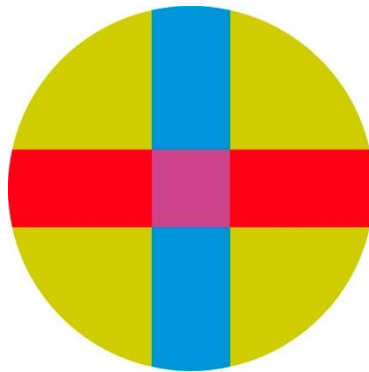


UNIVERSIDAD SAN PABLO - CEU

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN



TRABAJO FIN DE GRADO

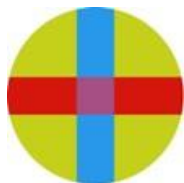
**Desarrollo y encapsulación en un
servicio REST de un sistema inteligente**

***Development and encapsulation of an
intelligent system in a REST service***

Autor: Juan Pablo Zurita Soto

Tutor: Mariano Fernández López

Septiembre 2024



UNIVERSIDAD SAN PABLO-CEU

ESCUELA POLITÉCNICA SUPERIOR

División de Ingeniería

Calificación del Trabajo Fin de Grado

Datos del alumno

NOMBRE: JUAN PABLO ZURITA SOTO

Datos del Trabajo

TÍTULO DEL PROYECTO: Desarrollo y encapsulación en un servicio REST de un sistema inteligente

Tribunal calificador

PRESIDENTE:

FDO.:

SECRETARIO:

FDO.:

VOCAL:

FDO.:

Reunido este tribunal el ____/____/_____, acuerda otorgar al Trabajo Fin de Grado presentado por D./Dña. _____ la calificación de _____

Resumen

Este proyecto consiste en una API *RESTful* desarrollada con Node.js y Express que gestiona datos de equipos, jugadores y partidos de *LaLiga*, donde está encapsulado un modelo inteligente, basado en aprendizaje automático, de predicción de resultados. La API permite realizar operaciones CRUD sobre distintos *endpoints*, proporcionando rutas protegidas para usuarios autenticados y roles administrativos.

El modelo de predicción, desarrollado en *Python*, analiza múltiples estadísticas de los partidos para anticipar el resultado, ya sea victoria local, visitante o empate. Emplea técnicas avanzadas de *machine learning* y puede ser actualizado y reentrenado mediante puntos de acceso dedicados en la API.

El despliegue del proyecto está diseñado principalmente para la nube, utilizando AWS para gestionar cada microservicio de manera independiente: la API, el modelo de predicción y la base de datos MongoDB. Sin embargo, también es posible desplegarlo localmente mediante Docker, que encapsula los servicios en contenedores para un entorno controlado y replicable.

Palabras Clave

API RESTful

Modelo predictivo

Aprendizaje automático

Endpoints

Predicciones

Arquitectura de microservicios

CRUD

Precisión

Validación cruzada

Datos de entrada

Rutas

Abstract

This project consists of a RESTful API developed with Node.js and Express that manages LaLiga teams, players and matches data, where an intelligent model, based on machine learning, for predicting results is encapsulated. The API allows CRUD operations on different endpoints, providing protected routes for authenticated users and administrative roles.

The prediction model, developed in Python, analyzes multiple match statistics to anticipate the outcome, whether it is a home win, away win or draw. It employs advanced machine learning techniques and can be updated and retrained via dedicated API access points.

The project deployment is primarily designed for the cloud, using AWS to manage each microservice independently: the API, the prediction model and the MongoDB database. However, it is also possible to deploy it locally using Docker, which encapsulates the services in containers for a controlled and replicable environment.

Keywords

RESTful API

Predictive Modeling

Machine Learning

Endpoints

Predictions

Microservices architecture

CRUD

Accuracy

Cross validation

Input data

Routes

Índice de contenidos

Capítulo 1 Introducción	1
1.1 Contexto del TFG	1
1.2 Objetivos.....	2
1.3 Organización del trabajo.....	3
Capítulo 2 Gestión del proyecto	5
2.1 Modelo de ciclo de vida.....	5
2.2 Planificación.....	5
2.3 Estimación.....	10
Capítulo 3 Análisis.....	17
3.1 Especificación de requisitos.....	17
3.2 Análisis de los Casos de Uso	21
Capítulo 4 Diseño.....	23
4.1 Arquitectura del sistema	23
4.1.1 Arquitectura lógica	23
4.1.2 Arquitectura física	24
4.2 Diseño del servicio REST	26
4.2.1 Funcionalidades del servicio REST.....	27
4.2.2 Descripción de los Endpoints	28
4.3 Diseño de datos	29
4.3.1 Fuente de datos.....	29
4.3.2 Migración y carga inicial de datos	31
4.3.3 Modelo Entidad-Relación simplificado.....	31
4.4 Diseño del servicio de pronósticos	32
4.4.1 Transformación de datos	32
4.4.2 Selección de algoritmo de aprendizaje	33
Capítulo 5 Implementación y validación del sistema	36
5.1 Entorno de construcción.....	36
5.2 Referencia al repositorio de software	39
5.3 Plan de pruebas	40
5.4 Evaluación del sistema.....	42

Capítulo 6 Conclusiones y líneas futuras	48
6.1 Conclusiones	48
6.2 Líneas futuras.....	49
Bibliografía.....	50
Anexo I – OpenAPI	51
Anexo II – Scripts migración y carga de datos	106

Índice de figuras

Figura 1 - Planificación del proyecto a día 20 de febrero de 2024	14
Figura 2 - Planificación del Proyecto el 15 de septiembre de 2024	16
Figura 3 - Diagrama de Casos de uso de la API a realizar	21
Figura 4 - Arquitectura lógica del sistema	23
Figura 5 - Arquitectura física del sistema	25
Figura 6 - Arquitectura REST (Fuente: https://aprendiendoarduino.wordpress.com/2019/10/27/api-rest/)	26
Figura 7 - Arquitectura API Football (Fuente: https://www.api-football.com/documentation-v3#section/Architecture)	31
Figura 8 - Modelo Entidad-Relación de Base de Datos API (simplificado)	32
Figura 9 - Logistic Regression (Fuente: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/)	34
Figura 10 - Resultados de Logistic Regression	34
Figura 11 - IDE VS Code	36
Figura 12 - Docker Desktop	37
Figura 13 - Consulta MongoDB	38
Figura 14 - Instancias AWS	39
Figura 15 - Grupos de seguridad AWS	39
Figura 16 - Tests unitarios de modelo de aprendizaje correctos	40
Figura 17 - Tests rutas servicio REST e integración con modelo	41
Figura 18 - Parámetros del grupo de hilos en JMeter	41
Figura 19 – Consultas a realizar	42

Figura 20 - Resultados de JMeter	42
Figura 21 - Entrenamiento del modelo desde la máquina ML.....	43
Figura 22 - Servicio ML desplegado	43
Figura 23 - Servicio web levantado	43
Figura 24 - Login de administrador.....	44
Figura 25 - Token de usuario en authorization para tener acceso a una ruta protegida	45
Figura 26 - Pronóstico empate	45
Figura 27 - Pronóstico victoria local	46
Figura 28 - Pronóstico victoria visitante	47

Índice de tablas

<i>Tabla 1 - Actividades y Tareas del Proyecto</i>	6
Tabla 2 - Estados de las tareas del proyecto a día 20 de febrero de 2024	10
<i>Tabla 3 - Estimaciones de Esfuerzo y Duración para cada tarea del Proyecto a día 20 de febrero de 2024</i>	12
Tabla 4 - Requisitos funcionales	18
Tabla 5 - Requisitos de rendimiento.....	19
Tabla 6 - Requisitos de seguridad.....	19
Tabla 7 - Requisitos de Interoperabilidad	20
Tabla 8 - Requisitos sobre el entorno tecnológico	20

Capítulo 1

Introducción

1.1 Contexto del TFG

El presente Trabajo de Fin de Grado se enmarca en el ámbito de la inteligencia artificial (IA) y los servicios REST, dos áreas que constan de una importancia notoria en la manera en la que interactuamos con los sistemas informáticos. La inteligencia artificial, ha demostrado ser una herramienta muy valiosa para la toma de decisiones basada en datos. Una rama clave de la IA es el aprendizaje automático (*machine learning*), que permite a las máquinas identificar patrones y realizar predicciones a partir de grandes volúmenes de información sin intervención humana directa. A medida que se dispone de más datos, estos modelos pueden ajustarse y mejorar su capacidad predictiva. Esto tiene aplicaciones prácticas en diversos sectores, como la medicina, las finanzas y, en este caso, el deporte. En el contexto deportivo, el aprendizaje automático permite analizar datos históricos de los partidos para predecir posibles resultados, lo que resulta útil tanto para entrenadores y analistas como para aficionados.

El aprendizaje automático, en este caso, emplea algoritmos que modelan la relación entre diversas características de los partidos (número de tiros, pases, goles esperados, etc.) y los resultados obtenidos. Estos modelos predictivos permiten determinar el resultado de los encuentros, como victorias locales, empates o victorias visitantes, y son fundamentales en la toma de decisiones estratégicas. La capacidad de ajustar estos modelos a partir de nuevos datos garantiza que las predicciones evolucionen con el tiempo, manteniendo su relevancia.

Por otro lado, los servicios REST (*Representational State Transfer*) han revolucionado el desarrollo de aplicaciones distribuidas, proporcionando una arquitectura ligera y escalable que permite la interacción entre diferentes sistemas. REST es ampliamente utilizado en la construcción APIs (Interfaces de Programación de Aplicaciones) que permiten la comunicación entre clientes y servidores de manera eficiente y segura. En este contexto, REST facilita la exposición de servicios como la consulta de datos deportivos o la ejecución de algoritmos predictivos. Esta arquitectura no solo mejora la eficiencia en el manejo y procesamiento de datos, sino que también ofrece una interfaz accesible para integrar modelos de *machine learning* sobre datos históricos de partidos y hacerlos disponibles a través de internet. De esta forma, los usuarios o sistemas pueden realizar consultas a los modelos predictivos y obtener resultados en tiempo real, lo que facilita el análisis y la toma de decisiones basadas en datos actualizados.

1.2 Objetivos

El objetivo principal es el desarrollo y encapsulación de un sistema inteligente de predicción de resultados deportivos en un servicio REST. Este sistema debe ser capaz de procesar datos históricos de partidos de fútbol y generar predicciones automatizadas mediante el uso de algoritmos de aprendizaje automático. Para lograr este objetivo, se plantean los siguientes subobjetivos:

1. **Desarrollo del modelo predictivo basado en *machine learning*:** crear un modelo de predicción utilizando técnicas de aprendizaje automático, capaz de predecir el resultado de un partido de fútbol (victoria local, empate o victoria visitante) en función del historial de partidos.
2. **Entrenamiento y ajuste del modelo:** recopilar y procesar datos históricos de partidos para entrenar el modelo predictivo. Se deberá ajustar el modelo para optimizar su precisión y garantizar que las predicciones sean lo más fiables posible.
3. **Implementación de la API REST:** desarrollar un servicio REST que permita a los usuarios interactuar con el modelo predictivo. Esto incluirá la creación de rutas

para realizar consultas, obtener predicciones y gestionar los datos de los partidos, equipos y jugadores.

4. **Seguridad y gestión de usuarios:** implementar un sistema de autenticación y roles que permita proteger ciertas rutas API, de manera que solo los usuarios autorizados puedan acceder a las funcionalidades de administración o actualización del modelo.
5. **Despliegue del sistema en un entorno *cloud*:** implementar el despliegue del servicio en un entorno de nube (AWS), asegurando escalabilidad, disponibilidad y eficiencia de los servicios.
6. **Validación y pruebas del sistema:** realizar pruebas exhaustivas del modelo predictivo y del servicio REST para validar su correcto funcionamiento, tanto en entornos locales (mediante Docker) como en la nube.

1.3 Organización del trabajo

Este trabajo se dividirá en 6 capítulos, además de los correspondientes anexos que detallan información adicional.

- **Capítulo 1 – Introducción:** se relata el contexto sobre la temática de TFG, los objetivos que se deben cumplir y la organización del trabajo
- **Capítulo 2 – Gestión del proyecto:** se comenta la planificación realizada, con sus correspondientes actividades y tareas con una estimación de esfuerzo y seguimiento de estas, aportando la duración real de cada una.
- **Capítulo 3 – Análisis:** se detalla la especificación de requisitos, así como los casos de uso.
- **Capítulo 4 – Diseño:** se desarrolla la arquitectura física y lógica del sistema, el diseño de datos, el diseño del servicio REST y el algoritmo de aprendizaje empleado.
- **Capítulo 5 – Implementación y validación del sistema:** se analiza la implementación de los distintos servicios y se validan los requisitos de este.

- **Capítulo 6 – Conclusiones y líneas futuras:** se determinan las conclusiones adquiridas tras la finalización del proyecto y se exponen las distintas áreas en las cuales se podría mejorar el rendimiento de este.
- En los anexos se adjunta el OpenAPI y los scripts de migración y carga de datos

Capítulo 2

Gestión del proyecto

En el presente capítulo se aborda la fase de estimación y planificación del proyecto. En esta sección, se detallan los procesos y metodologías utilizadas para estimar la duración de las tareas y desarrollar un cronograma detallado que guiará la ejecución del proyecto. La planificación adecuada no solo garantiza una distribución eficiente de los recursos disponibles, sino que también permite anticipar posibles desafíos y riesgos, facilitando así la toma de decisiones informadas y la adaptación a los cambios que puedan surgir durante el desarrollo del TFG.

2.1 Modelo de ciclo de vida

En relación con el modelo de ciclo de vida del proyecto, se ha optado por un enfoque tradicional basado en la técnica de cascada (*Waterfall*) combinado con metodologías modernas de seguimiento y control.

Las fases del proyecto no se revisan hasta que se completan. Es útil cuando se busca el control sobre el avance del proyecto y se tienen los requisitos bien definidos desde el principio. Sin embargo, esta metodología puede carecer de flexibilidad ante proyectos muy cambiantes.

2.2 Planificación

En cuanto a la planificación del proyecto, como evidencia la *Tabla 1*, se ha definido una serie de actividades principales, donde algunas de las cuales se desglosan en sus tareas respectivas. Estas actividades son necesarias para la ejecución del trabajo asignado, y se llevarán a cabo de la siguiente manera:

Tabla 1 - Actividades y Tareas del Proyecto

Actividades	Tareas
Elaborar el anteproyecto	
Planificación del proyecto	
	<i>Planificar el proyecto</i>
	<i>Documentar la planificación del proyecto</i>
Elaborar especificación de requisitos	
Diseño de la arquitectura del sistema	
	<i>Elaborar la arquitectura del sistema</i>
	<i>Documentar la arquitectura del sistema</i>
Búsqueda fuente de datos	
	<i>Buscar fuente de datos</i>
	<i>Documentar fuente de datos</i>
Diseño del servicio REST	
	<i>Diseñar el servicio REST</i>
	<i>Documentar el diseño del servicio REST</i>
Transformación de la fuente de datos	
	<i>Transformar fuente de datos</i>
	<i>Documentar transformación de la fuente de datos</i>
Selección del algoritmo de aprendizaje	
	<i>Seleccionar algoritmo de aprendizaje</i>
	<i>Documentar selección del algoritmo de aprendizaje</i>
Aprendizaje del modelo	
	<i>Llevar a cabo el proceso de aprendizaje</i>
	<i>Realizar pruebas unitarias del modelo aprendido</i>
	<i>Documentar tanto el modelo como las pruebas</i>
Encapsulación del modelo en servicio REST	
	<i>Implementar el servicio REST</i>
	<i>Realizar pruebas unitarias del servicio REST</i>
	<i>Realizar pruebas de integración con el modelo aprendido</i>
	<i>Realizar las pruebas del sistema antes del despliegue</i>
	<i>Documentar tanto el servicio REST como las pruebas</i>
Despliegue del servicio en entorno de producción	
	<i>Desplegar el servicio</i>
	<i>Realizar las pruebas del sistema después del despliegue</i>
	<i>Documentar tanto el despliegue como las pruebas</i>
Terminar memoria	
Realizar depósito	
Preparar presentación	
Realizar presentación	

- **Elaborar el anteproyecto:** esta actividad implica la redacción del documento inicial que describe los aspectos básicos del proyecto, como el título, el área temática, los objetivos y la justificación técnica.
- **Planificación del proyecto**

- *Planificar el proyecto:* se definen los pasos y recursos necesarios para llevar a cabo el proyecto.
 - *Documentar la planificación:* se registran detalladamente los resultados de la planificación, incluyendo el cronograma.
- **Elaborar especificación de requisitos:** esta actividad implica la identificación y documentación de los requisitos funcionales y no funcionales del sistema a desarrollar.
- **Diseño de la arquitectura del sistema**
 - *Elaborar arquitectura:* se define la estructura general del sistema, incluyendo componentes, interfaces y relaciones.
 - *Documentar la arquitectura:* se registra los detalles del diseño arquitectónico para su futura referencia y comunicación.
- **Búsqueda fuente de datos**
 - *Buscar la fuente de datos:* se identifican y recopilan las fuentes de datos necesarias para el desarrollo del sistema.
 - *Documentar la fuente de datos:* se registra la información relevante sobre las fuentes de datos encontradas, como su origen, formato y disponibilidad.
- **Diseño del servicio REST**
 - *Diseñar el servicio REST:* se define los *endpoints*, métodos HTTP y estructura del servicio REST.
 - *Documentar diseño REST:* se registran los detalles del diseño del servicio REST para su implementación y referencia futura.
- **Transformación de la fuente de datos**
 - *Transformar fuente de datos:* se realizan las operaciones necesarias para adaptar las fuentes de datos a los requisitos del sistema.
 - *Documentar la transformación de la fuente de datos:* se registran los pasos y resultados de la transformación de datos para su validación y reproducibilidad.
- **Selección algoritmo de aprendizaje**

- *Seleccionar algoritmo*: se analiza y elige el algoritmo de aprendizaje más adecuado para el problema a resolver.
- *Documentar el algoritmo elegido*: se describe el algoritmo seleccionado, sus características y su justificación para su uso en el proyecto.
- **Aprendizaje del modelo**
 - *Llevar a cabo el proceso de aprendizaje*: se entrena el modelo utilizando los datos disponibles.
 - *Realizar pruebas unitarias del modelo aprendido*: se verifica el funcionamiento correcto del modelo entrenado en diferentes escenarios.
 - *Documentar modelo y las pruebas*: se registran los detalles del modelo entrenado y los resultados de las pruebas unitarias.
- **Encapsulación del modelo en servicio REST**
 - *Implementar servicio REST*: se desarrolla e implementa el servicio REST que encapsula el modelo de aprendizaje.
 - *Realizar pruebas unitarias del servicio REST*: se verifica la funcionalidad del servicio REST de forma aislada.
 - *Realizar pruebas de integración con el modelo aprendido*: se evalúa la integración entre el servicio REST y el modelo de aprendizaje.
 - *Realizar las pruebas del sistema antes del despliegue*: se verifica el funcionamiento global del sistema antes de su implementación en un entorno de producción.
 - *Documentar servicio REST y las pruebas*: se registran los detalles del servicio REST implementado y los resultados de las pruebas realizadas.
- **Despliegue del servicio en entorno de producción**
 - *Desplegar el servicio*: se instala el servicio en un entorno de producción para su uso real.
 - *Realizar las pruebas del sistema después del despliegue*: se verifica el correcto funcionamiento del sistema en el entorno de producción.

- *Documentar tanto el despliegue como las pruebas:* se registran los detalles del despliegue del sistema y los resultados de las pruebas realizadas en el entorno de producción.
- **Terminar memoria:** esta tarea implica la finalización y revisión del documento de memoria del TFG.
- **Realizar depósito:** se realiza el depósito del TFG en la plataforma o institución correspondiente para su evaluación y archivo.
- **Preparar presentación:** se preparan los materiales y contenidos necesarios para la presentación oral del TFG.
- **Realizar presentación:** se presenta oralmente el TFG ante el tribunal evaluador, exponiendo los aspectos más relevantes del trabajo realizado.

Según se observa en la *Tabla 2*, cada tarea se gestiona según su estado, reflejado en la columna correspondiente de la tabla. Las tareas pendientes se clasifican como '**To do**', las que están en proceso como '**Doing**', y las finalizadas como '**Done**'. La elaboración del anteproyecto fue la primera tarea completada, marcada como '**Done**' al finalizar.

En cuanto a la planificación, las tareas están actualmente en proceso, mientras que el resto de las actividades del proyecto están pendientes de realizarse.

Tabla 2 - Estados de las tareas del proyecto a día 20 de febrero de 2024

Actividades	Tareas	To do	Doing	Done
Elaborar el anteproyecto				
Planificación del proyecto				
	Planificar el proyecto			
	Documentar la planificación del proyecto			
Elaborar especificación de requisitos				
Diseño de la arquitectura del sistema				
	Elaborar la arquitectura del sistema			
	Documentar la arquitectura del sistema			
Búsqueda fuente de datos				
	Buscar fuente de datos			
	Documentar fuente de datos			
Diseño del servicio REST				
	Diseñar el servicio REST			
	Documentar el diseño del servicio REST			
Transformación de la fuente de datos				
	Transformar fuente de datos			
	Documentar transformación de la fuente de datos			
Selección del algoritmo de aprendizaje				
	Seleccionar algoritmo de aprendizaje			
	Documentar selección del algoritmo de aprendizaje			
Aprendizaje del modelo				
	Llevar a cabo el proceso de aprendizaje			
	Realizar pruebas unitarias del modelo aprendido			
	Documentar tanto el modelo como las pruebas			
Encapsulación del modelo en servicio REST				
	Implementar el servicio REST			
	Realizar pruebas unitarias del servicio REST			
	Realizar pruebas de integración con el modelo aprendido			
	Realizar las pruebas del sistema antes del despliegue			
	Documentar tanto el servicio REST como las pruebas			
Despliegue del servicio en entorno de producción				
	Desplegar el servicio			
	Realizar las pruebas del sistema después del despliegue			
	Documentar tanto el despliegue como las pruebas			
Terminar memoria				
Realizar depósito				
Preparar presentación				
Realizar presentación				

2.3 Estimación

En esta sección se analiza el proceso esencial de calcular el esfuerzo y el tiempo requeridos para llevar a cabo el proyecto. Una estimación precisa permite prever desafíos y adaptarse a los cambios durante su desarrollo.

Para cada una de estas tareas, se estima un nivel de esfuerzo, representado por valores de **1**, **2** o **3**, los cuales indican la dificultad esperada, tal y como se muestra en la *Tabla 3*.

El valor '1' corresponde a una tarea considerada fácil, el '2' indica una tarea sencilla pero menos que el nivel anterior, y el '3' señala una tarea que podría presentar dificultades. Estas estimaciones de esfuerzo no se traducen directamente en días, sino que representan puntos de función. Inicialmente, se estima que la velocidad de realización será de un punto de función por día, aunque esta estimación se ajustará a medida que avance el proyecto.

La duración de cada tarea se medirá en días, dependiendo del esfuerzo estimado y de la velocidad de ejecución. La estimación inicial de duración se registrará en la tabla como una referencia para comparar con la duración real a medida que se completen las tareas. La duración real de las tareas se asignará en la columna respectiva a medida que se completen, como se observa en la *Tabla 3*. A la hora de computar un día en la duración de una actividad, se requiere un tiempo empleado de **3 horas** en la misma.

Tabla 3 - Estimaciones de Esfuerzo y Duración para cada tarea del Proyecto a día 20 de febrero de 2024

Actividades	Tareas	Esfuerzo estimado	Duración estimada en días	Duración real en días
Elaborar el anteproyecto		1	1	1
Planificación del proyecto				
	<i>Planificar el proyecto</i>	1	1	3
	<i>Documentar la planificación del proyecto</i>	1	1	1
Elaborar especificación de requisitos		2	2	
Diseño de la arquitectura del sistema				
	<i>Elaborar la arquitectura del sistema</i>	1	1	
	<i>Documentar la arquitectura del sistema</i>	1	1	
Búsqueda fuente de datos				
	<i>Buscar fuente de datos</i>	3	3	
	<i>Documentar fuente de datos</i>	1	1	
Diseño del servicio REST				
	<i>Diseñar el servicio REST</i>	3	3	
	<i>Documentar el diseño del servicio REST</i>	2	2	
Transformación de la fuente de datos				
	<i>Transformar fuente de datos</i>	3	3	
	<i>Documentar transformación de la fuente de datos</i>	2	2	
Selección del algoritmo de aprendizaje				
	<i>Seleccionar algoritmo de aprendizaje</i>	3	3	
	<i>Documentar selección del algoritmo de aprendizaje</i>	2	2	
Aprendizaje del modelo				
	<i>Llevar a cabo el proceso de aprendizaje</i>	3	3	
	<i>Realizar pruebas unitarias del modelo aprendido</i>	2	2	
	<i>Documentar tanto el modelo como las pruebas</i>	2	2	
Encapsulación del modelo en servicio REST				
	<i>Implementar el servicio REST</i>	3	3	
	<i>Realizar pruebas unitarias del servicio REST</i>	2	2	
	<i>Realizar pruebas de integración con el modelo aprendido</i>	2	2	
	<i>Realizar las pruebas del sistema antes del despliegue</i>	2	2	
	<i>Documentar tanto el servicio REST como las pruebas</i>	2	2	
Despliegue del servicio en entorno de producción				
	<i>Desplegar el servicio</i>	3	3	
	<i>Realizar las pruebas del sistema después del despliegue</i>	3	3	
	<i>Documentar tanto el despliegue como las pruebas</i>	2	2	
Terminar memoria		2	2	
Realizar depósito		1	1	
Preparar presentación		2	2	
Realizar presentación		1	1	
	TOTAL ESFUERZO ESTIMADO	58		

En la *Figura 1* se presenta una visión general del *portfolio* de tareas, incluyendo todos los campos mencionados anteriormente.

PLANIFICACIÓN PROYECTO							
Actividades	Tareas	Esfuerzo estimado	Duración estimada en días	Duración real en días	To do	Doing	Done
Elaborar el anteproyecto		1	1	1			
Planificación del proyecto							
	Planificar el proyecto	1	1	3			
	Documentar la planificación del proyecto	1	1	1			
Elaborar especificación de requisitos		2	2				
Diseño de la arquitectura del sistema							
	Elaborar la arquitectura del sistema	1	1				
	Documentar la arquitectura del sistema	1	1				
Búsqueda fuente de datos							
	Buscar fuente de datos	3	3				
	Documentar fuente de datos	1	1				
Diseño del servicio REST							
	Diseñar el servicio REST	3	3				
	Documentar el diseño del servicio REST	2	2				
Transformación de la fuente de datos							
	Transformar fuente de datos	3	3				
	Documentar transformación de la fuente de datos	2	2				
Selección del algoritmo de aprendizaje							
	Seleccionar algoritmo de aprendizaje	3	3				
	Documentar selección del algoritmo de aprendizaje	2	2				
Aprendizaje del modelo							
	Llevar a cabo el proceso de aprendizaje	3	3				
	Realizar pruebas unitarias del modelo aprendido	2	2				
	Documentar tanto el modelo como las pruebas	2	2				
Encapsulación del modelo en servicio REST							
	Implementar el servicio REST	3	3				
	Realizar pruebas unitarias del servicio REST	2	2				
	Realizar pruebas de integración con el modelo aprendido	2	2				
	Realizar las pruebas del sistema antes del despliegue	2	2				
	Documentar tanto el servicio REST como las pruebas	2	2				
Despliegue del servicio en entorno de producción							
	Desplegar el servicio	3	3				
	Realizar las pruebas del sistema después del despliegue	3	3				
	Documentar tanto el despliegue como las pruebas	2	2				
Terminar memoria		2	2				
Realizar depósito		1	1				
Preparar presentación		2	2				
Realizar presentación		1	1				
	TOTAL ESFUERZO ESTIMADO	58					

Figura 1 - Planificación del proyecto a día 20 de febrero de 2024

En la fecha cercana a la finalización del proyecto, el documento de planificación, como se puede observar en la *Figura 2* , muestra que la duración real ha sido considerablemente superior a la estimada, 92,5 días reales frente a los 58 días estimados, en torno a un 60% más. Trasladando ese dato a horas esto supone, en estos momentos, una duración de 277,5 horas dedicadas a las correspondientes tareas del proyecto.

PLANIFICACIÓN PROYECTO							
Actividades	Tareas	Esfuerzo estimado	Duración estimada en días	Duración real en días	To do	Doing	Done
Elaborar el anteproyecto		1	1	1			
Planificación del proyecto							
	Planificar el proyecto	1	1	6			
	Documentar la planificación del proyecto	1	1	5			
Elaborar especificación de requisitos		2	2	9,5			
Diseño de la arquitectura del sistema							
	Elaborar la arquitectura del sistema	1	1	8,5			
	Documentar la arquitectura del sistema	1	1	6			
Búsqueda fuente de datos							
	Buscar fuente de datos	3	3	5			
	Documentar fuente de datos	1	1	1			
Diseño del servicio REST							
	Diseñar el servicio REST	3	3	5,5			
	Documentar el diseño del servicio REST	2	2	2,5			
Transformación de la fuente de datos							
	Transformar fuente de datos	3	3	7			
	Documentar transformación de la fuente de datos	2	2	2,5			
Selección del algoritmo de aprendizaje							
	Seleccionar algoritmo de aprendizaje	3	3	6			
	Documentar selección del algoritmo de aprendizaje	2	2	2,5			
Aprendizaje del modelo							
	Llevar a cabo el proceso de aprendizaje	3	3	3,5			
	Realizar pruebas unitarias del modelo aprendido	2	2	2			
	Documentar tanto el modelo como las pruebas	2	2	1			
Encapsulación del modelo en servicio REST							
	Implementar el servicio REST	3	3	4			
	Realizar pruebas unitarias del servicio REST	2	2	1,5			
	Realizar pruebas de integración con el modelo aprendido	2	2	1			
	Realizar las pruebas del sistema antes del despliegue	2	2	2			
	Documentar tanto el servicio REST como las pruebas	2	2	2,5			
Despliegue del servicio en entorno de producción							
	Desplegar el servicio	3	3	2			
	Realizar las pruebas del sistema después del despliegue	3	3	2			
	Documentar tanto el despliegue como las pruebas	2	2	1			
Terminar memoria		2	2	2			
Realizar depósito		1	1				
Preparar presentación		2	2				
Realizar presentación		1	1				
	TOTAL ESFUERZO ESTIMADO	58					

Velocidad estimada en puntos de función/día =	1
Tiempo estimado del proyecto	58 días
Tiempo real acumulado	92,5 días
	11,6 semanas
	2,9 meses

Contador de horas día	2
-----------------------	---

Figura 2 - Planificación del Proyecto el 15 de septiembre de 2024

Capítulo 3

Análisis

3.1 Especificación de requisitos

En el presente capítulo, se aborda la especificación detallada de los requisitos para el desarrollo y encapsulación en un servicio REST de un sistema inteligente diseñado para realizar pronósticos de partidos de fútbol. El sistema utilizará técnicas avanzadas de aprendizaje automático para analizar datos históricos de partidos, proporcionando predicciones en base a estadísticas. El servicio web también proporciona información sobre los equipos, jugadores y partidos de la temporada 2023-2024 *de LaLiga* (Primera División de España).

Este proceso de especificación se fundamenta en la comprensión exhaustiva de las necesidades y expectativas del usuario, así como en los estándares y prácticas establecidas en el ámbito de los sistemas de información.

Por consiguiente, esta sección está estructurada de manera sistemática para abordar distintos aspectos clave como funcionalidades del sistema, rendimiento, seguridad, interoperabilidad y requisitos sobre el entorno tecnológico y de comunicaciones. Este enfoque garantiza que los requisitos estén alineados con los objetivos del proyecto y los estándares de calidad establecidos.

- **Funcionalidades del sistema**
 - **RF01 - Registro e inicio de sesión:** el sistema debe permitir a los usuarios crear una cuenta e iniciar sesión para acceder a funcionalidades personalizadas. Durante el registro, los usuarios deberán proporcionar un correo electrónico, un nombre de usuario

único y una contraseña. Además, se implementará un mecanismo de recuperación de contraseña para permitir a los usuarios restablecer sus contraseñas en caso de olvido, mediante un enlace enviado a su correo electrónico registrado.

- **RF02 – Consulta de los datos:** el sistema asegura que todos los usuarios, incluyendo los anónimos, aquellos sin registrar, podrán realizar consultas relacionadas con la obtención de datos (GET) de *LaLiga*.
- **RF03 – Pronósticos:** el sistema debe permitir que los usuarios registrados pueden acceder a la ruta de predicciones para enviar una solicitud con datos de entrada, y así obtener un pronóstico.
- **RF04 - Gestión de los datos:** el sistema permitirá que los usuarios administradores podrán añadir, modificar o eliminar los datos que el sistema utiliza para procesar las consultas de la información de la API.
- **RF05 - Procesamiento inteligente:** el sistema analizará y procesará los datos ingresados por el usuario para generar una respuesta inteligente, empleando técnicas de aprendizaje automático.

Tabla 4 - Requisitos funcionales

Ref.	Descripción
RF01	Registro e inicio de sesión
RF02	Consulta de los datos
RF03	Pronósticos
RF04	Gestión de los datos
RF05	Procesamiento inteligente

- **Rendimiento**

- **RE01 - Tiempos de respuesta para despliegue en local:** el sistema desplegado mediante contenedores (puesto que el balanceo de carga de

AWS no es gratuito) debe asegurar que el 90% de las solicitudes REST sean respondidas en menos de 2 segundos bajo condiciones normales de carga. La herramienta *JMeter* será utilizada para medir el rendimiento.

- **RE02 - Capacidad de carga en local:** el sistema desplegado mediante contenedores deberá ser capaz de manejar un mínimo de 350 solicitudes por minuto, garantizando que los tiempos de respuesta se mantengan por debajo de 3 segundos en todo momento.

Tabla 5 - Requisitos de rendimiento

Ref.	Descripción
RE01	Tiempos de respuesta
RE02	Capacidad de carga

- **Seguridad**

- **SE01 – Protección de contraseñas:** Las contraseñas se almacenarán de forma segura utilizando el algoritmo de *hashing bcrypt* para garantizar la protección de las credenciales de los usuarios.
- **SE02 - Autenticación robusta:** el sistema utilizará JWT (*JSON Web Tokens*) para asegurar que los usuarios son quienes dicen ser.
- **SE03 – Protección de rutas:** se garantizará la protección de los *endpoints* de la API mediante la verificación de tokens y la implementación de controles de acceso basados en roles para cada ruta y método CRUD de la misma. Los roles son: anónimos (usuarios sin registrar, usuarios registrados y usuarios administradores).

Tabla 6 - Requisitos de seguridad

Ref.	Descripción
SE01	Protección de contraseñas

SE02	Autenticación robusta
SE03	Protección de rutas

- **Interoperabilidad con otros sistemas**

- **IN01 – Compatibilidad con estándar JSON:** el sistema garantizará la interoperabilidad mediante JSON facilitando así una integración eficaz y fiable con sistemas externos que emplean estos formatos para el intercambio de datos.

Tabla 7 - Requisitos de Interoperabilidad

Ref.	Descripción
IN01	Compatibilidad con otros sistemas

- **Requisitos sobre entorno tecnológico y de comunicaciones**

- **TEC01 – Backend:** el servicio REST se desarrollará utilizando Node.js para el *backend*.
- **TEC02 – Gestor de bases de datos:** se utilizará el gestor de bases de datos MongoDB, elegido por su flexibilidad y facilidad de integración.
- **TEC03 – Despliegue:** el sistema se desplegará en el entorno de nube pública AWS asegurando escalabilidad y disponibilidad.
- **TEC04 - Comunicaciones:** se usará el protocolo HTTP para las comunicaciones del usuario con el servicio web.
- **TEC05 – Documentación de API REST:** el sistema estará documentado en *OpenAPI 3.0*, incluyendo endpoints, métodos HTTP, parámetros, respuestas, y será accesible públicamente en formato YAML.

Tabla 8 - Requisitos sobre el entorno tecnológico

Ref.	Descripción
------	-------------

TEC01	Backend
TEC02	Gestor de bases de datos
TEC03	Despliegue
TEC04	Comunicaciones
TEC05	Documentación API REST

3.2 Análisis de los Casos de Uso

El propósito de esta sección es examinar y evaluar los diversos escenarios de utilización del sistema, es decir, la manera en que los usuarios se relacionan con las características del sistema para alcanzar sus metas.

En la *Figura 3*, se hace referencia a los distintos casos de uso que se pueden llevar a cabo a través del servicio web que se desea implementar.

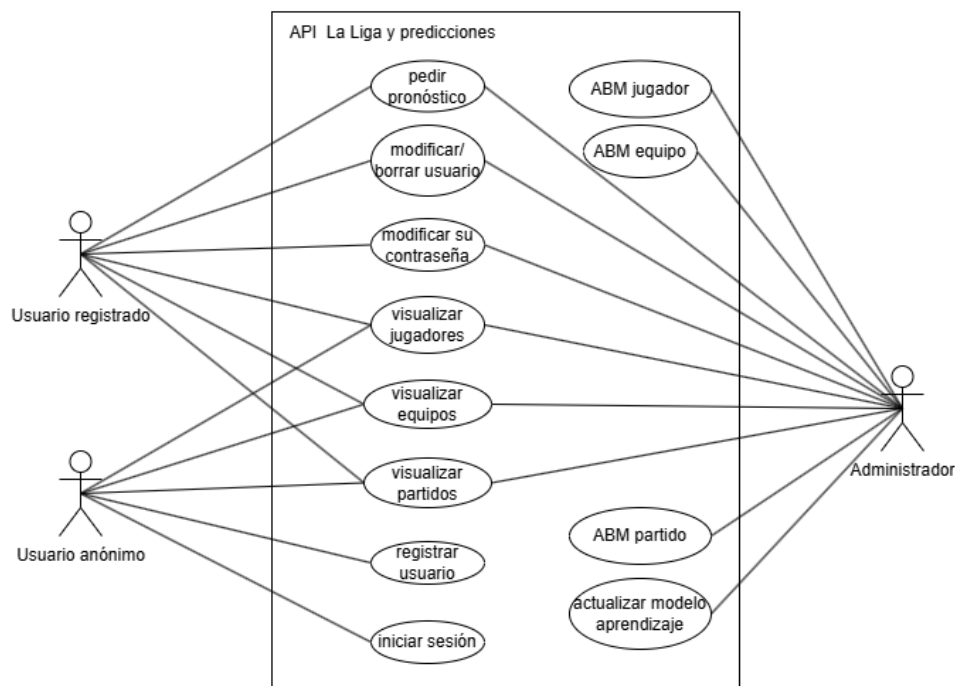


Figura 3 - Diagrama de Casos de uso de la API a realizar

La aplicación consta de 3 actores principales. Por un lado, se encuentra el usuario anónimo, el cual podrá visualizar todos los datos de los cuales dispone la API (partidos, equipos y jugadores) así como poder llevar a cabo las funcionalidades de registro de un nuevo usuario y el de iniciar sesión.

El usuario registrado, además de poder realizar las funcionalidades relacionadas con la visualización de datos que tiene a su disposición el usuario anónimo, puede acceder a la sección de la predicción de un encuentro, así como modificar la información de su cuenta o eliminarla.

El administrador será capaz de realizar todas las funciones que tiene a su disposición el usuario registrado además de realizar las operaciones ABM (Altas, Bajas y Modificaciones) de partidos, equipos o jugadores en la base de datos. También está autorizado a actualizar el modelo de aprendizaje que determina las predicciones.

Capítulo 4

Diseño

4.1 Arquitectura del sistema

El diseño de la arquitectura del sistema de pronósticos de partidos dentro de una API sobre *LaLiga* se enfoca en proporcionar una visión clara de cómo se estructura el sistema. Se desarrollarán varios diagramas para ilustrar la interacción entre las interfaces del sistema, los servicios proporcionados y los componentes subyacentes.

4.1.1 Arquitectura lógica

En relación con la arquitectura lógica del sistema, tal y como se indica en Figura 4, se dividirá en los siguientes componentes:

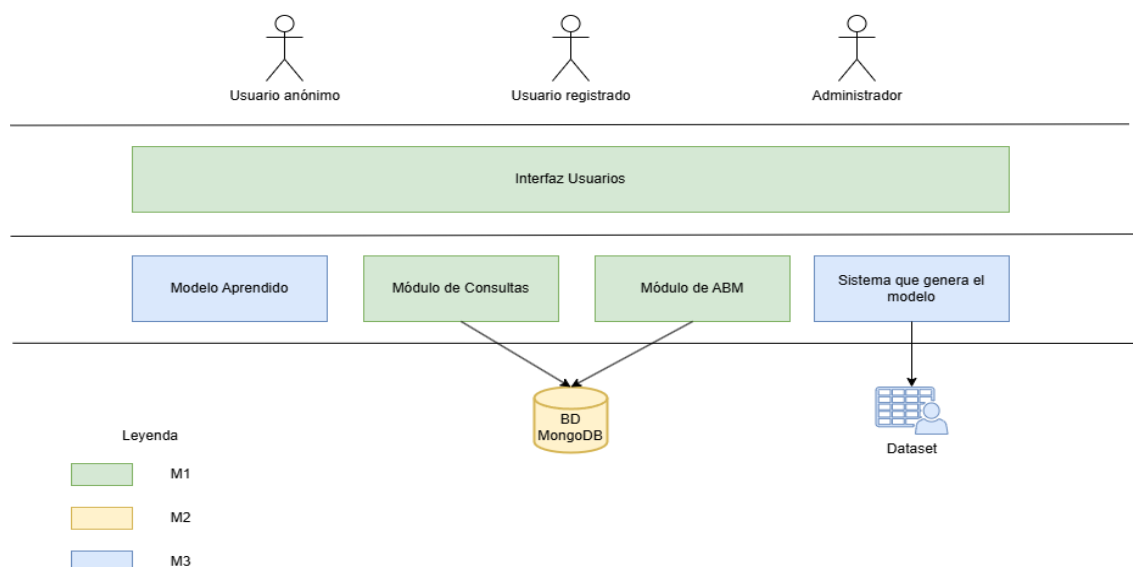


Figura 4 - Arquitectura lógica del sistema

Componentes

- **Interfaz de usuarios:** actúa como el único punto de entrada para todas las solicitudes por parte de los usuarios. Proporciona una capa de abstracción sobre los servicios internos y maneja las preocupaciones comunes como la autenticación correcta o el registro.
- **Módulo de consultas:** gestiona todos los aspectos relacionados con las consultas CRUD propias del servicio REST.
- **Módulo de ABM:** gestiona todos los aspectos relacionados con las consultas de altas, bajas o modificaciones realizadas por un usuario administrador.
- **Sistema que genera el modelo:** servicio destinado a entrenar el modelo y activar el servicio que realizará los pronósticos que se soliciten mediante el servicio web.
- **Modelo aprendido:** modelo entrenado que se encarga de determinar un pronóstico.

En cuanto a la leyenda que se observa en la Figura 4, se explicará en la arquitectura física.

4.1.2 Arquitectura física

La arquitectura física del sistema proporciona todos los componentes físicos que contribuyen al despliegue y funcionamiento del sistema. Tal y como se observa en la Figura 5, se trata de un despliegue en la nube.

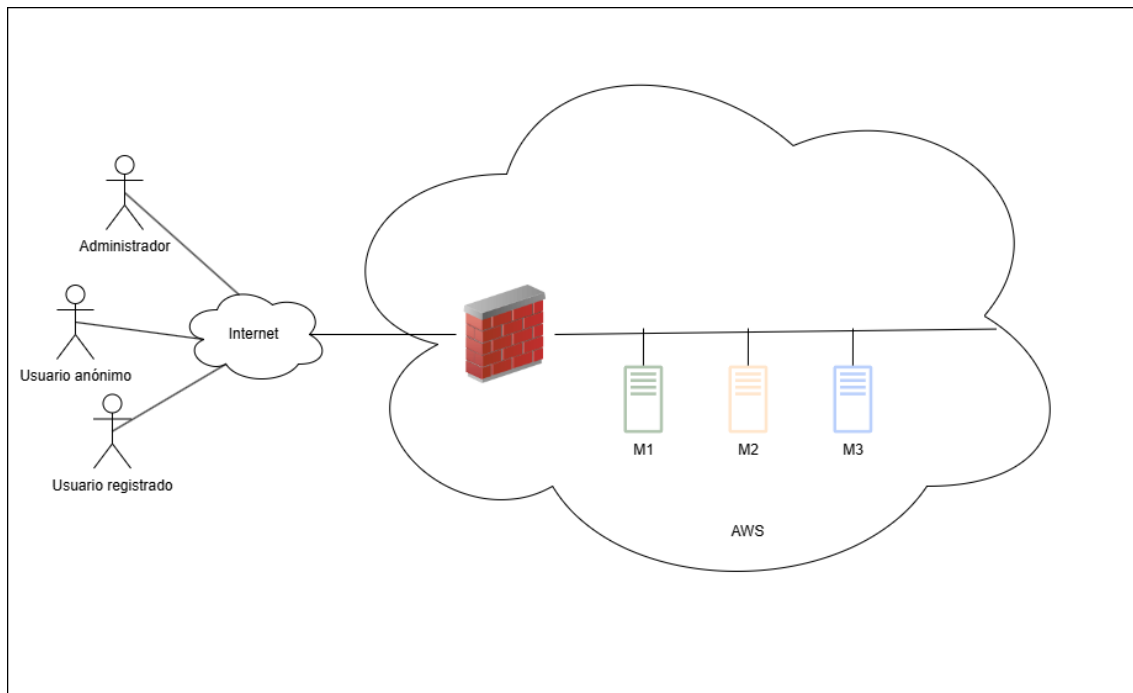


Figura 5 - Arquitectura física del sistema

- **AWS:** entorno de infraestructuras proporcionado por *Amazon* en el cual se encuentran una gran cantidad de recursos (instancias, grupos de seguridad, redes privadas virtuales, etc).
- **Cortafuegos:** a través de los grupos de seguridad que pone a su disposición AWS, para cada una de las máquinas se establece una serie de reglas de entrada y de salida.
- **Máquina 1 (M1):** servidor dedicado al servicio web, basado en una arquitectura REST. Es con la única máquina con la que el usuario interactúa de forma directa.
- **Máquina 2 (M2):** servidor dedicado a almacenar la base de datos MongoDB que recoge toda la información de la API. Se ha separado de la máquina 1 para asegurar la disponibilidad.
- **Máquina 3 (M3):** servidor dedicado al servicio de pronósticos. En esta máquina se cuenta con el *dataset* de los partidos que servirán para entrenar al modelo y se despliega el servicio que emitirá una respuesta ante las solicitudes que se envíen mediante el servicio REST.

4.2 Diseño del servicio REST

REST (*Representational State Transfer*) es un estilo de arquitectura para sistemas distribuidos, especialmente adecuado para aplicaciones web. Introducido por Roy Fielding en su disertación doctoral en el año 2000, REST se basa en un conjunto de principios y restricciones que, cuando se aplican correctamente, ayudan a crear sistemas eficientes, escalables y fáciles de mantener.

Arquitectura REST

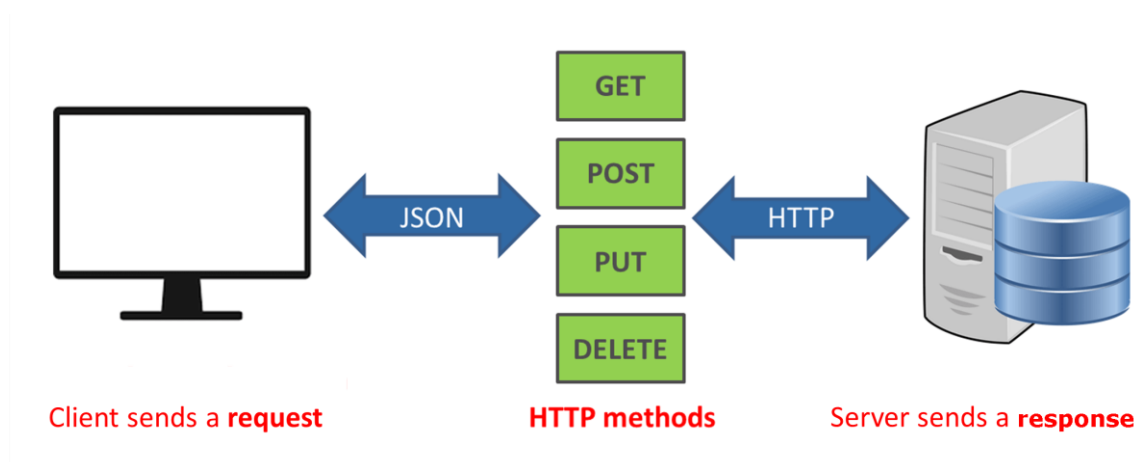


Figura 6 - Arquitectura REST (Fuente: <https://aprendiendoarduino.wordpress.com/2019/10/27/api-rest/>)

Uno de los principios fundamentales de REST es la separación de responsabilidades entre el cliente y el servidor. Esta separación permite que ambas partes evolucionen de manera independiente, mejorando la escalabilidad y la flexibilidad del sistema. Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud, lo que se conoce como ausencia de estado (*stateless*). Esto significa que el servidor no necesita almacenar el estado del cliente entre solicitudes, lo que mejora la escalabilidad y la robustez del sistema.

Además, está la noción de HATEOAS (*Hypermedia as the Engine of Application State*). Según este principio, los clientes interactúan con la aplicación a través de hipermedios proporcionados dinámicamente por las respuestas del servidor. Esto significa que cada

respuesta del servidor no solo contiene los datos solicitados, sino también enlaces a otros recursos relacionados y acciones disponibles, permitiendo una navegación intuitiva y autodocumentada a través de la API.

Diseño RESTful

Para que el servicio REST cumpla con los criterios RESTful y la noción de HATEOAS, se implementarán varias características clave. En primer lugar, los recursos se identificarán mediante URLs, utilizando métodos HTTP estándar (GET, POST, PUT, DELETE) para operar sobre ellos. Cada mensaje del cliente al servidor será autoexplicativo, conteniendo toda la información necesaria para entender y procesar la solicitud, junto con enlaces a otros recursos y acciones disponibles.

Los recursos se representarán en formato JSON, proporcionando una descripción clara y completa de cada recurso. Además, las respuestas del servidor incluirán hipervínculos a otros recursos y posibles acciones, permitiendo a los clientes navegar a través de la aplicación de manera intuitiva.

4.2.1 Funcionalidades del servicio REST

El servicio REST incluye las siguientes funcionalidades principales:

1. Autenticación y gestión de Usuarios

- Registro e inicio de sesión de usuarios.
- Recuperación de contraseñas.
- Modificación y eliminación del usuario.

2. Gestión de Datos de LaLiga

- CRUD para partidos, equipos y jugadores.

3. Predicciones de Partidos

- *Endpoint* para obtener predicciones de resultados.
- Método para actualizar el modelo reentrenándolo.

4.2.2 Descripción de los Endpoints

La API incluye varios endpoints, cada uno diseñado para realizar una tarea específica. A continuación, se describen los principales endpoints y sus funcionalidades:

1. Usuarios

- **POST /usuarios/registro:** registrar un nuevo usuario.
- **POST /usuarios/login:** iniciar sesión de usuario.
- **POST /usuarios/recuperar:** enviar correo de recuperación de contraseña.
- **POST /usuarios/reset/{token}:** restablecer la contraseña.
- **PUT /usuario/{id}:** modificar correo o nombre de usuario (solo disponible para el usuario en cuestión).
- **DELETE /usuario/{id}:** eliminar usuario (permitido para el usuario en cuestión).

2. Partidos

- **GET /partidos:** obtener todos los partidos.
- **POST /partidos:** añadir un nuevo partido (admin).
- **GET /partidos/{id}:** obtener un partido específico por ID.
- **PUT /partidos/{id}:** actualizar un partido (admin).
- **DELETE /partidos/{id}:** eliminar un partido (admin).
- **GET /partidos/{id}/estadisticas:** obtener estadísticas de un partido específico por ID.

3. Predicciones

- **POST /predicciones:** obtener predicción para un partido (user/admin).
- **POST/predicciones/actualizar:** reentrenar el modelo de aprendizaje (admin).

4. Equipos:

- **GET /equipos:** obtener todos los equipos.
- **POST /equipos:** añadir un nuevo equipo (admin).
- **GET /equipos/{id}:** obtener un equipo específico por ID.
- **PUT /equipos/{id}:** actualizar un equipo (admin).

- **DELETE /equipos/{id}**: eliminar un equipo (admin).
- **GET /equipos/{id}/jugadores**: obtener todos los jugadores de un equipo específico.

5. Jugadores:

- **GET /jugadores**: obtener todos los jugadores.
- **POST /jugadores**: añadir un nuevo jugador (admin).
- **GET /jugadores/{id}**: obtener un jugador específico por ID.
- **PUT /jugadores/{id}**: actualizar un jugador (admin).
- **DELETE /jugadores/{id}**: eliminar un jugador (admin).

Especificación OpenAPI

El archivo OpenAPI define los endpoints, métodos HTTP, parámetros y esquemas de datos utilizados en la API. Esto asegura una documentación clara y facilita la integración con otras aplicaciones. El archivo OpenAPI se encuentra ubicado en el directorio 'schema/' del proyecto bajo el nombre 'openapi_liga.yaml'. Se puede consultar el *Anexo I – OpenAPI* para más información.

4.3 Diseño de datos

4.3.1 Fuente de datos

El proyecto requiere acceso a una fuente de datos robusta, confiable y actualizada que proporcione información detallada sobre los partidos, equipos y jugadores de *LaLiga*. La elección de la fuente de datos es crítica, ya que impacta directamente en la calidad y precisión de las predicciones realizadas por el sistema.

Criterios de Evaluación

Para asegurar una selección justificada, se establecen varios criterios de evaluación:

- **Cobertura de Datos**: la completitud de los datos sobre partidos, equipos y jugadores.

- **Costos:** La viabilidad económica dentro del presupuesto del proyecto.
- **Soporte Técnico y Documentación:** La disponibilidad de guías y asistencia técnica para facilitar la integración y el uso continuo.
- **Fiabilidad:** La estabilidad y confiabilidad del servicio de datos.

Elección Final: API-Football

Tras un análisis exhaustivo, se seleccionó *API-Football* como la fuente de datos principal. Esta decisión se basó en:

- **Cobertura Exhaustiva:** *API-Football* proporciona una extensa variedad de datos sobre LaLiga, incluyendo información detallada de partidos, equipos y jugadores.
- **Accesibilidad Económica:** Ofrece un plan gratuito que satisface las necesidades del proyecto en su fase inicial, con opciones de escalado asequibles.
- **Documentación detallada y soporte técnico:** Dispone de documentación comprensiva y un buen soporte técnico, facilitando la integración y el manejo continuo.

La elección de *API-Football* como fuente de datos cumple con los criterios establecidos y proporciona la mejor combinación de características para el desarrollo exitoso del proyecto. Esta fuente no solo garantiza el acceso a datos de calidad y actualizados, sino que también se alinea con las restricciones presupuestarias y técnicas del proyecto.

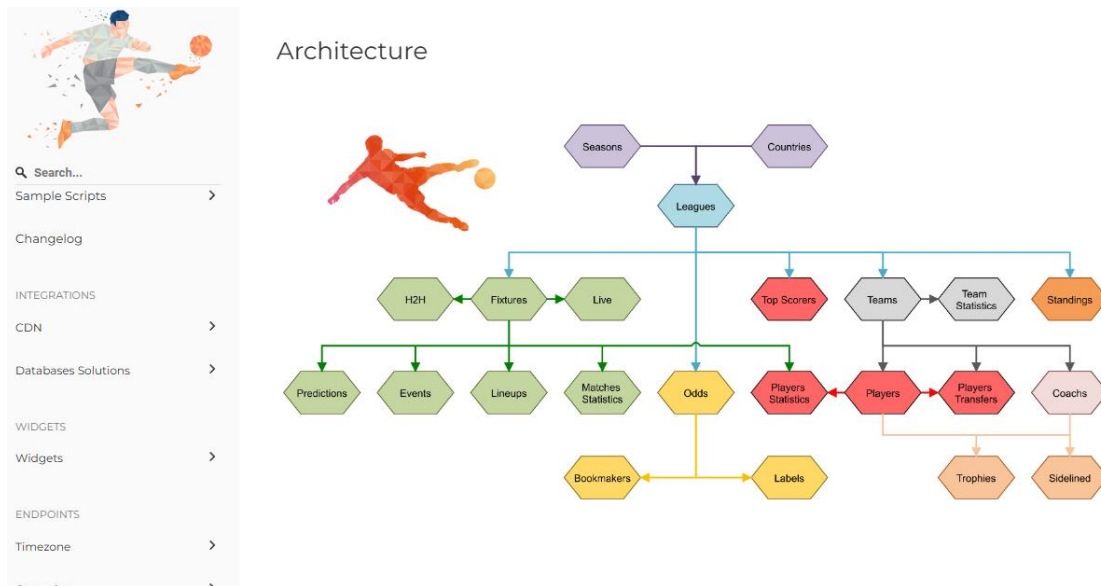


Figura 7 - Arquitectura API Football (Fuente: <https://www.api-football.com/documentation-v3#section/Architecture>)

4.3.2 Migración y carga inicial de datos

Para la migración carga inicial de los datos, se realizan una serie de scripts, donde se procede a almacenar en una base de datos MongoDB, los datos referentes a los equipos, jugadores y partidos de LaLiga en la temporada 2023-24, asegurando que se cumplen los límites que marca el plan gratuito (100 consultas al día a API-Football y 20 por minuto), por lo que en lo que se refiere a los partidos, podía cargar en torno a 100 partidos al día como máximo.

4.3.3 Modelo Entidad-Relación simplificado

Los datos almacenados en la base de datos MongoDB tendrán la estructura señalada en el modelo que se indica en la -. El modelo está de forma simplificada puesto que, a excepción de los usuarios, cada objeto tiene un gran número atributos.

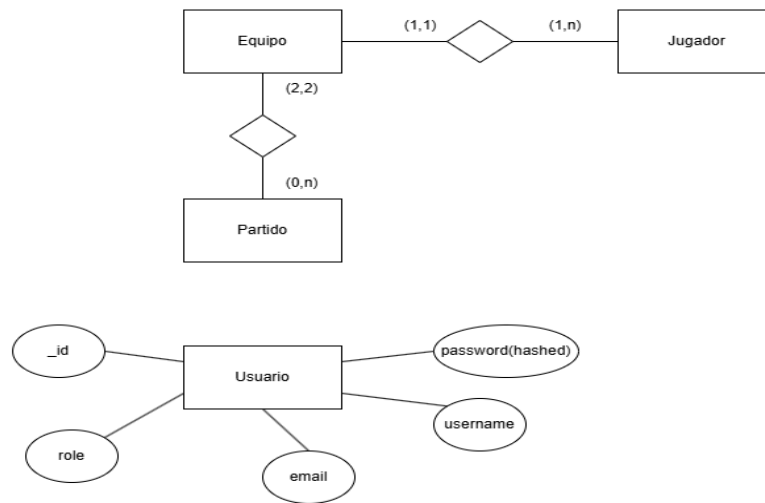


Figura 8 - Modelo Entidad-Relación de Base de Datos API (simplificado)

4.4 Diseño del servicio de pronósticos

Se desea llevar a cabo el servicio de pronósticos mediante un modelo de aprendizaje automático que determine si en un partido resulta vencedor el equipo local, el visitante o sucede un empate. Se consideró conveniente, tomando los datos de los partidos de la base de datos en MongoDB, se optó por exportarlos en formato CSV mediante un script y así contar con un *dataset* que facilita su implementación como datos de entrenamiento y ahorra incompatibilidades.

Además, se incrementó el número de partidos en el dataset, cargando más partidos desde *API-Football*, incluyendo los partidos de la temporada 2022-2023 de LaLiga, así como los de la temporada 2023-2024 de la *Bundesliga* (competición alemana), asegurando una mayor precisión del modelo.

4.4.1 Transformación de datos

Una vez se obtiene CSV de todos los partidos (*matches.csv*), se procede a su transformación para seleccionar aquellas estadísticas que enriquezcan el modelo y ayuden a obtener una mayor precisión. En primer lugar, se eliminan todos aquellos campos que no resulten relevantes (nombres de equipos, estadios, árbitros...).

Posteriormente, y una vez seleccionado el algoritmo, el cual se explicará más adelante, se eliminan estadísticas adicionales que provocan una confusión al algoritmo (posesión de balón, tarjetas, faltas...).

Finalmente, aquellas estadísticas de cada uno de los equipos que tendrá en cuenta el modelo son las siguientes:

- Disparos a portería
- Disparos fuera de portería
- Disparos totales
- Disparos bloqueados
- Disparos dentro del área
- Disparos fuera del área
- Saques de esquina
- Paradas del portero
- Pases totales
- Pases precisos
- Goles esperados

4.4.2 Selección de algoritmo de aprendizaje

Tras evaluar exhaustivamente varios modelos como *Random Forest*, *Gradient Boosting* o *Naive Bayes*, se obtuvo una mayor precisión con **Logistic Regression**. La regresión logística es efectiva debido a que permite realizar predicciones sobre eventos discretos, puesto que se basa en una clasificación multiclase; es un modelo relativamente simple, lo que significa que es fácil de entrenar para datos de menor complejidad; y cuenta con una interpretación intuitiva, ya que las salidas del modelo son probabilidades, algo muy útil para datos deportivos.

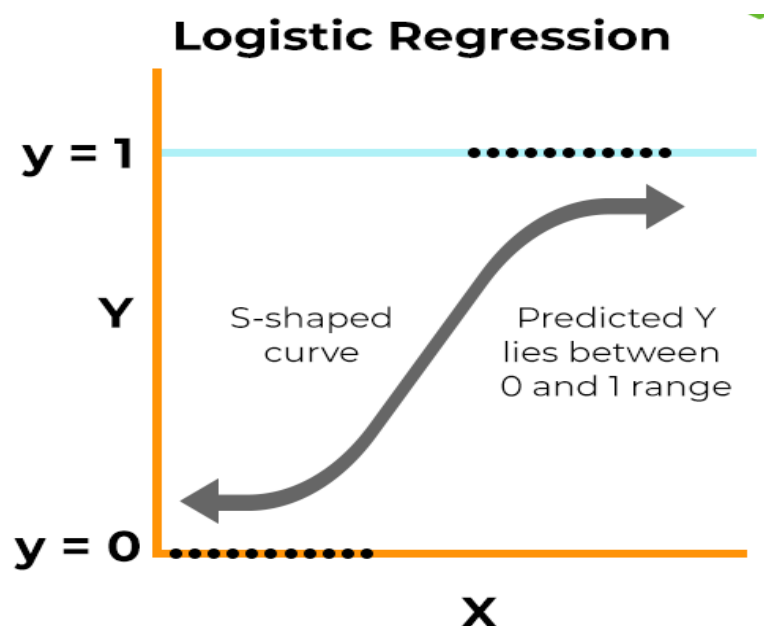


Figura 9 - Logistic Regression (Fuente: <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/>)

El algoritmo asigna como resultado 0 como victoria visitante, 1 como empate y 2 como victoria local. Al entrenar el modelo se obtienen los siguientes resultados de precisión:

Informe de clasificación Logistic Regression:				
	precision	recall	f1-score	support
0	0.84	0.91	0.87	57
1	0.73	0.71	0.72	58
2	0.92	0.89	0.90	99
accuracy			0.85	214
macro avg	0.83	0.84	0.83	214
weighted avg	0.85	0.85	0.85	214
Matriz de confusión Logistic Regression:				
[[52 4 1]				
[10 41 7]				
[0 11 88]]				
Precisión promedio Logistic Regression en validación cruzada: 0.8415686274509804				

Figura 10 - Resultados de Logistic Regression

En consecuencia, se puede apreciar que para la victoria visitante el modelo tiene una precisión del 84%, en el caso del empate tiene una precisión del 73%, lo cual es lógico que sea menor puesto que existe una mayor igualdad en las estadísticas de cada equipo; y una precisión del 92% en caso de victorias locales. La precisión media es del 84,51%

en validación cruzada (puesto que se dividen los datos del dataset entre entrenamiento y prueba).

Estos datos determinan que este modelo es adecuado para el proyecto que se está llevando a cabo, ya que los resultados son bastante notables.

Para ver otros algoritmos alternativos y sus resultados, se puede consultar el código `evaluate_models.py`.

Capítulo 5

Implementación y validación del sistema

5.1 Entorno de construcción

Para la implementación de este proyecto se han empleado diversas herramientas, tecnologías y frameworks.

IDE

- **Visual Studio Code (VS Code):** el editor de código para el desarrollo del trabajo.

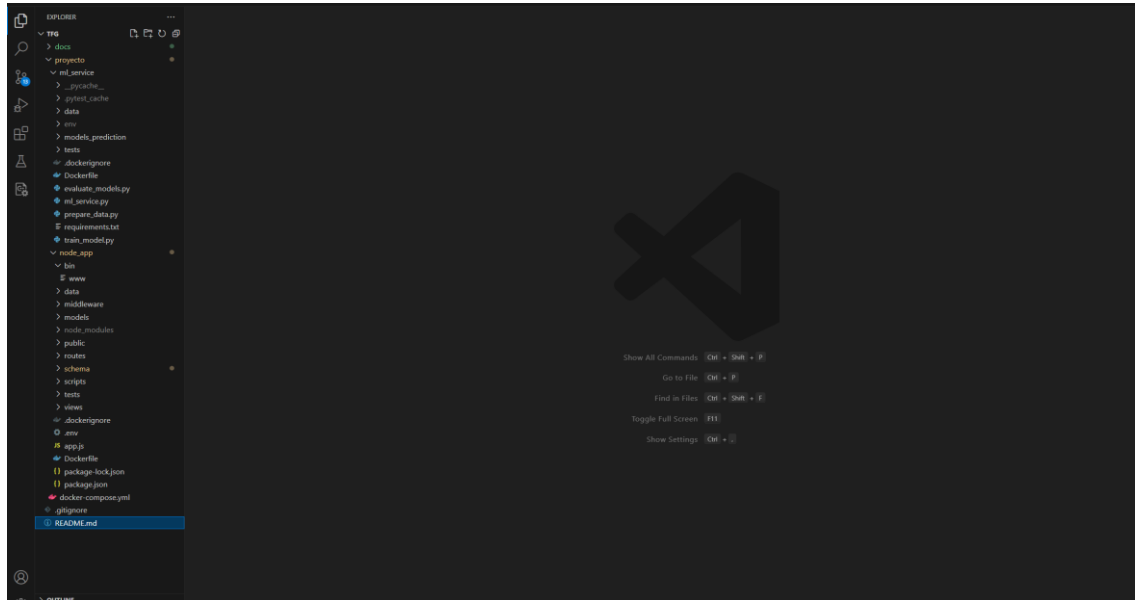


Figura 11 - IDE VS Code

Lenguajes de programación

- **Javascript (Node.js):** usado para la creación de la API RESTful, gestionando las rutas y operaciones CRUD sobre equipos, jugadores y partidos.
- **Python:** utilizado para desarrollar el modelo de predicción basado en aprendizaje automático, integrado con la API.

Frameworks

- **Express.js:** framework de Node.js empleado para crear la API REST, que facilita la gestión de rutas, middleware y manejo de solicitudes HTTP.
- **Flask:** utilizado para encapsular el modelo de aprendizaje automático en *Python* y exponerlo como un servicio que interactúa con la API REST.

Contenedores

- **Docker:** destinado a crear contenedores que encapsulan los servicios para el despliegue local.

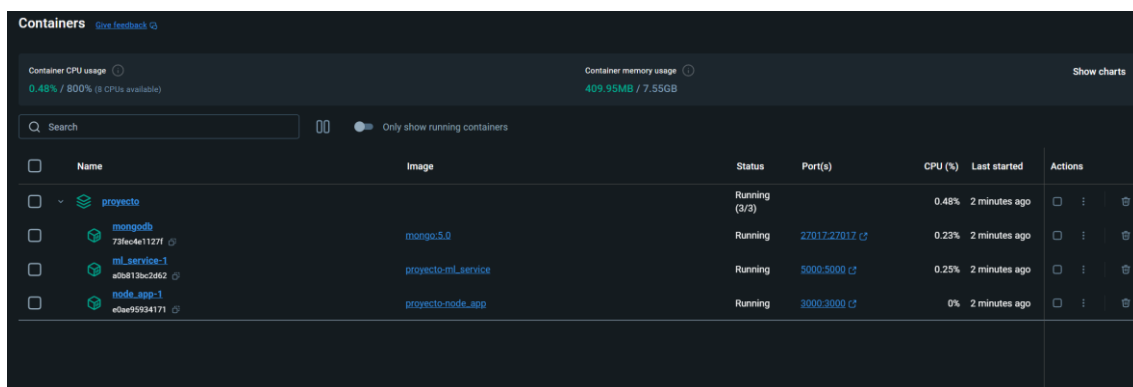


Figura 12 - Docker Desktop

Bases de datos

- **MongoDB:** base de datos NoSQL utilizada para almacenar información de la API. Se ha elegido por su capacidad de manejar datos no estructurados y por su encaje con los servicios REST.


```
footballDB> db.teams.findOne()
{
  _id: ObjectId('6658e7c92b380aa0fb464dd5'),
  teamId: 529,
  name: 'Barcelona',
  code: 'BAR',
  country: 'Spain',
  founded: 1899,
  national: false,
  logo: 'https://media.api-sports.io/football/teams/529.png',
  venue: {
    id: 19939,
    name: 'Estadi Olímpic Lluís Companys',
    address: 'Carrer de l&apos;Estadi',
    city: 'Barcelona',
    capacity: 55926,
    surface: 'grass',
    image: 'https://media.api-sports.io/football/venues/19939.png'
  }
}
footballDB> |
```

Figura 13 - Consulta MongoDB

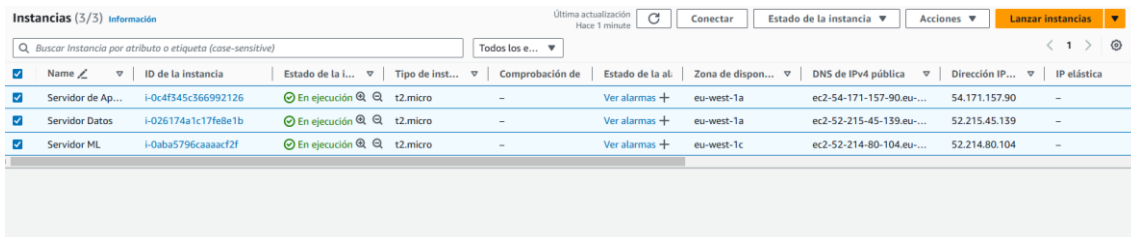
Librerías y módulos (simplificado)

- **Mongoose:** utilizado como para facilitar la manipulación de datos almacenados en MongoDB desde la API.
- **Bcrypt:** librería usada para el hashing de contraseñas de los usuarios de la API.
- **jsonwebtoken (JWT):** empleada para la autenticación basada en tokens, gestionando la protección de rutas y acceso a recursos.
- **Axios:** librería usada para realizar solicitudes desde la API a otros servicios, como el de pronósticos alojado en Flask.
- **Scikit-learn:** usado en Python para el desarrollo del modelo de aprendizaje automático, proporcionando herramientas para el entrenamiento y validación de modelos.

Despliegue en la nube

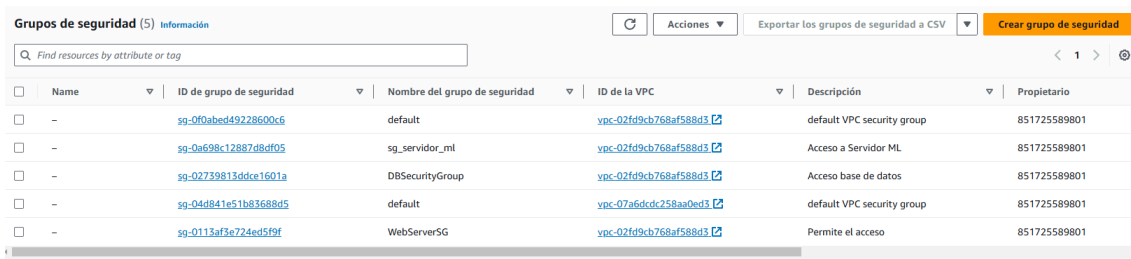
- **AWS (Amazon Web Services):** la infraestructura en la nube para alojar los servicios, asegurando escalabilidad y acceso en remoto. Se han empleado los

servicios relacionados con VPC y EC2 en la región *eu-west* (Irlanda) para el despliegue de instancias en el mismo entorno y sus consecuentes reglas de seguridad.



Seleccionar	Name	ID de la instancia	Estado de la i...	Tipo de inst...	Comprobación de	Estado de la al...	Zona de dispon...	DNS de IPv4 pública	Dirección IP...	IP elástica
<input checked="" type="checkbox"/>	Servidor de Ap...	i-0c4f345c366992126	En ejecución	t2.micro	-	Ver alarmas +	eu-west-1a	ec2-54-171-157-90.eu-...	54.171.157.90	-
<input checked="" type="checkbox"/>	Servidor Datos	i-026174a1c17f8e1b	En ejecución	t2.micro	-	Ver alarmas +	eu-west-1a	ec2-52-215-45-139.eu-...	52.215.45.139	-
<input checked="" type="checkbox"/>	Servidor ML	i-0ba5796caaac2f	En ejecución	t2.micro	-	Ver alarmas +	eu-west-1c	ec2-52-214-80-104.eu-...	52.214.80.104	-

Figura 14 - Instancias AWS



Seleccionar	Name	ID de grupo de seguridad	Nombre del grupo de seguridad	ID de la VPC	Descripción	Propietario
<input type="checkbox"/>	-	sg-0f0abed49228600c6	default	vpc-02fd9cb768af588d3	default VPC security group	851725589801
<input type="checkbox"/>	-	sg-0a698c12887d8df05	sg_servidor_ml	vpc-02fd9cb768af588d3	Acceso a Servidor ML	851725589801
<input type="checkbox"/>	-	sg-02739813ddce1601a	DBSecurityGroup	vpc-02fd9cb768af588d3	Acceso base de datos	851725589801
<input type="checkbox"/>	-	sg-04d841e51b83688d5	default	vpc-07a6dcdc258aa0ed3	default VPC security group	851725589801
<input type="checkbox"/>	-	sg-0113af3e724ed5f9f	WebServerSG	vpc-02fd9cb768af588d3	Permite el acceso	851725589801

Figura 15 - Grupos de seguridad AWS

Gestores de paquetes

- **npm:** utilizado en el entorno de Node.js para gestionar las dependencias del proyecto.
- **pip:** empleado en el entorno de Python para instalar y gestionar librerías necesarias para el modelo de predicción.

5.2 Referencia al repositorio de software

Repositorio de software: <https://github.com/juanpablozs/TFG>

Este proyecto está preparado para desplegarse en *cloud*, pero ante la imposibilidad de asignar una dirección IP pública al servicio de forma gratuita durante un largo tiempo, se ha incluido el despliegue a través del gestor de contenedores, levantando los 3

servicios indicados en la arquitectura. En la defensa del proyecto se procederá a mostrar su ejecución en AWS.

5.3 Plan de pruebas

En ambos servicios, tanto en el servidor web, como en el de *machine learning*, se han incluido una serie de pruebas unitarias, incluyendo, además algunas pruebas de integración del modelo con el servicio REST.

En cuanto a los *tests* que se encuentran en el servicio de pronósticos, contamos con las siguientes comprobaciones:

1. Prueba para datos faltantes en la predicción
2. Prueba de pronóstico para una victoria local
3. Prueba de pronóstico para una victoria visitante
4. Prueba de pronóstico para un empate

El resultado de dichos tests son los siguientes:

```
(env) ubuntu@ip-172-31-38-178:~/TFG/proyecto/ml_service$ pytest tests/test_ml_service.py
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/ubuntu/TFG/proyecto/ml_service
collected 4 items

tests/test_ml_service.py .... [100%]

===== 4 passed in 0.92s =====
(env) ubuntu@ip-172-31-38-178:~/TFG/proyecto/ml_service$
```

Figura 16 - Tests unitarios de modelo de aprendizaje correctos

En relación con los tests unitarios y de integración del modelo existentes en el servicio web, empleando el framework *jest* se prueban las siguientes funcionalidades:

1. Obtener lista de equipos y añadir un equipo (si fue añadido previamente lo da por válido) (GET equipos/ y POST equipos/)
2. Obtener lista de jugadores y añadir un jugador (si es un jugador ya existente lo da por válido (GET jugadores/ y POST jugadores/)

3. Obtener una predicción válida para datos de entrada correctos. (POST/predicciones)

El resultado de dichas pruebas (con el servidor de *machine learning* arrancado) es el siguiente:

```
ubuntu@ip-172-31-1-213:~/TFG/proyecto/node_app$ npm test
> proyecto@0.0.0 test
> jest

(node:1563) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use 'node --trace-warnings ...' to show where the warning was created)
(node:1563) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
console.log
  Conectado a la base de datos
    at NativeConnection.log (app.js:40:31)

GET /equipos 200 70.951 ms - 8096
POST /equipos 400 24.872 ms - 45
GET /jugadores 200 10.234 ms - 6987
POST /jugadores 400 7.538 ms - 46
POST /predicciones 200 21.287 ms - 31
1/53 tests/test_routes.test.js
  Rutas de equipos
    ✓ Debería devolver una lista de equipos en GET /equipos (282 ms)
    ✓ Debería crear un nuevo equipo en POST /equipos (37 ms)
  Rutas de jugadores
    ✓ Debería devolver una lista de jugadores en GET /jugadores (21 ms)
    ✓ Debería crear un nuevo jugador en POST /jugadores (13 ms)
  Predicciones
    ✓ Debería devolver una predicción válida para datos de entrada correctos en POST /predicciones (28 ms)

Test Suites: 1 passed, 1 total
Tests: 5 passed, 5 total
Snapshots: 0 total
Time: 2.562 s, estimated 3 s
Ran all test suites.
```

Figura 17 - Tests rutas servicio REST e integración con modelo

Para validar los requisitos de rendimiento del sistema, se usa la herramienta *Apache JMeter* para comprobar los tiempos de respuesta del servicio web y las solicitudes que es capaz de manejar.

Se procede a hacer una consulta GET equipos/ a la IP de la máquina AWS desplegada:

Grupo de Hilos

Nombre: Grupo de Hilos

Comentarios

Acción a tomar después de un error de Muestreador

☒ Continuar ☐ Comenzar siguiente iteración ☐ Parar Hilo ☐ Parar Test ☐ Parar test ahora

Propiedades de Hilo

Número de Hilos: 40

Periodo de Subida (en segundos): 60

Contador del bucle: ☐ Sin fin 10

☒ Same user on each Iteration

☐ Retrasar la creación de Hilos hasta que se necesiten

☐ Planificador

Duración (segundos)

Retardo de arranque (segundos)

Figura 18 - Parámetros del grupo de hilos en JMeter

Figura 19 – Consultas a realizar

Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
Petición HTTP	400	5	3	11	1,01	0,00%	6,8/sec	55,29	0,79	8289,0
Total	400	5	3	11	1,01	0,00%	6,8/sec	55,29	0,79	8289,0

Figura 20 - Resultados de JMeter

Con estos resultados se puede afirmar que se cumplen los requisitos de rendimiento establecidos, puesto que en un minuto se han producido 400 consultas y el tiempo de respuesta máximo ha sido de 11ms.

5.4 Evaluación del sistema

En primer lugar, se proceden a levantar las instancias de AWS, visibles en la *Figura 14*. Posteriormente, se accede por SSH a las máquinas del servidor web y de *machine learning*, accesibles mediante clave pública y solo desde la red IP del administrador.

En la máquina de aprendizaje automático, hay un archivo destinado a transformar datos, por si se quiere avanzar en la profundidad del modelo, y dividir los datos de entrenamiento, además de normalizarlos, y prueba.

Cuando se tengan los datos en formato CSV divididos entre entrenamiento y de prueba (*train/test*), se procede a entrenar el modelo:

```
(env) ubuntu@ip-172-31-38-178:~/TFG/proyecto/ml_service$ python3 train_model.py
Informe de clasificación Logistic Regression:
      precision    recall  f1-score   support

   0       0.84       0.91       0.87        57
   1       0.73       0.71       0.72        58
   2       0.92       0.89       0.90       99

 accuracy         0.85
 macro avg       0.83
 weighted avg    0.85

Matriz de confusión Logistic Regression:
[[52  4  1]
 [10 41  7]
 [ 0 11 88]]
Precisión promedio Logistic Regression en validación cruzada: 0.8415686274509804

(env) ubuntu@ip-172-31-38-178:~/TFG/proyecto/ml_service$ |
```

Figura 21 - Entrenamiento del modelo desde la máquina ML

Una vez se tenga un modelo entrenado, se levanta el servicio:

```
(env) ubuntu@ip-172-31-38-178:~/TFG/proyecto/ml_service$ python3 ml_service.py
Modelo y scaler cargados correctamente.
* Serving Flask app 'ml_service'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.31.38.178:5000
Press CTRL+C to quit
* Restarting with stat
Modelo y scaler cargados correctamente.
* Debugger is active!
* Debugger PIN: 101-327-602
```

Figura 22 - Servicio ML desplegado

En la máquina del servicio web, se procede a instalar las dependencias en el directorio correspondiente con: *npm install*, y una vez se instalan los paquetes se puede iniciar el servicio que se conectará con la base de datos de la máquina del servidor de datos:

```
ubuntu@ip-172-31-1-213:~/TFG/proyecto/node_app$ npm start
> proyecto@0.0.0 start
> node ./bin/www

(node:1686) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use 'node --trace-warnings ...' to show where the warning was created)
(node:1686) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Conectado a la base de datos
```

Figura 23 - Servicio web levantado

Una vez esté todo desplegado, se pasa a las comprobaciones. Para ello, se usará la herramienta *Postman*, y así contar con una herramienta más cómoda que la consola del navegador.

En la ruta se introduce la dirección IP de la máquina que aloja el servicio web y el puerto 3000.

Haciendo un login al usuario administrador, se proporciona el token que se introducirá como valor en *Authorization* para acceder a rutas protegidas:

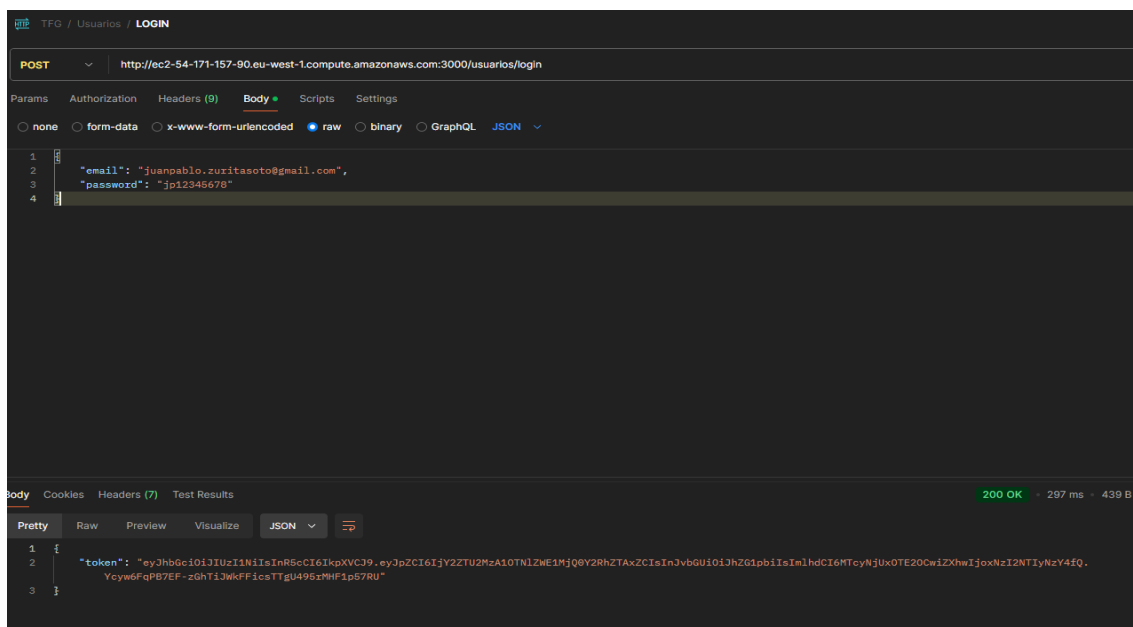


Figura 24 - Login de administrador

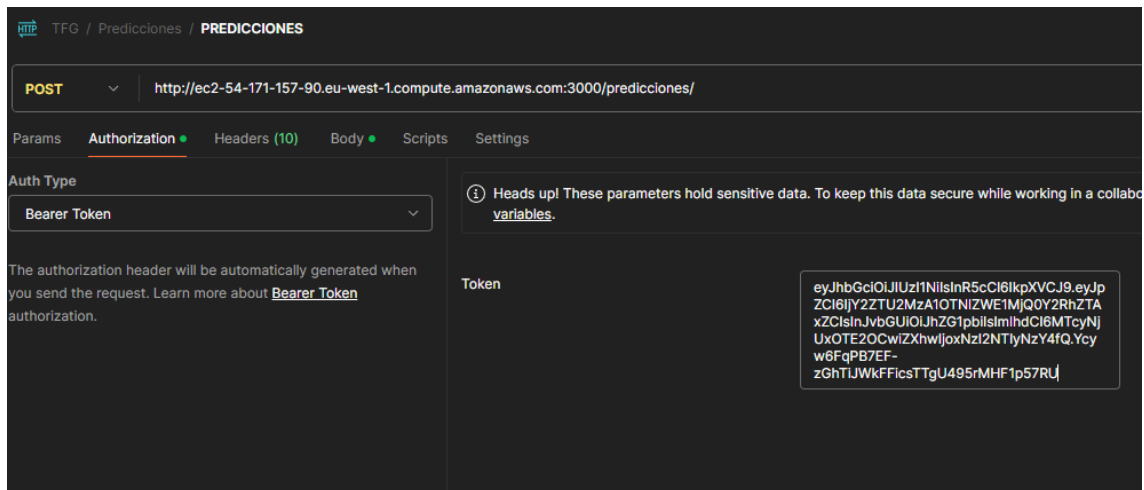


Figura 25 - Token de usuario en authorization para tener acceso a una ruta protegida

A la hora de comprobar el servicio de pronósticos, se introduce por cuerpo del mensaje en formato JSON las estadísticas requeridas para que el modelo pueda predecir un resultado. En el caso de poner estadísticas propias de un partido igualado, el modelo de forma mayoritaria tenderá al empate:

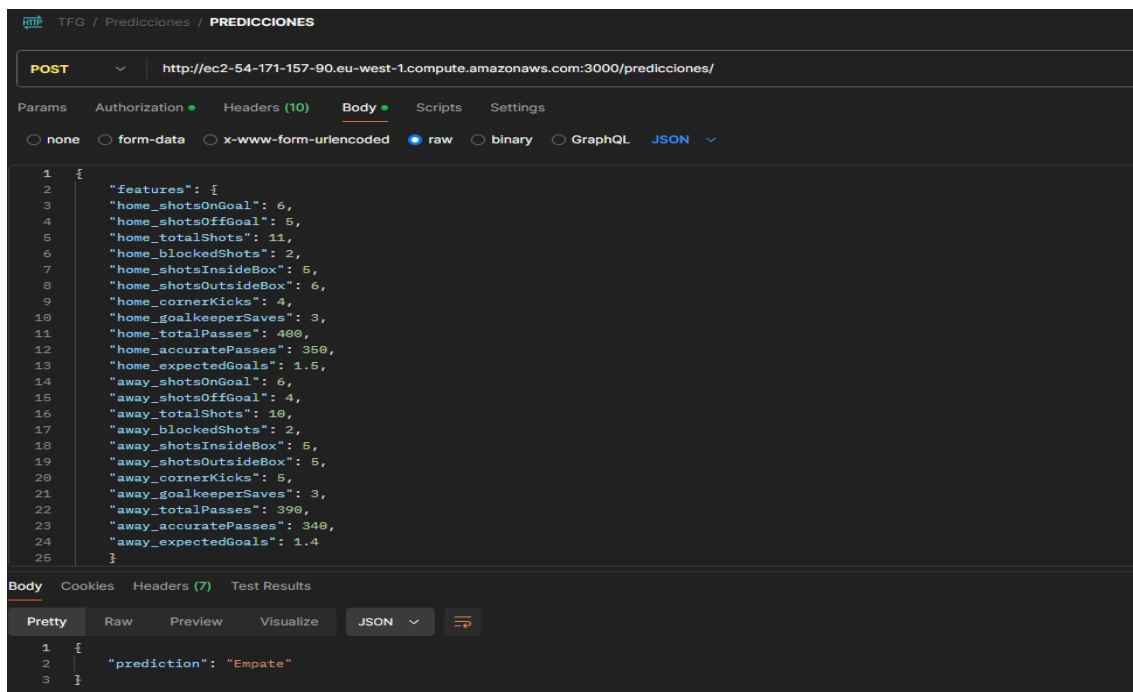


Figura 26 - Pronóstico empate

Si las estadísticas son favorables al equipo local, el modelo predice que ganará el equipo que juegue en su estadio:

```

POST http://ec2-54-171-157-90.eu-west-1.compute.amazonaws.com:3000/predicciones/

{
  "features": {
    "home_shotsOnGoal": 10,
    "home_shotsOffGoal": 8,
    "home_totalShots": 18,
    "home_blockedShots": 4,
    "home_shotsInsideBox": 12,
    "home_shotsOutsideBox": 6,
    "home_cornerKicks": 7,
    "home_goalkeeperSaves": 5,
    "home_totalPasses": 600,
    "home_accuratePasses": 450,
    "home_expectedGoals": 2.5,
    "away_shotsOnGoal": 2,
    "away_shotsOffGoal": 3,
    "away_totalShots": 5,
    "away_blockedShots": 1,
    "away_shotsInsideBox": 3,
    "away_shotsOutsideBox": 2,
    "away_cornerKicks": 2,
    "away_goalkeeperSaves": 4,
    "away_totalPasses": 300,
    "away_accuratePasses": 250,
    "away_expectedGoals": 0.8
  }
}

{
  "prediction": "Victoria Local"
}
  
```

Figura 27 - Pronóstico victoria local

Sucede el mismo caso para una victoria del equipo visitante:

```

POST http://ec2-54-171-157-90.eu-west-1.compute.amazonaws.com:3000/predicciones/

{
  "features": {
    "home_shotsOnGoal": 3,
    "home_shotsOffGoal": 5,
    "home_totalShots": 8,
    "home_blockedShots": 1,
    "home_shotsInsideBox": 5,
    "home_shotsOutsideBox": 3,
    "home_cornerKicks": 3,
    "home_goalkeeperSaves": 6,
    "home_totalPasses": 400,
    "home_accuratePasses": 350,
    "home_expectedGoals": 0.9,
    "away_shotsOnGoal": 9,
    "away_shotsOffGoal": 7,
    "away_totalShots": 16,
    "away_blockedShots": 3,
    "away_shotsInsideBox": 10,
    "away_shotsOutsideBox": 6,
    "away_cornerKicks": 8,
    "away_goalkeeperSaves": 2,
    "away_totalPasses": 520,
    "away_accuratePasses": 470,
    "away_expectedGoals": 3.1
  }
}

{
  "prediction": "Victoria Visitante"
}
  
```

Figura 28 - Pronóstico victoria visitante

Además de estas rutas, se pueden probar el resto de las funcionalidades disponibles en la API, dependiendo del tipo de usuario que se autentique. La API cuenta con gestión de errores, proporcionando los mensajes adecuados.

Capítulo 6

Conclusiones y líneas futuras

6.1 Conclusiones

Este proyecto ha demostrado la efectividad de integrar modelos de inteligencia artificial dentro de una API RESTful para proporcionar predicciones automáticas en el ámbito deportivo. A través de la construcción de un modelo de aprendizaje automático basado en regresión logística, ha sido posible predecir resultados de partidos de fútbol con un enfoque preciso y eficiente. El uso de técnicas de machine learning ha permitido analizar una serie de características de los partidos, optimizando el proceso de pronóstico. Además, el hecho de encapsular el modelo en un servicio REST ha facilitado el acceso a las predicciones desde diferentes puntos, lo que ofrece una gran flexibilidad en el uso del sistema.

Una de las principales conclusiones del proyecto es la confirmación de que la combinación de machine learning con servicios REST abre nuevas posibilidades en la automatización de procesos y la toma de decisiones basada en datos. Este enfoque no solo es aplicable al deporte, sino que tiene el potencial de ser utilizado en muchos otros campos donde las predicciones basadas en datos históricos juegan un papel crucial.

Otra conclusión relevante ha sido el interés de observar cómo el despliegue en la nube y el uso de Docker han proporcionado diferentes formas eficientes de gestionar el proyecto, asegurando flexibilidad y estabilidad en distintos entornos.

6.2 Líneas futuras

En el futuro, una de las líneas de trabajo más interesantes sería el desarrollo de un *frontend* que convierta este sistema en una aplicación web completa, permitiendo a los usuarios interactuar de forma más intuitiva con las funcionalidades de la API. Además, sería beneficioso explorar la integración de características avanzadas en la infraestructura de AWS, como el uso de VPN para el administrador o la implementación de balanceo de carga para mejorar la escalabilidad y el rendimiento en entornos con mayor tráfico.

También se podría evaluar la viabilidad de transformar el proyecto en una plataforma comercial, similar a las quinielas deportivas, proporcionando predicciones basadas en inteligencia artificial. La implementación de HTTPS mejoraría la seguridad en las comunicaciones, protegiendo tanto a los usuarios como a los datos.

Otra línea por considerar sería la ampliación de las predicciones a otras ligas y deportes, aumentando el alcance del sistema. A largo plazo, optimizar el procesamiento en tiempo real y mejorar los aspectos de seguridad permitirían una evolución sólida del proyecto, haciéndolo más competitivo y fiable en el mercado.

Bibliografía

- [1] Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Media.
- [2] MongoDB, Inc. (n.d.). MongoDB: The application data platform. <https://www.mongodb.com/>
- [3] Node.js Foundation. (n.d.). Node.js. <https://nodejs.org/>
- [4] Express.js. (n.d.). Express - Node.js web application framework. <https://expressjs.com/>
- [5] OpenAI. (2024). ChatGPT (version 3.5). <https://chatgpt.com/>
- [6] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction.
- [7] API Football. (n.d.). API Football. <https://www.api-football.com/documentation-v3>
- [8] Sánchez Picot, Álvaro. (2024). REST, Sistemas Web II. Universidad CEU San Pablo.
- [9] García García, Raúl. (2023). Metodologías Clásicas de Construcción de Software, Ingeniería del Software. Universidad CEU San Pablo.
- [10] Amazon Web Services (AWS). (n.d.). AWS Cloud Products. <https://aws.amazon.com/products/>
- [11] Logistic Regression. (n.d.). Logistic Regression Algorithm - Towards Data Science. <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>

Anexo I – OpenAPI

openapi: 3.0.0

info:

title: API de LaLiga y predicción de partidos.

description: API para gestionar datos de LaLiga y predecir resultados en base a estadísticas.

version: 1.0.0

servers:

- url: http://localhost:3000

description: Servidor local

components:

schemas:

Usuario:

type: object

properties:

id:

type: string

username:

type: string

email:

type: string

password:

type: string

role:

type: string

links:

type: array

items:

\$ref: '#/components/schemas/Link'

Partido:

type: object

required:

- matchId

- teams

- goals

properties:

id:

type: string

date:

type: string

format: date-time

teams:

type: object

required:

- home

- away

properties:

home:

type: object

required:

- id

- name

- winner

properties:

id:

type: integer

name:

type: string

logo:

type: string

winner:

type: boolean

away:

type: object

required:

- id

- name

- winner

properties:

id:

type: integer

name:

type: string

logo:

type: string

winner:

type: boolean

goals:

type: object

required:

- home

- away

properties:

home:

type: integer

away:

type: integer

statistics:

type: array

items:

type: object

properties:

team:

type: object

properties:

id:

type: integer

name:

type: string

logo:

type: string

stats:

type: object

properties:

shotsOnGoal:

type: integer

shotsOffGoal:

type: integer

totalShots:

type: integer

blockedShots:

type: integer

shotsInsideBox:

type: integer

shotsOutsideBox:

type: integer

fouls:

type: integer

cornerKicks:

type: integer

offsides:

type: integer

possession:

type: string

yellowCards:

type: integer

redCards:

type: integer

goalkeeperSaves:

type: integer

totalPasses:

type: integer

accuratePasses:

type: integer

passPercentage:

type: string

expectedGoals:

type: string

links:

type: array

items:

\$ref: '#/components/schemas/Link'

Equipo:

type: object

required:

- teamId

- name

- venue

properties:

id:

type: string

teamId:

type: integer

name:

type: string

code:

type: string

country:

type: string

founded:

type: integer

logo:

type: string

venue:

type: object

required:

- name

properties:

id:

type: integer

name:

type: string

address:

type: string

city:

type: string

capacity:

type: integer

image:

type: string

jugadores:

type: array

items:

type: object

properties:

playerId:

type: string

name:

type: string

position:

type: string

links:

type: array

items:

\$ref: '#/components/schemas/Link'

Jugador:

type: object

required:

- playerId

- name

- position

- age

- teamId

properties:

id:

type: string

playerId:

type: integer

name:

type: string

firstname:

type: string

lastname:

type: string

age:

type: integer

birth:

type: object

properties:

date:

type: string

place:

type: string

country:

type: string

nationality:

type: string

height:

type: string

weight:

type: string

photo:

type: string

equipold:

type: string

equipoNombre:

type: string

position:

type: string

links:

type: array

items:

\$ref: '#/components/schemas/Link'

Link:

type: object

properties:

rel:

type: string

href:

type: string

method:

type: string

paths:

/usuarios/registro:

post:

summary: Registrar un nuevo usuario

requestBody:

content:

application/json:

schema:

\$ref: '#/components/schemas/Usuario'

responses:

'201':

description: Usuario registrado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Usuario'

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Datos de usuario no válidos"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al registrar el usuario"

/usuarios/login:

post:

summary: Iniciar sesión de usuario

requestBody:

content:

application/json:

schema:

type: object

properties:

email:

type: string

password:

type: string

responses:

'200':

description: Usuario autenticado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Usuario'

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Credenciales no válidas"

'401':

description: No autorizado

content:

application/json:

example:

message: "Correo electrónico o contraseña incorrectos"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al iniciar sesión del usuario"

/usuarios/recuperar:

post:

summary: Enviar correo de recuperación de contraseña

requestBody:

content:

application/json:

schema:

type: object

properties:

email:

type: string

responses:

'200':

description: Correo de recuperación enviado

content:

application/json:

example:

message: "Correo de recuperación enviado"

'404':

description: Usuario no encontrado

content:

application/json:

example:

message: "Usuario no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al enviar el correo de recuperación"

/usuarios/reset/{token}:

post:

summary: Restablecer la contraseña con un token de recuperación

parameters:

- name: token

in: path

required: true

schema:

type: string

requestBody:

content:

application/json:

schema:

type: object

properties:

password:

type: string

responses:

'200':

description: Contraseña actualizada exitosamente

content:

application/json:

example:

message: "Contraseña actualizada correctamente"

'400':

description: Token inválido o expirado

content:

application/json:

example:

message: "Token inválido o expirado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al restablecer la contraseña"

/usuarios/{id}:

get:

summary: Obtener los datos de un usuario por ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: Datos del usuario obtenidos exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Usuario'

'403':

description: No tienes permiso para ver este usuario

content:

application/json:

example:

message: "No tienes permiso para ver este usuario"

'404':

description: Usuario no encontrado

content:

application/json:

example:

message: "Usuario no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener el usuario"

put:

summary: Actualizar los datos de un usuario por ID

parameters:

- name: id

in: path

required: true

schema:

type: string

requestBody:

content:

application/json:

schema:

\$ref: '#/components/schemas/Usuario'

responses:

'200':

description: Usuario actualizado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Usuario'

'403':

description: No tienes permiso para actualizar este usuario

content:

application/json:

example:

message: "No tienes permiso para actualizar este usuario"

'404':

description: Usuario no encontrado

content:

application/json:

example:

message: "Usuario no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al actualizar el usuario"

delete:

summary: Eliminar un usuario por ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: Usuario eliminado exitosamente

content:

application/json:

example:

message: "Usuario eliminado correctamente"

'403':

description: No tienes permiso para eliminar este usuario

content:

application/json:

example:

message: "No tienes permiso para eliminar este usuario"

'404':

description: Usuario no encontrado

content:

application/json:

example:

message: "Usuario no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al eliminar el usuario"

/partidos:

get:

summary: Obtener todos los partidos

responses:

'200':

description: Lista de partidos

content:

application/json:

schema:

type: array

items:

\$ref: '#/components/schemas/Partido'

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener los partidos"

post:

summary: Añadir un nuevo partido

requestBody:

content:

application/json:

schema:

\$ref: '#/components/schemas/Partido'

responses:

'201':

description: Partido creado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Partido'

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Datos del partido no válidos"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al crear el partido"

/partidos/{id}:

get:

summary: Obtener un partido específico por ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: Detalles del partido

content:

application/json:

schema:

\$ref: '#/components/schemas/Partido'

'404':

description: Partido no encontrado

content:

application/json:

example:

message: "Partido no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener el partido"

put:

summary: Actualizar un partido

parameters:

- name: id

in: path

required: true

schema:

type: string

requestBody:

content:

application/json:

schema:

\$ref: '#/components/schemas/Partido'

responses:

'200':

description: Partido actualizado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Partido'

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Datos del partido no válidos"

'404':

description: Partido no encontrado

content:

application/json:

example:

message: "Partido no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al actualizar el partido"

delete:

summary: Eliminar un partido

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'204':

description: Partido eliminado exitosamente

'404':

description: Partido no encontrado

content:

application/json:

example:

message: "Partido no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al eliminar el partido"

/partidos/{id}/estadisticas:

get:

summary: Obtener estadísticas de un partido específico por ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: Estadísticas del partido

content:

application/json:

schema:

type: object

properties:

id:

type: string

estadisticas:

type: object

'404':

description: Partido no encontrado

content:

application/json:

example:

message: "Partido no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener las estadísticas del partido"

/equipos:

get:

summary: Obtener todos los equipos

responses:

'200':

description: Lista de equipos

content:

application/json:

schema:

type: array

items:

\$ref: '#/components/schemas/Equipo'

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener los equipos"

post:

summary: Añadir un nuevo equipo

requestBody:

content:

application/json:

schema:

\$ref: '#/components/schemas/Equipo'

responses:

'201':

description: Equipo creado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Equipo'

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Datos del equipo no válidos"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al crear el equipo"

/equipos/{id}:

get:

summary: Obtener un equipo específico por ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: Detalles del equipo

content:

application/json:

schema:

\$ref: '#/components/schemas/Equipo'

'404':

description: Equipo no encontrado

content:

application/json:

example:

message: "Equipo no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener el equipo"

put:

summary: Actualizar un equipo

parameters:

- name: id

in: path

required: true

schema:

type: string

requestBody:

content:

application/json:

schema:

\$ref: '#/components/schemas/Equipo'

responses:

'200':

description: Equipo actualizado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Equipo'

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Datos del equipo no válidos"

'404':

description: Equipo no encontrado

content:

application/json:

example:

message: "Equipo no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al actualizar el equipo"

delete:

summary: Eliminar un equipo

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'204':

description: Equipo eliminado exitosamente

'404':

description: Equipo no encontrado

content:

application/json:

example:

message: "Equipo no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al eliminar el equipo"

/equipos/{id}/jugadores:

get:

summary: Obtener todos los jugadores de un equipo específico

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: Lista de jugadores del equipo

content:

application/json:

schema:

type: array

items:

\$ref: '#/components/schemas/Jugador'

'404':

description: Equipo no encontrado

content:

application/json:

example:

message: "Equipo no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener los jugadores del equipo"

/jugadores:

get:

summary: Obtener todos los jugadores

responses:

'200':

description: Lista de jugadores

content:

application/json:

schema:

type: array

items:

\$ref: '#/components/schemas/Jugador'

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener los jugadores"

post:

summary: Añadir un nuevo jugador

requestBody:

content:

application/json:

schema:

\$ref: '#/components/schemas/Jugador'

responses:

'201':

description: Jugador creado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Jugador'

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Datos del jugador no válidos"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al crear el jugador"

/jugadores/{id}:

get:

summary: Obtener un jugador específico por ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: Detalles del jugador

content:

application/json:

schema:

\$ref: '#/components/schemas/Jugador'

'404':

description: Jugador no encontrado

content:

application/json:

example:

message: "Jugador no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al obtener el jugador"

put:

summary: Actualizar un jugador

parameters:

- name: id

in: path

required: true

schema:

type: string

requestBody:

content:

application/json:

schema:

\$ref: '#/components/schemas/Jugador'

responses:

'200':

description: Jugador actualizado exitosamente

content:

application/json:

schema:

\$ref: '#/components/schemas/Jugador'

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Datos del jugador no válidos"

'404':

description: Jugador no encontrado

content:

application/json:

example:

message: "Jugador no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al actualizar el jugador"

delete:

summary: Eliminar un jugador

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'204':

description: Jugador eliminado exitosamente

'404':

description: Jugador no encontrado

content:

application/json:

example:

message: "Jugador no encontrado"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al eliminar el jugador"

/predicciones:

post:

summary: Realizar una predicción de partido

requestBody:

content:

application/json:

schema:

type: object

properties:

features:

type: object

required:

- home_shotsOnGoal
- home_shotsOffGoal
- home_totalShots
- home_blockedShots
- home_shotsInsideBox
- home_shotsOutsideBox

- home_cornerKicks
- home_goalkeeperSaves
- home_totalPasses
- home_accuratePasses
- home_expectedGoals
- away_shotsOnGoal
- away_shotsOffGoal
- away_totalShots
- away_blockedShots
- away_shotsInsideBox
- away_shotsOutsideBox
- away_cornerKicks
- away_goalkeeperSaves
- away_totalPasses
- away_accuratePasses
- away_expectedGoals

properties:

home_shotsOnGoal:

type: number

home_shotsOffGoal:

type: number

home_totalShots:

type: number

home_blockedShots:

type: number

home_shotsInsideBox:

type: number

home_shotsOutsideBox:

type: number

home_cornerKicks:

type: number

home_goalkeeperSaves:

type: number

home_totalPasses:

type: number

home_accuratePasses:

type: number

home_expectedGoals:

type: number

away_shotsOnGoal:

type: number

away_shotsOffGoal:

type: number

away_totalShots:

type: number

away_blockedShots:

type: number

away_shotsInsideBox:

type: number

away_shotsOutsideBox:

type: number

away_cornerKicks:

type: number

away_goalkeeperSaves:

type: number

away_totalPasses:

type: number

away_accuratePasses:

type: number

away_expectedGoals:

type: number

responses:

'200':

description: Predicción realizada con éxito

content:

application/json:

schema:

type: object

properties:

prediction:

type: string

'400':

description: Solicitud incorrecta

content:

application/json:

example:

message: "Faltan características o formato inválido"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al realizar la predicción"

/predicciones/actualizar:

post:

summary: Reentrenar y actualizar el modelo de predicción

responses:

'200':

description: Modelo actualizado correctamente

content:

application/json:

schema:

type: object

properties:

message:

type: string

classification_report:

type: object

confusion_matrix:

type: array

items:

type: integer

cross_val_mean:

type: number

'403':

description: No autorizado

content:

application/json:

example:

message: "No tienes permiso para actualizar el modelo"

'500':

description: Error interno del servidor

content:

application/json:

example:

message: "Error al reentrenar el modelo"

Anexo II – Scripts migración y carga de datos

Scripts/loadTeamsAndPlayers.js:

```
const axios = require('axios');
const { MongoClient } = require('mongodb');
require('dotenv').config();

const apiKey = process.env.API_FOOTBALL_KEY;
const mongoUri = process.env.MONGO_URI;

const delay = ms => new Promise(resolve => setTimeout(resolve, ms));

async function fetchWithRetry(url, params, retries = 3, delayMs = 60000) {
  try {
    const response = await axios.get(url, {
      headers: {
        'x-rapidapi-key': apiKey,
        'x-rapidapi-host': 'api-football-v1.p.rapidapi.com'
      },
      params
    });
    return response.data;
  } catch (error) {
    if (error.response && error.response.status === 429 && retries > 0) {
      console.warn(`Rate limit exceeded, retrying in ${delayMs / 1000} seconds...`);
      await delay(delayMs);
      return fetchWithRetry(url, params, retries - 1, delayMs);
    }
  }
}
```

```
} else {  
  throw error;  
}  
}  
}  
  
async function fetchPlayersForTeam(teamId) {  
  let players = [];  
  let page = 1;  
  let totalPages = 1;  
  
  while (page <= totalPages) {  
    const data = await fetchWithRetry('https://api-football-  
v1.p.rapidapi.com/v3/players', {  
      team: teamId,  
      season: 2023,  
      page: page  
    });  
  
    players.push(...data.response);  
    totalPages = data.paging.total;  
    page++;  
  
    if (page <= totalPages) {  
      await delay(1000);  
    }  
  }  
  
  return players;  
}  
  
async function fetchTeamsAndPlayers(startIndex = 0, endIndex = 5) {  
  try {
```

```
const teamsData = await fetchWithRetry('https://api-football-
v1.p.rapidapi.com/v3/teams', {
  league: 140,
  season: 2023
});
const teams = teamsData.response.slice(startIndex, endIndex);

const players = [];
for (const team of teams) {
  const teamPlayers = await fetchPlayersForTeam(team.team.id);
  players.push(...teamPlayers);
}

return { teams, players };
} catch (error) {
  console.error('Error fetching data:', error);
  throw error;
}
}
```

```
async function loadToMongo(data) {
  const client = new MongoClient(mongoUri);
  try {
    await client.connect();
    const db = client.db('footballDB');
    const teamsCollection = db.collection('teams');
    const playersCollection = db.collection('players');

    await teamsCollection.insertMany(data.teams.map(team => ({
      teamId: team.team.id,
      name: team.team.name,
      code: team.team.code,
      country: team.team.country,
```

```
founded: team.team.founded,  
national: team.team.national,  
logo: team.team.logo,  
venue: {  
  id: team.venue.id,  
  name: team.venue.name,  
  address: team.venue.address,  
  city: team.venue.city,  
  capacity: team.venue.capacity,  
  surface: team.venue.surface,  
  image: team.venue.image  
}  
}}));
```

```
await playersCollection.insertMany(data.players.map(player => ({  
  playerId: player.player.id,  
  name: player.player.name,  
  firstname: player.player.firstname,  
  lastname: player.player.lastname,  
  age: player.player.age,  
  birth: player.player.birth,  
  nationality: player.player.nationality,  
  height: player.player.height,  
  weight: player.player.weight,  
  injured: player.player.injured,  
  photo: player.player.photo,  
  teamId: player.statistics[0].team.id,  
  teamName: player.statistics[0].team.name,  
  position: player.statistics[0].games.position  
})));
```

```
console.log('Teams and players loaded successfully');
```



```
} catch (error) {  
  console.error('Error loading data into MongoDB:', error);  
  throw error;  
} finally {  
  await client.close();  
}  
}  
  
async function main() {  
  try {  
    const startIndex = parseInt(process.argv[2], 10) || 0;  
    const endIndex = parseInt(process.argv[3], 10) || 5;  
    const data = await fetchTeamsAndPlayers(startIndex, endIndex);  
    await loadToMongo(data);  
  } catch (error) {  
    console.error('Error in main function:', error);  
  }  
}
```

```
main().catch(console.error);
```

Scripts/loadMatchesAndStatistics.js:

```
const axios = require('axios');  
const { MongoClient } = require('mongodb');  
require('dotenv').config();  
  
const apiKey = process.env.API_FOOTBALL_KEY;  
const mongoUri = process.env.MONGO_URI;  
  
const delay = ms => new Promise(resolve => setTimeout(resolve, ms));
```

```
async function fetchWithRetry(url, params, retries = 3, delayMs = 60000) {
  try {
    const response = await axios.get(url, {
      headers: {
        'x-rapidapi-key': apiKey,
        'x-rapidapi-host': 'api-football-v1.p.rapidapi.com'
      },
      params
    });
    return response.data;
  } catch (error) {
    if (error.response && error.response.status === 429 && retries > 0) {
      console.warn(`Rate limit exceeded, retrying in ${delayMs / 1000} seconds...`);
      await delay(delayMs);
      return fetchWithRetry(url, params, retries - 1, delayMs);
    } else {
      throw error;
    }
  }
}

async function fetchMatchesAndStatistics(startIndex = 0, endIndex = 20) {
  try {
    const leagueId = 78;
    const season = 2023;

    const matchesData = await fetchWithRetry('https://api-football-
v1.p.rapidapi.com/v3/fixtures', {
      league: leagueId,
      season: season
    });
    const matches = matchesData.response.slice(startIndex, endIndex);
```

```
const matchesWithStats = await Promise.all(matches.map(async match => {  
    const statsData = await fetchWithRetry('https://api-football-  
v1.p.rapidapi.com/v3/fixtures/statistics', {  
        fixture: match.fixture.id  
    });  
    return { ...match, statistics: statsData.response };  
}));  
  
return matchesWithStats;  
} catch (error) {  
    console.error('Error fetching data:', error);  
    throw error;  
}  
}
```

```
async function loadToMongo(matches) {  
    const client = new MongoClient(mongoUri);  
    try {  
        await client.connect();  
        const db = client.db('footballDB');  
        const matchesCollection = db.collection('matches_bundesliga');  
  
        await matchesCollection.insertMany(matches.map(match => ({  
            matchId: match.fixture.id,  
            referee: match.fixture.referee,  
            date: match.fixture.date,  
            venue: match.fixture.venue,  
            status: match.fixture.status,  
            league: match.league,  
            teams: match.teams,  
            statistics: match.statistics.map(stat => ({  
                team: stat.team,  
                stats: {
```

```
shotsOnGoal: getStatValue(stat.statistics, 'Shots on Goal'),
shotsOffGoal: getStatValue(stat.statistics, 'Shots off Goal'),
totalShots: getStatValue(stat.statistics, 'Total Shots'),
blockedShots: getStatValue(stat.statistics, 'Blocked Shots'),
shotsInsideBox: getStatValue(stat.statistics, 'Shots insidebox'),
shotsOutsideBox: getStatValue(stat.statistics, 'Shots outsidebox'),
fouls: getStatValue(stat.statistics, 'Fouls'),
cornerKicks: getStatValue(stat.statistics, 'Corner Kicks'),
offsides: getStatValue(stat.statistics, 'Offsides'),
possession: getStatValue(stat.statistics, 'Ball Possession'),
yellowCards: getStatValue(stat.statistics, 'Yellow Cards'),
redCards: getStatValue(stat.statistics, 'Red Cards'),
goalkeeperSaves: getStatValue(stat.statistics, 'Goalkeeper Saves'),
totalPasses: getStatValue(stat.statistics, 'Total passes'),
accuratePasses: getStatValue(stat.statistics, 'Passes accurate'),
passPercentage: getStatValue(stat.statistics, 'Passes %'),
expectedGoals: getStatValue(stat.statistics, 'expected_goals'),
goalsPrevented: getStatValue(stat.statistics, 'goals_prevented')
}
}))
}});

console.log('Matches and statistics loaded successfully into matches_bundesliga');
} catch (error) {
  console.error('Error loading data into MongoDB:', error);
  throw error;
} finally {
  await client.close();
}
}

function getStatValue(statistics, type) {
```

```
const stat = statistics.find(stat => stat.type === type);  
return stat ? stat.value : null;  
}
```

```
async function main() {  
  try {  
    const startIndex = parseInt(process.argv[2], 10) || 0;  
    const endIndex = parseInt(process.argv[3], 10) || 20;  
    const matches = await fetchMatchesAndStatistics(startIndex, endIndex);  
    await loadToMongo(matches);  
  } catch (error) {  
    console.error('Error in main function:', error);  
  }  
}
```

```
main().catch(console.error);
```

