segunda ventana tal y como aparece en la Fig. 77, donde habrá que añadir una nueva entrada de nombre “JAVA_HOME” y valor igual a la ruta de nuestro equipo dónde se haya instalado el componente JDK.

Variables de entorno

Variables de usuario para jnv77

Variable	Valor
JAVA_HOME	C:\Program Files\Eclipse Adoptium\jdk-21.0.1.12-hotspot
OneDrive	C:\Users\jnv77\OneDrive - UNIVERSIDAD DE SEVILLA
OneDriveCommercial	C:\Users\jnv77\OneDrive - UNIVERSIDAD DE SEVILLA
Path	C:\Users\jnv77\AppData\Local\activestate\cache\bin;C:\Users...
TEMP	C:\Users\jnv77\AppData\Local\Temp
TMP	C:\Users\jnv77\AppData\Local\Temp

Nueva... Editar... Eliminar

Figura 77: Variables de entorno del sistema

Si todo ha salido bien, al igual que con Maven se podrá comprobar que la instalación ha sido exitosa con el comando “java -version”

C) Instalación de XAMPP y configuración de phpMyAdmin

Para instalar XAMPP, simplemente hay que ingresar en la [página oficial](#) y pulsar en XAMPP para Windows, tal y como aparece en la Fig. 78.



Figura 78: Página oficial de XAMPP

Durante el proceso de instalación, se tienen que elegir los componentes: Apache, MySQL y Tomcat.

Una vez instalada, se inicia la aplicación y se pulsa en el botón “start” de los componentes Apache y MySQL.

El paso siguiente es acceder <http://localhost/phpmyadmin/> desde el navegador. Una vez dentro, pulsaremos sobre la opción Bases de Datos, remarcada en naranja en la Fig. 79 y le pondremos el nombre de mytraining a la BBDD en el recuadro también remarcado en naranja y se pulsa sobre el botón “Crear”.

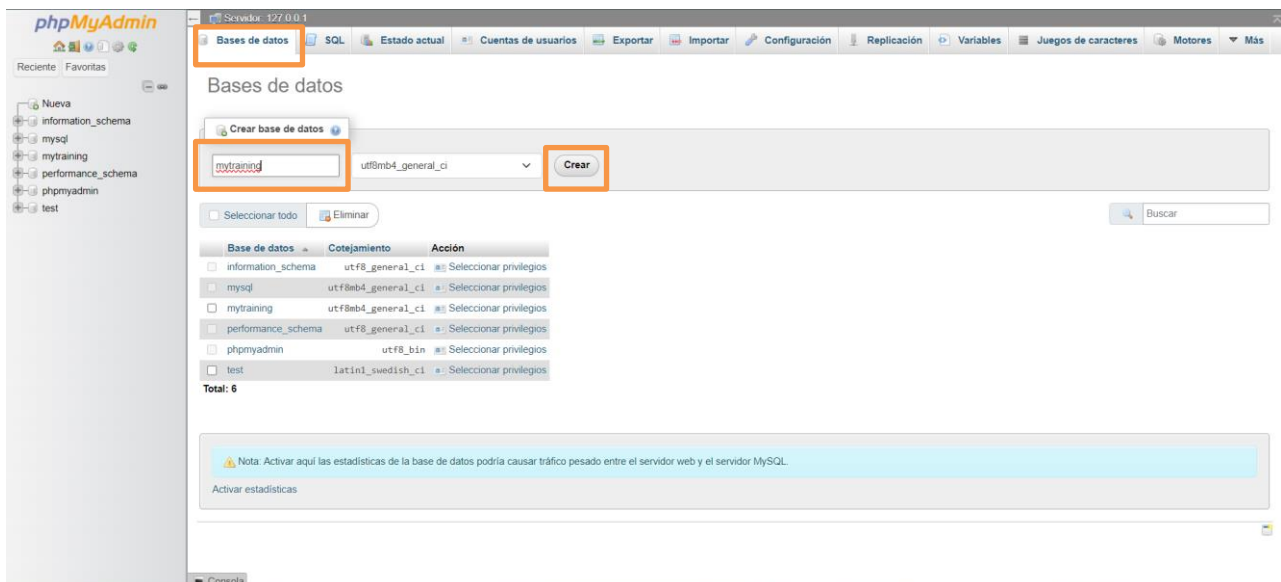


Figura 79: Página principal de phpMyAdmin

Una vez creada la base de datos, hay que dirigirse al botón “privilegios”, situado en la barra horizontal de la parte superior de la página y remarcado en naranja en la Fig. 80.

Una vez estamos en la pestaña de privilegios, hay que pulsar sobre “Agregar cuenta de usuario”, remarcado en naranja en la Fig. 80.

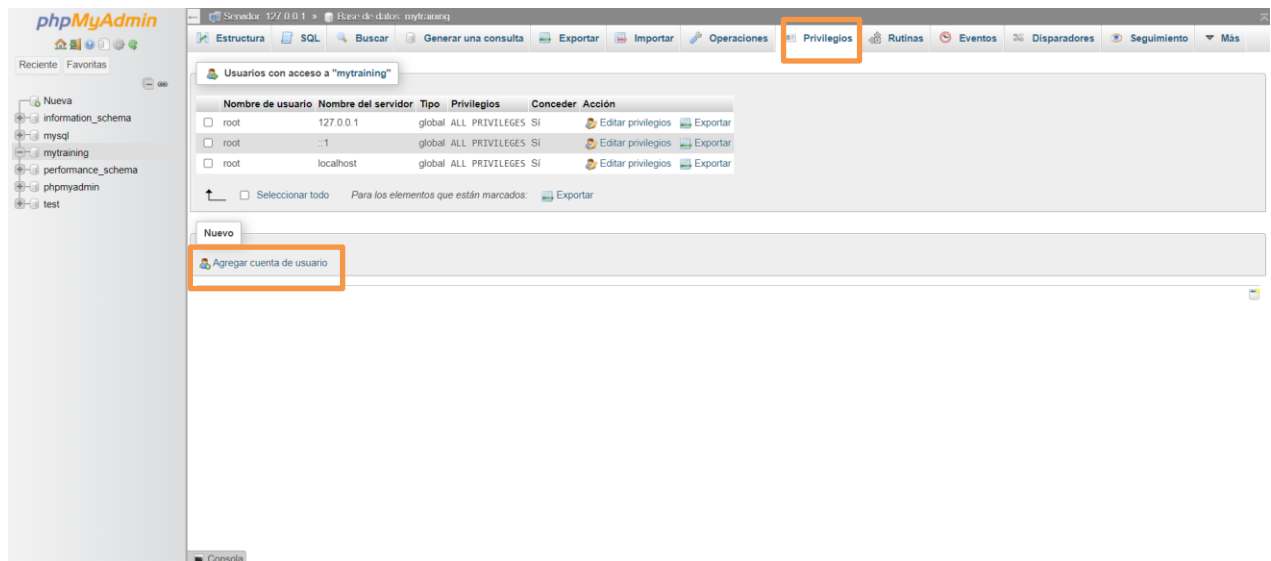


Figura 80: Pestaña de privilegios de la página phpMyAdmin

En la opción de Agregar cuenta de usuario, habrá que rellenar los campos de nombre y contraseña señalados en naranja en la Fig. 81.

El valor del nombre tiene que ser “conexionTFG” y la contraseña “1234”.

También hay que marcar la casilla de la parte inferior señalada en naranja que pone “Seleccionar todo”, justo al lado de “Privilegios Globales”.

Una vez realizadas estas 3 acciones, hay que pulsar en el botón “Continuar” que aparece al final de la página y XAMPP estará listo para su uso.

D) Instalación de Visual Studio Code y de sus extensiones

El editor Visual Studio Code se puede descargar desde su [página oficial](#). Una vez descargado, tenemos que abrir la carpeta “mytraining” que implementa el proyecto e importarla dándole al botón que aparece en la Fig. 81.

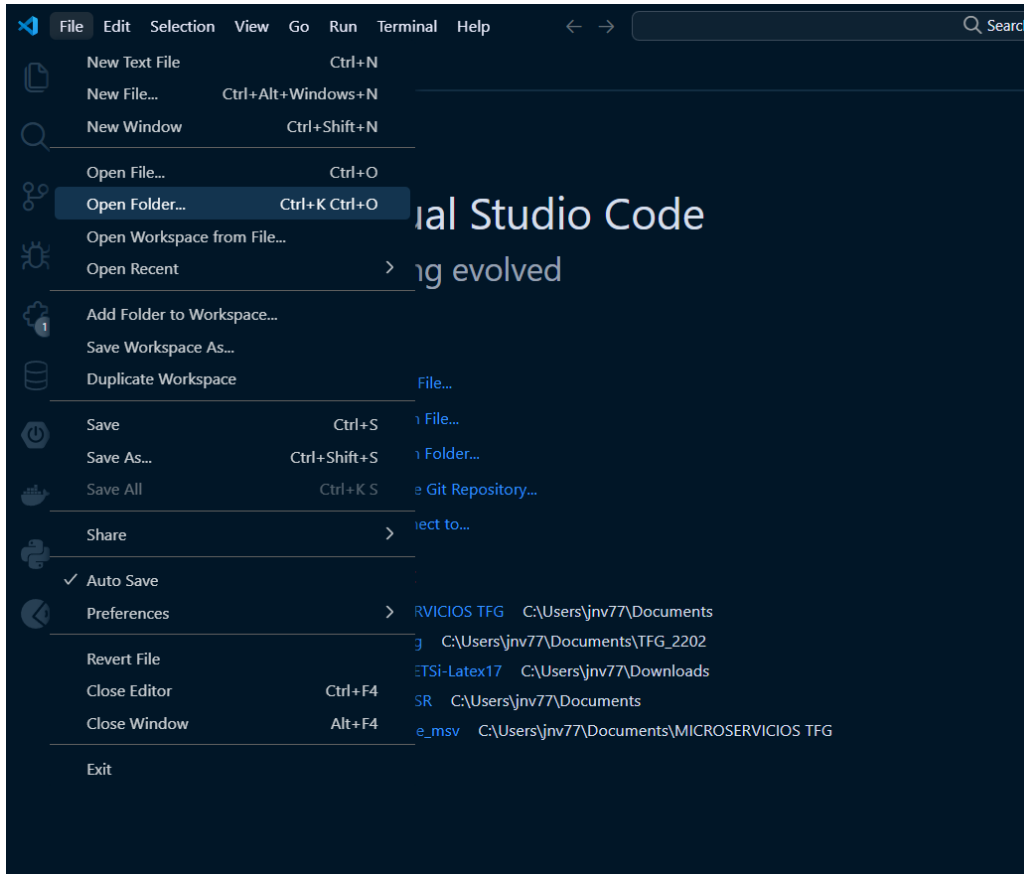


Figura 81: Opción de abrir carpeta de Visual Studio Code

Una vez dentro, pulsamos sobre el cuarto icono de la barra vertical izquierda con forma de pieza de puzle señalado en naranja en la Fig. 82.

El botón con forma de puzle abrirá un buscador en el que se deben buscar e instalar las siguientes extensiones:

- Spring Boot Dashboard
- REST Client
- Project Manager for Java
- MySQL
- Maven for Java

En esta última extensión, “Maven for Java”, si se abre se puede observar un engranaje de configuración. Al pulsarlo, aparecerán varias opciones y hay que elegir la opción de “Extension settings”, señalada en naranja en la Fig. 82.

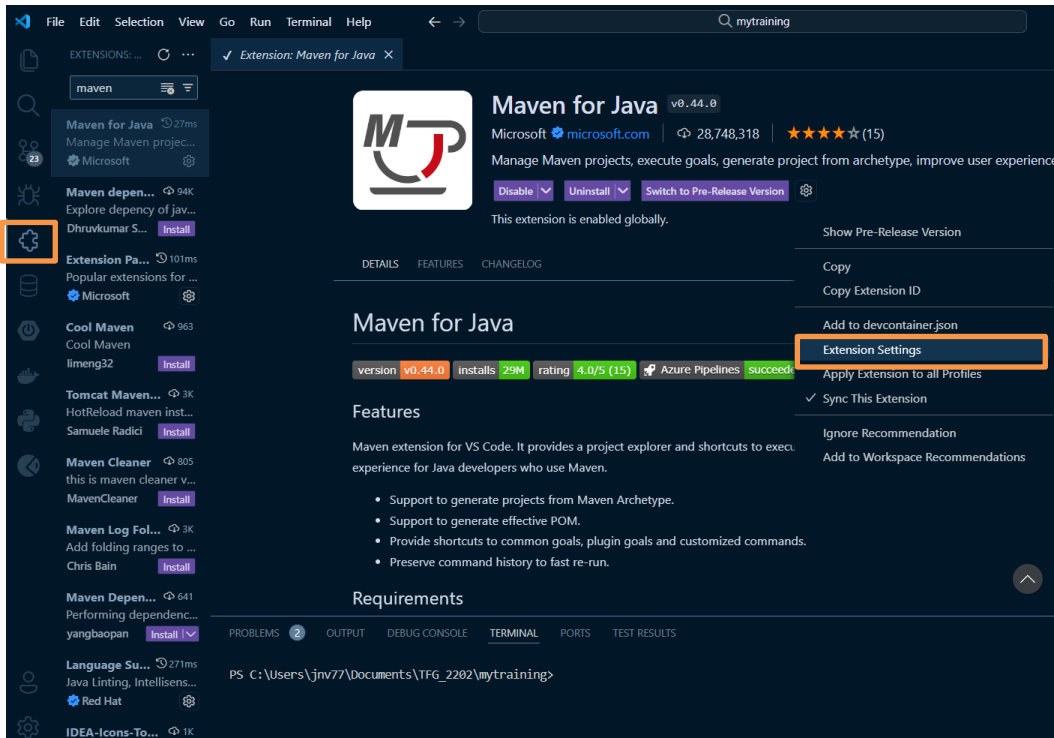


Figura 82: Apartado de extensiones en Visual Studio Code

Una vez dentro de “Extension Settings”, hay que modificar el campo “Maven > Executable: Path” y añadir la ruta del directorio de instalación de Maven, en concreto la ruta donde está la carpeta “bin\mvn”, tal y como se puede ver en la Fig. 83.

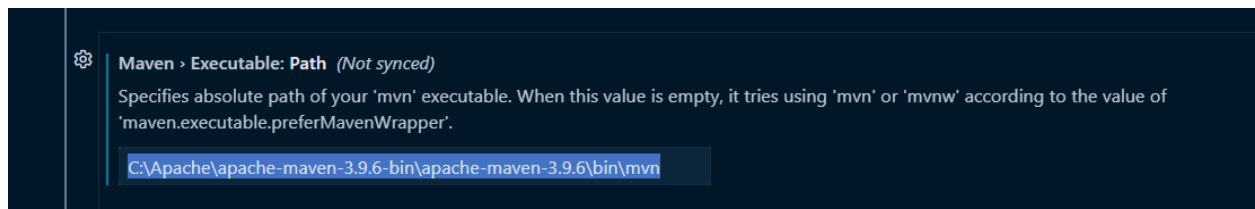


Figura 83: Extension settings de la extensión “Maven For Java”

Tras esto, hay que dirigirse a la extensión Database, a la que se puede acceder mediante el botón remarcado en naranja en la barra vertical de la parte izquierda que aparece en la Fig. 84.

Tras entrar, hay que pulsar sobre el botón (+), situado en la parte superior izquierda de la extensión y nos aparecerá una página como la de la Fig.84.

En esta página hay que rellenar los campos “Username” y “Password” con los valores conexionTFG y 1234 y pulsar sobre el botón “Connect”.

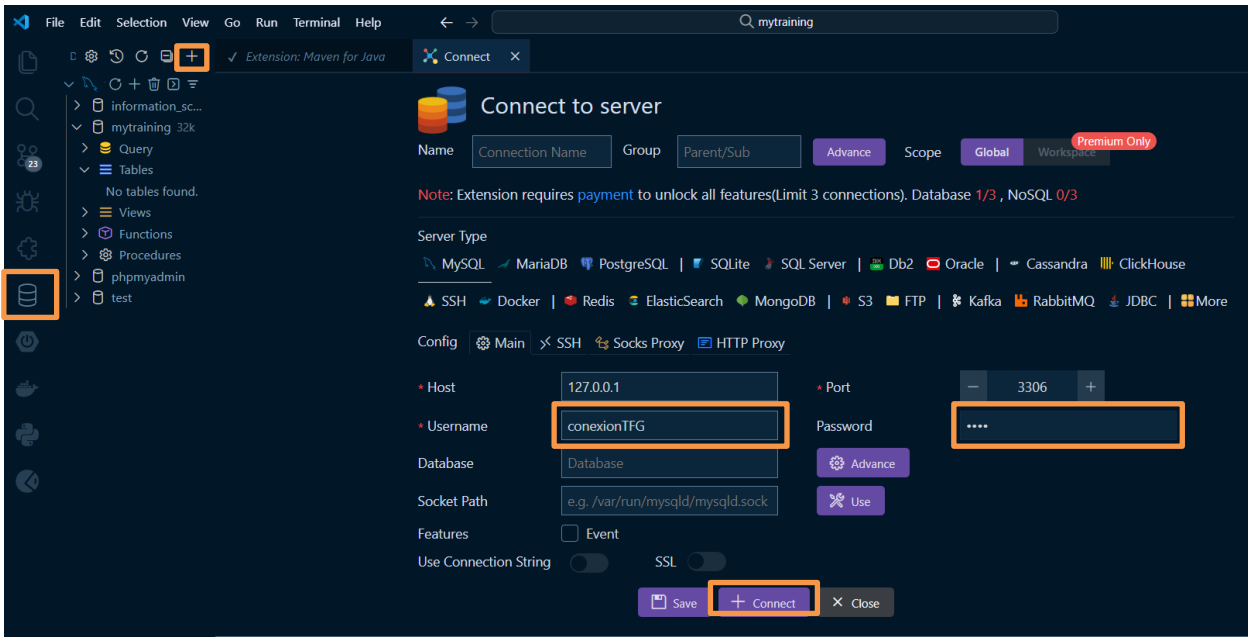


Figura 84: Extensión Database de Visual Studio Code

Una vez realizada la conexión con la base de datos, hay que pulsar sobre la extensión de “Spring Boot Dashboard”, cuyo icono es una forma hexagonal con un botón de encendido en medio.

Una vez dentro, simplemente hay que pulsar sobre el botón de “play” (triángulo tumbado) remarcado en naranja. Esto bastará para iniciar el componente Spring.

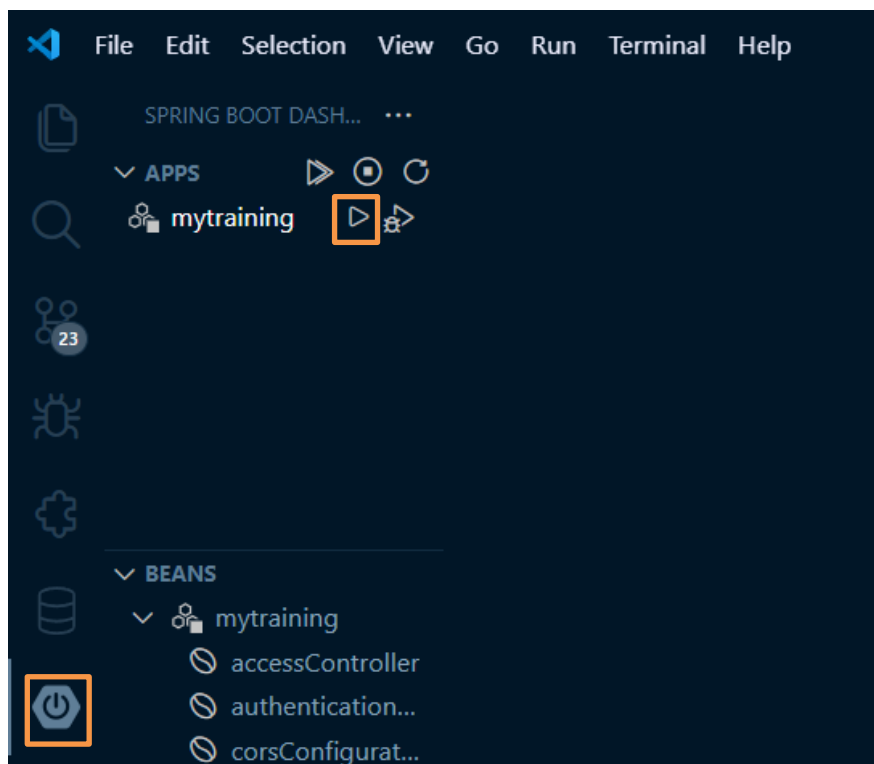


Figura 85: Extensión Spring Boot Dashboard de Visual Studio Code

E) Instalación de Docker Desktop y puesta en funcionamiento del container

Para instalar Docker Desktop hay que ingresar en la [página oficial](#) y pulsar en el botón “Download for Windows”, tal y como aparece en la Fig. 86.

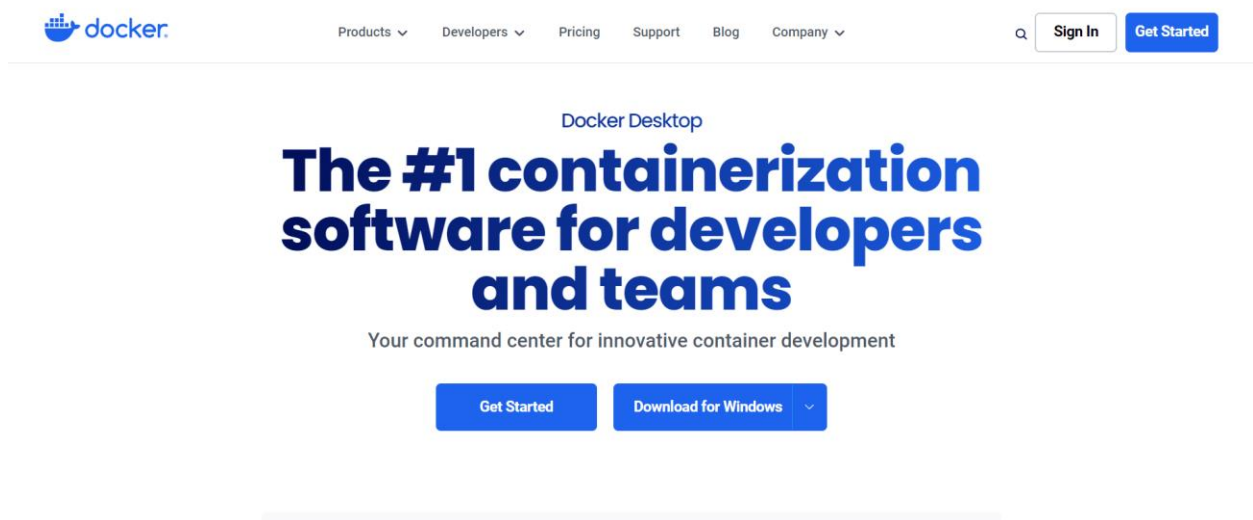


Figura 86: Página oficial de Docker

Una vez esté instalado, solo hay que iniciarlo para que Docker funcione en el sistema.

Con Docker Desktop iniciado, hay que irse a la terminal, situarse en el directorio de trabajo del “prediction-container”, el cual incluye todos los archivos necesarios para construir el contenedor de Docker para el microservicio Python.

Una vez en la terminal, hay que escribir los dos comandos que aparecen en la Fig. 87.

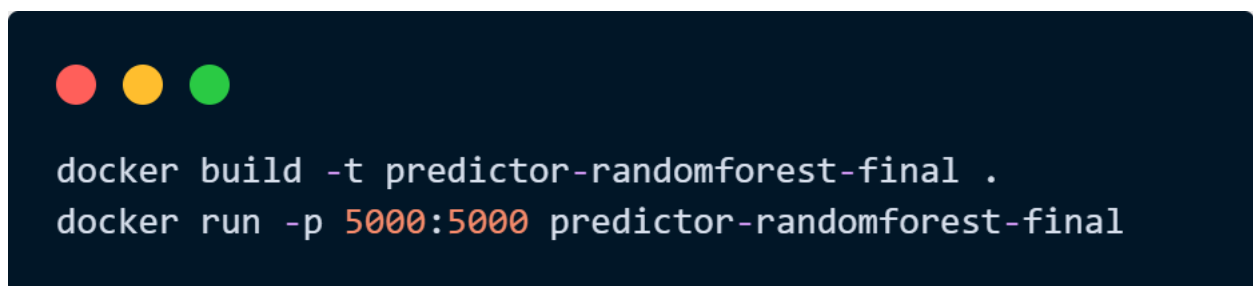


Figura 87: Comandos para construir y desplegar el contenedor Docker

Con todo desplegado, simplemente hay que entrar desde Google Chrome en la dirección localhost:8080/login y ya se puede usar la aplicación.

REFERENCIAS

[1] Díaz Lora, A. J. (2014). Aplicación android para la rehabilitación física de usuario usando MySQL, PHP, y codificación JSON. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/12197/>

[2] García Piosa, J. (2014). Personal trainer, entrenador personal para android. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/12168/>

[3] Franco Maireles, M. (2015). Aplicación android de rutinas de entrenamiento adaptadas al usuario usando SQLite y JSON. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/12316/>

[4] Valverde Baena, J. M. (2015). Desarrollo de servicio web RESTful para el acceso a base de datos de entrenamiento desde android. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/12304/>

[5] Díaz Romero, F. J. (2017). Plataforma web para la creación de rutinas de entrenamiento para la app RutinaApp. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/91247/>

[6] García Corredera, L. (2018). Aplicación móvil de prescripción de ejercicio con acceso a servicio web Spring. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/91750/>

[7] Carmona Rebollo, P. (2018). Plataforma web de prescripción de ejercicios usando Spring. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/91731/>

[8] Machuca Romero, M. R. (2022). Desarrollo de una aplicación web para la gestión de rutinas de entrenamiento personalizado. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/94414/>

[9] Rufo López, J. A. (2023). Actualización y mejora de la plataforma web de prescripción de ejercicios físicos MyTraining usando Spring y la integración de la pila ELK. Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

[Consultado el 2 de mayo de 2024]. Disponible en: <https://biblus.us.es/bibing/proyectos/abreproy/94537/>

[10] Documentación de Docker: <https://docs.docker.com/build/building/packaging/>

[11] En relación a Spring

[11][1] Spring Boot: <https://spring.io/guides/gs/spring-boot>

[11][2] Spring Security: <https://spring.io/projects/spring-security>

[11][3] Guía de Spring Security: <https://spring.io/guides/gs/securing-web>

[11][4] SpringDoc: <https://springdoc.org/>

[11][5] Página oficial de Spring: <https://spring.io/>

[12] En relación a Maven

[12][1] Página oficial de Maven: <https://maven.apache.org>

[12][2] Repositorio de Maven: <https://mvnrepository.com/>

[13] En relación a Hibernate

[13][1] Página oficial de Hibernate: <https://hibernate.org/>

[13][2] Guía de Hibernate: https://docs.jboss.org/hibernate/orm/6.5/quickstart/html_single/

[14] En relación a JSON

[14][1] Página oficial de JSON: <https://www.json.org/json-es.html>

[15] En relación a MySQL

[15][1] Documentación oficial de MySQL: <https://dev.mysql.com/doc/>

[16] En relación a Python

[16][1] Python Basics: <https://pythonbasics.org/>

[16][2] Documentación de la librería Scikit-Learn: <https://scikit-learn.org/>

[16][4] Documentación oficial de la librería Pandas: <https://pandas.pydata.org/>

[17] En relación a HTML y desarrollo web

[17][1] W3Schools: <https://www.w3schools.com/>

[17][2] Página oficial de Swagger: <https://swagger.io/>

[17][3] Documentación sobre HTML: <https://html.spec.whatwg.org/>

[17][4] Documentación sobre Bootstrap: <https://developer.mozilla.org/>

[19] Página oficial de la API Rest: <https://restfulapi.net/>

[20] En relación a recursos utilizados en el programa:

[20][1] Plantilla de Bootstrap: <https://html.design/download/neogym-gym-website-template/>

[20][2] *Dataset* de Kaggle: <https://www.kaggle.com/code/anetakovacheva/exploring-and-modelling-obesity-dataset/input>

[21] Otras Fuentes:

[21][1] Documentación oficial de JWT: <https://jwt.io/>

[21][2] IBM - Información sobre Random Forest: <https://www.ibm.com/>

[21][3] GeeksforGeeks – Patrón MVC: <https://www.geeksforgeeks.org/mvc-framework-introduction/>

[21][4] EdX: <https://www.edx.org/>

[21][5] Página oficial de XAMPP: <https://www.apachefriends.org/>

[21][6] Información sobre GitHub: <https://blog.hubspot.com/website/what-is-github-used-for#what-github>

[21][7] Stack overflow: <https://stackoverflow.co/>

[21][8] YouTube: <https://www.youtube.com/>