



PROYECTO SGE

CFGS Desarrollo de Aplicaciones Multiplataforma
Informática y Comunicaciones

**APLICACIÓN CON FASTAPI: CHATBOT CON FASTAPI Y
TENSORFLOW**

Año: 2025

Fecha de presentación: 29-05-2025

Nombre y Apellidos: Pablo Villagrán

Email: pablo.vilgon@educa.jcyl.es

1. Entrenamiento del modelo (Google Colab o local)

- Cargar un dataset de preguntas y respuestas desde un archivo (testdesordenado.txt)
- Preprocesar los datos usando TextVectorization
- Construir un modelo LSTM con Keras que clasifique las preguntas en categorías
- Entrenar el modelo con los datos del archivo
- Añadir comentarios en el código
- Si se está haciendo con Google Colab, guardar localmente los 3 ficheros generados:
 - ✓ modelo.keras
 - ✓ vocabulary.txt
 - ✓ etiquetas.json

2. Creación de una API con FastAPI con las siguientes rutas como mínimo:

- GET /prueba: ruta de prueba para confirmar el funcionamiento del router.
- GET /ping: devuelve el estado de salud de la API
- POST /predict: recibe una pregunta, devuelve la categoría (etiqueta) y guarda el resultado en la base de datos
- Documentar las rutas de la API utilizando herramientas nativas de FastAPI (Swagger o Redoc).

3. Configurar la base de datos con SQLAlchemy:

- Realizar un fichero docker-compose.yml para levantar un contenedor de postgresQL y otro de pgAdmin.
- Crear una tabla “predicciones” con los campos: id (entero, clave primaria, texto (pregunta enviada), categoría (respuesta/predicción del modelo). Usar SQLAlchemy

4. Estructurar el código en módulos: una posible estructura sería la siguiente

- **venv/** # Entorno virtual
- **main.py** # Punto de entrada a la aplicación
- **requirements.txt**
- **docker-compose.yml**
- **app/**
 - **db/**
 - **database.py** # Conexión a PostgreSQL o SQLite
 - **models.py** # Modelos de base de datos con SQLAlchemy
 - **routers/**
 - **predict.py** # Endpoints de la API
 - **schemas/**
 - **predict.py** # Modelos de datos con Pydantic
 - **model.py** # Carga del modelo, tokenizer y etiquetas y realizar la predicción
- **saved_model /** # Contiene: etiquetas.json, modelo.keras, vocabulary.txt

5. Autenticación y seguridad:

- Implementar autenticación basada en **JWT**
- Añadir una ruta /login para autenticar usuarios
- Devolver un token JWT si las credenciales son válidas
- Proteger las rutas sensibles para que solo usuarios autenticados puedan acceder
- Usar dependencias de seguridad proporcionadas por FastAPI

6. Manejo de errores:

- Gestionar errores comunes de forma centralizada.
- Devolver respuestas HTTP claras y adecuadas (400, 401, 403, 404, 500).
- Incluir mensajes personalizados en los errores.

7. Pruebas unitarias:

- Implementar pruebas con pytest u otra herramienta similar.
- Incluir pruebas para validar el comportamiento del endpoint /predict.

8. Validación con Pydantic:

- Utilizar **Pydantic** en la API para definir y validar modelos de datos (BaseModel)
- Asegurarse de que los modelos Pydantic manejen las validaciones necesarias (campos requeridos, formatos, etc.).

9. Otras consideraciones:

- Crear un fichero requirements.txt con las librerías necesarias para el proyecto
- Crear un entorno virtual para desarrollar el proyecto
- Configurar pgAdmin para ver el contenido de la base de datos

Índice

MEMORIA DEL PROYECTO: APARTADOS	6
1. INTRODUCCIÓN	6
2. ESTADO DEL ARTE	7
3. DESCRIPCIÓN GENERAL DEL PROYECTO	11
A. OBJETIVOS	11
B. ENTORNO DE TRABAJO (TECNOLOGÍAS DE DESARROLLO Y HERRAMIENTAS)	11
4. DOCUMENTACIÓN TÉCNICA: ANÁLISIS, DISEÑO, IMPLEMENTACIÓN, PRUEBAS Y DESPLIEGUE	12
A. ANÁLISIS DEL SISTEMA (FUNCIONALIDADES BÁSICAS DE LA APLICACIÓN)	12
B. DISEÑO DE LA BASE DE DATOS	13
C. IMPLEMENTACIÓN.....	14
D. PRUEBAS	16
5. MANUALES	23
A. MANUAL DE USUARIO:	23
B. MANUAL DE INSTALACIÓN:	26
6. CONCLUSIONES Y POSIBLES AMPLIACIONES	28
7. BIBLIOGRAFÍA	28

MEMORIA DEL PROYECTO: apartados

La memoria debe contener una portada (utilizar la que aparece en este documento), un índice y los siguientes apartados:

1. INTRODUCCIÓN

El objetivo de este proyecto es desarrollar un chatbot inteligente capaz de clasificar preguntas en distintas categorías temáticas utilizando un modelo LSTM (Long Short-Term Memory), entrenado con TensorFlow y Keras.

Este modelo permitirá analizar preguntas de texto y asignarlas automáticamente a su categoría correspondiente, facilitando la organización y gestión de la información.

La solución se integrará dentro de una API desarrollada con FastAPI, lo que permitirá exponer sus funcionalidades a través de endpoints accesibles de forma estructurada y segura. Además, se implementará una base de datos relacional utilizando SQLAlchemy para almacenar tanto las preguntas recibidas como sus respectivas predicciones.

El sistema incluirá mecanismos de:

- Seguridad y autenticación de usuarios mediante JWT.
- Validación automática de datos a través de tipado estático y modelos Pydantic.
- Documentación interactiva de la API generada automáticamente mediante Swagger.
- Pruebas automáticas para garantizar el correcto funcionamiento de la lógica y las rutas de la API.

Con este enfoque, se busca construir una herramienta robusta, escalable y fácilmente integrable en otros sistemas o servicios.

2. ESTADO DEL ARTE

- **VPS**: es un tipo de alojamiento que proporciona un entorno virtualizado en la nube, simulando el funcionamiento de un servidor físico. Permite mayor control, personalización y escalabilidad respecto a los alojamientos compartidos.
- **SSH**: es un protocolo de red criptográfico para operar servicios de red de forma segura a través de una red no protegida. Cualquier servicio de red puede protegerse con SSH.
- **Servidor web**: sirven para almacenar contenidos de internet y facilitar su disponibilidad de forma constante y segura.
- **Protocolo HTTP, métodos**: el protocolo HTTP (HyperText Transfer Protocol) es el formato de comunicación entre el cliente y el servidor web. Este, define varios métodos que indican la acción entre el cliente desea realizar sobre un recurso.
 - **POST**: envía datos al servidor para crear un nuevo recurso.
 - **GET**: solicita datos de un recurso específico. No modifica este, solo lo obtiene.
 - **PUT**: envía datos al servidor para actualizar completamente un recurso existente.
 - **DELETE**: solicita al servidor eliminar un recurso específico.
 - **PATCH**: modifica parcialmente un recurso existente.
 - **HEAD**: hace la misma función que GET, pero solo devuelve los encabezados.
 - **OPTIONS**: devuelve los métodos HTTP permitidos para un recurso.

- **Apache, nginx:**

Característica	Apache	Nginx
Tipo de servidor	Servidor web tradicional	Servidor web y proxy inverso
Arquitectura	Basada en procesos o hilos	Basada en eventos
Rendimiento	Bueno, pero consume más recursos	Alto, eficiente con varias conexiones
Configuración dinámica	Fácil con .htaccess	No permite .htaccess, requiere config centralizada
Uso común	Sitios web dinámicos, compatible con PHP	Contenido estático, balanceo de carga API

- Indicar 3 lenguajes de programación del lado del servidor:
 - PHP
 - Java
 - Python
 - C#
- **Tensorflow:** es una biblioteca de código abierto, desarrollada por Google Brain para realizar tareas de machine learning y depp learning. Es una de las más utilizadas a nivel industrial gracias a su flexibilidad, escalabilidad y soporte. Permite construir modelos complejos desde cero o usando interfaces como Keras.
- **Keras:** es una biblioteca de código abierto de alto nivel para machine learning y el deep learning. Se caracteriza por ser una biblioteca de alto nivel, flexible, que permite a los desarrolladores crear y entrenar modelos de aprendizaje automatico de manera rápida y eficiente. Actúa como interfaz amigable para TensorFlow.

- **Modelo LSTM**: es un tipo de red neuronal recurrente (RNN) diseñada para el trabajo con datos secuenciales. Estas son capaces de recordar información durante largos periodos de tiempo, lo que las hace idóneas para tareas como:
 - Análisis de sentimientos
 - Reconocimiento de voz
 - Procesamiento de lenguaje natural

- **Numpy**: es una biblioteca para python especializada en el manejo de arrays y matrices multidimensionales. Es la base de muchas bibliotecas científicas en Python y permite realizar operaciones vectorizadas de forma eficiente.

- **Matplotlib**: es una biblioteca de visualización en Python que permite crear gráficos y representaciones visuales de datos, como barras, histogramas... el módulo más usado es pyplot.

- **Google Colab**: es un entorno gratuito ofrecido por Google. Permite escribir y ejecutar código en python directamente en el navegador, con soporte para bibliotecas como TensorFlow, NumPy...

- **Definición de arquitectura de microservicios**: es un enfoque de desarrollo de software en el que una app se construye como un conjunto de servicios independientes, cada uno se comunica entre si a través de APIs.

- **Definición de API**: es un conjunto de reglas y protocolos que permite la comunicación entre diferentes sistemas. En una API REST, los clientes pueden realizar solicitudes a un servidor para crear, leer, actualizar o eliminar datos. Hay varios tipos de APIs:
 - **REST**(Representational State Transfer): basada en HTTP, utiliza JSON como formato de datos.
 - **SOAP** (Simple Object Protocol): utiliza XML y es más complejo.

- **GraphQL:** permite consultas personalizadas en una única solicitud.

Nosotros utilizaremos API REST, ya que es simple, ligera y ampliamente utilizada actualmente.

- Estructura de una API (protocolo utilizado, métodos, partes de las URL de una API):

Método	Descripción
GET	Obtiene datos
POST	Crea un nuevo recurso
PUT	Actualiza un recurso
DELETE	Elimina un recurso

Partes de las url, ejemplo:

<https://api.proyectosge.com/tiendas/5/productos>

- Protocolo: https:// -> Indica que usa HTTP seguro.
- Dominio: api.proyectosge.com -> Direccion del servidor
- Recurso: /tiendas/ -> Endpoint para gestionar tiendas.
- ID: /5/ -> Identificador de una tienda especifica.
- Sobrecurso: /productos/ -> Lista de productos de la tienda con ID 5.

- Formas de crear una API en python

Faskl:

- Es un microfameworl ligero para crear APIs en Python.
- Fácil de aprender y usar.
- No incluye validación automática de datos ni documentación automática.

FastAPI:

- Mas rápido que flask gracias a Starlette y Pydantic.
- Soporta tipado estático, lo que mejora la validación de datos.
- Genera documentación automática con Swagger.

Nosotros utilizaremos FastAPI ya que es más simple y eficiente que flask. Genera documentación automática con Swagger. Y, por último, facilita la validación de datos y autenticación con JWT.

3. DESCRIPCIÓN GENERAL DEL PROYECTO

a. Objetivos

Descripción de lo que se ha pretendido alcanzar con el proyecto

El objetivo de este proyecto ha sido desarrollar un sistema capaz de clasificar preguntas de texto según su categoría, utilizando técnicas de procesamiento de lenguaje natural y redes neuronales (LSTM).

Se ha buscado crear una solución que automatice este proceso de forma eficiente, integrando el modelo dentro de una API con FastAPI, y utilizando una base de datos para guardar tanto las preguntas como sus resultados.

Además, se han implementado mecanismos de seguridad, validación y documentación, para que el sistema sea robusto, seguro y fácil de integrar en otros entornos.

b. Entorno de trabajo (tecnologías de desarrollo y herramientas)

Python ha sido el lenguaje seleccionado para nuestro desarrollo debido a su simplicidad, versatilidad y porque lo hemos estado utilizando a lo largo del curso. Además, cuenta con numerosos frameworks y bibliotecas que facilitan el desarrollo de APIs y aplicaciones modernas.

Google Colab es una herramienta gratuita proporcionada por Google que permite escribir y ejecutar código en Python directamente desde el navegador. No requiere configuración local y ofrece acceso gratuito a recursos de computación como CPU y GPU. Una de sus principales ventajas es su integración con Google Drive, lo que facilita el guardado y la compartición de archivos. Además, cuenta con soporte completo para bibliotecas como NumPy, TensorFlow, entre otras.

FastAPI es un framework moderno y eficiente para construir APIs en Python. Basado en Starlette y Pydantic, permite la validación automática de datos y una ejecución rápida. Nos proporciona un alto rendimiento y un soporte nativo para la autenticación mediante JWT.

Para el control de acceso y autenticación de usuarios, se ha utilizado JWT junto con la librería PyJWT, que permite una autenticación segura sin necesidad de sesiones en la base de datos.

Los tokens generados son compactos, seguros y se integran fácilmente con FastAPI.

La documentación de la API se ha llevado a cabo mediante Swagger UI, que facilita tanto la visualización como la prueba de los distintos endpoints.

Por último, el desarrollo del proyecto se ha realizado en Visual Studio Code, un editor de código potente y ligero. Para facilitar el desarrollo, se han utilizado varias extensiones, como la de soporte para Python.

4. DOCUMENTACIÓN TÉCNICA: ANÁLISIS, DISEÑO, IMPLEMENTACIÓN, PRUEBAS Y DESPLIEGUE

a. Análisis del sistema (funcionalidades básicas de la aplicación)

Describir las funcionalidades de la API

La aplicación consiste en una API REST desarrollada con FastAPI. Su objetivo principal es recibir preguntas en texto plano, clasificarlas automáticamente en una categoría mediante un modelo LSTM, y almacenar el resultado en una base de datos relacional.

Las funcionalidades implementadas en la API son las siguientes:

- Rutas de prueba
 - **GET /prueba**: Retorna un mensaje simple de confirmación para verificar que la API está funcionando correctamente.
 - **GET /ping**: Retorna un estado ok, útil para comprobar si el servidor está activo (endpoint típico para pruebas de conectividad).
- Clasificación de preguntas
 - **POST /predict**: Este endpoint recibe una pregunta en formato JSON. El cuerpo de la petición debe contener un campo texto.
 - El modelo LSTM procesa el texto para predecir su categoría temática.

El resultado de la predicción se devuelve al usuario y se guarda en la base de datos con el texto original y la categoría generada.

- Gestión de base de datos

Cada predicción realizada se almacena en una tabla llamada Prediccion, con los campos texto (pregunta) y categoría (resultado del modelo).

El sistema utiliza SQLAlchemy para gestionar la conexión y operaciones con la base de datos.

- Estructura del código

El enrutador de FastAPI está organizado en un módulo separado (APIRouter) para facilitar la escalabilidad.

El acceso a la base de datos se gestiona mediante una función `get_db()` que usa Depends, lo cual garantiza una conexión controlada y su cierre automático tras cada operación.

- Modelo de predicción

El modelo de Machine Learning se encuentra en el módulo `model.py`, y se invoca desde la ruta `/predict` mediante la función `realizar_prediccion`.

Esta estructura proporciona una base robusta para añadir futuras funcionalidades, como autenticación, historial por usuario, o feedback sobre las predicciones.

b. Diseño de la base de datos

Explicar el diseño de la base de datos. Incluir un diagrama con las tablas y sus relaciones

predicciones	
ID	INTEGER (PK)
TEXTO	STRING
CATEGORIA	STRING

c. Implementación

Explicar la estructura del código, principales librerías utilizadas, Google Colab. Incluir explicaciones y capturas del código de todas las partes de la aplicación.

La implementación del proyecto se ha realizado utilizando Python como lenguaje principal y FastAPI como framework para construir la API. La estructura del proyecto está organizada de forma modular, facilitando el mantenimiento y la escalabilidad del sistema.

- Estructura del proyecto

- main.py: punto de entrada de la aplicación.
- app/: contiene la lógica principal de la API.
- bd/: contiene los archivos para la conexión a la base de datos:
- database.py: gestiona la conexión con la base de datos mediante SQLAlchemy.
- models.py: define el modelo Prediccion, que representa las predicciones almacenadas.
- routers/: contiene las rutas de la API.
- predict.py: define las rutas para probar la API y hacer predicciones.
- schemas/: contiene los esquemas de datos con Pydantic para validación.
- predict.py: define los modelos de entrada y salida de datos de la API.
- model.py: encapsula la lógica de carga del modelo LSTM y la predicción.
- saved_model/: carpeta donde se almacena el modelo entrenado (modelo.keras) y sus recursos (vocabulary.txt, etiquetas.json).
- docker-compose.yml: permite el despliegue del servicio de manera sencilla.
- requirements.txt: archivo con las dependencias necesarias para el entorno virtual.

- Librerías utilizadas
 - FastAPI: para la creación de la API web.
 - Uvicorn: como servidor ASGI para desplegar FastAPI.
 - TensorFlow: para el modelo LSTM de clasificación de preguntas.
 - NumPy: para procesamiento de datos numéricos.
 - Pydantic: para validación de datos de entrada/salida.
 - SQLAlchemy: ORM para la gestión de la base de datos relacional.
 - psycopg2-binary: para la conexión con la base de datos PostgreSQL.

- Entrenamiento y pruebas en Google Colab

Para entrenar y probar el modelo, se utilizó Google Colab, lo cual permitió aprovechar aceleración por GPU y facilitar el desarrollo. El flujo fue el siguiente:

- Entrenamiento del modelo LSTM con TensorFlow y Keras.
- Guardado del modelo como modelo.keras.
- Exportación del vocabulario del tokenizer (vocabulary.txt) y las etiquetas (etiquetas.json).
- Prueba local en Colab simulando un chatbot

A continuación, te muestro el enlace para que puedas ver lo anterior.

https://colab.research.google.com/drive/1zglzIXiAwSEQ_zNXhE_nb8WZPbh7kSHe?authuser=1

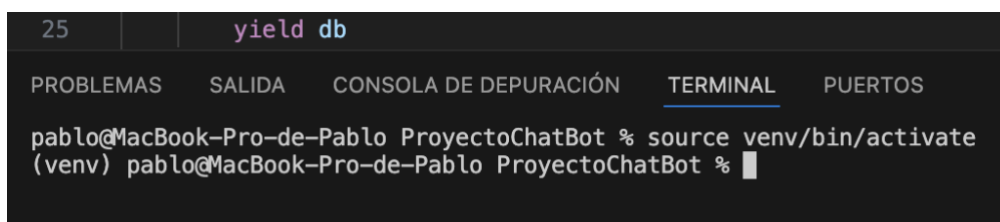
d. Pruebas

Realizar y documentar un mínimo de 2 pruebas: una con swagger y otra con pgadmin para ver el contenido de la tabla de la base de datos

Swagger

Para realizar la prueba de funcionamiento de nuestra API mediante swagger, seguiremos los siguientes pasos:

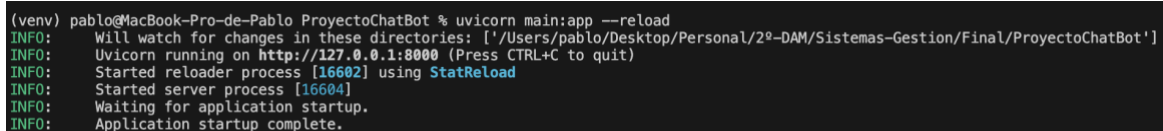
1. Levantamos el contenedor Docker que contiene nuestra base de datos PostgreSQL, asegurándonos de que el servicio está activo.
2. Activamos el entorno virtual:
 - En Windows: `.\venv\Scripts\activate.ps1`
 - En macOS: `source venv/bin/activate`



```
25 | yield db
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
pablo@MacBook-Pro-de-Pablo ProyectoChatBot % source venv/bin/activate
(venv) pablo@MacBook-Pro-de-Pablo ProyectoChatBot %
```

3. Ejecutamos la aplicación con el siguiente comando

Uvicorn main:app - -reload



```
(venv) pablo@MacBook-Pro-de-Pablo ProyectoChatBot % uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['/Users/pablo/Desktop/Personal/2ª-DAM/Sistemas-Gestion/Final/ProyectoChatBot']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [16602] using StatReload
INFO: Started server process [16604]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

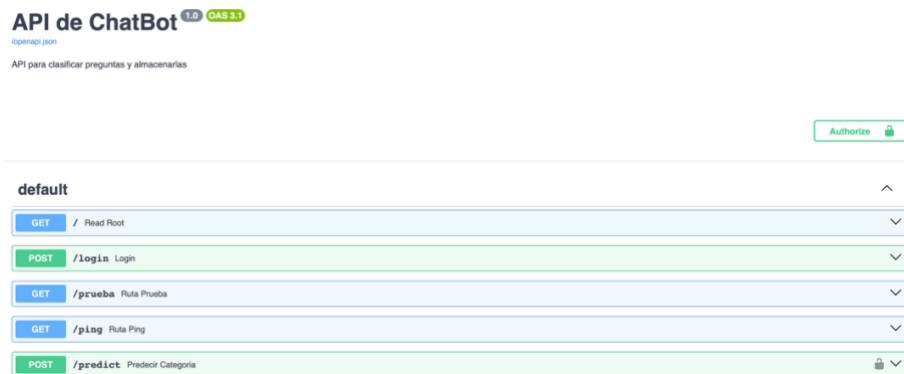
4. Podemos observar cómo nos aparece una url, si accedemos nos aparece lo siguiente:



Tenemos un breve mensaje donde nos dice claramente que añadiendo `/docs` en la url nos lleva a la página de swagger.



5. En esta interfaz podemos ver todos los endpoints de la API. Para comenzar las pruebas:



a. Login

Vamos al endpoint de este, y pulsamos en “Try it out” e introducimos unas credenciales validas de un usuario.

The screenshot shows the configuration for the 'POST /login Login' endpoint. The 'Parameters' tab is active, showing 'No parameters'. The 'Request body' tab is also active, showing a form with the following fields:

- grant_type: password
- username: usuario
- password: 1234
- scope: (empty)
- client_id: (empty)
- client_secret: (empty)

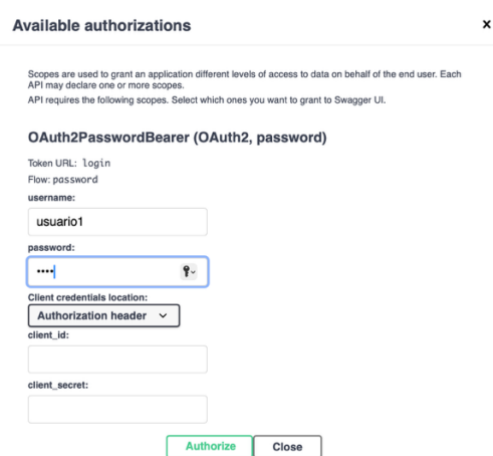
There are 'Cancel' and 'Reset' buttons at the top right, and an 'Execute' button at the bottom.

Si esto es correcto, nos devolverá un token de acceso JWT.



b. Autenticación en swagger

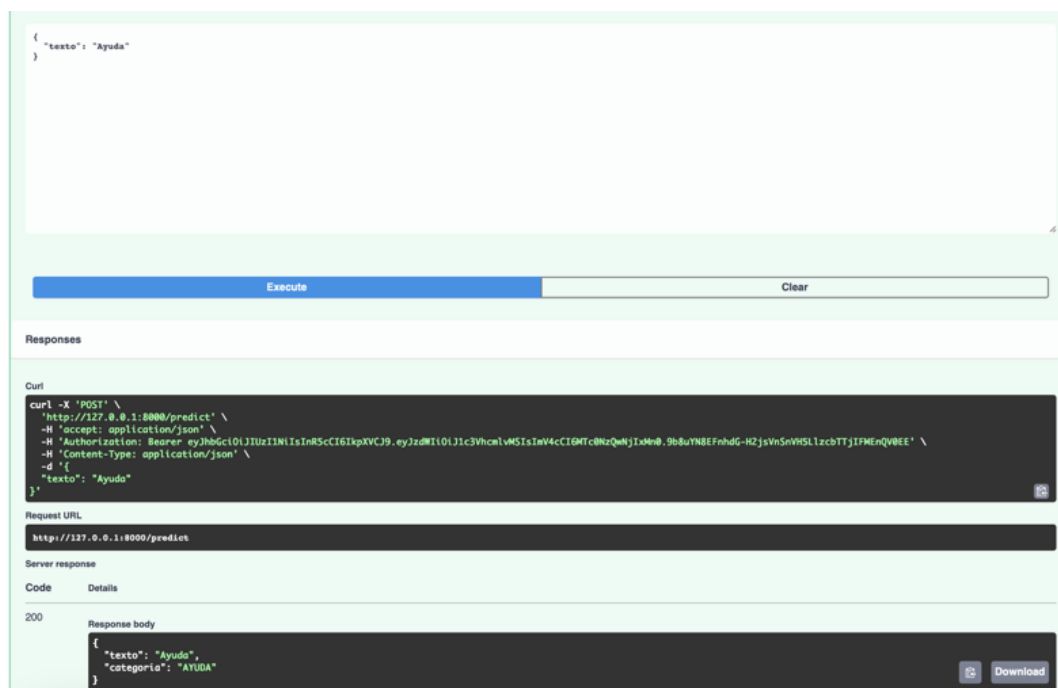
Una vez nos da el token, automáticamente debemos ir al icono del candado. Pulsando en él nos aparecerá lo siguiente. Una vez aquí debemos poner el mismo usuario y contraseña y pulsar en Authorize. El token lo agrega automáticamente.



The image shows a modal window titled "Available authorizations" with a close button (X) in the top right corner. Inside the modal, there is explanatory text about scopes. Below this, the "OAuth2PasswordBearer (OAuth2, password)" method is selected. The "Token URL" is "login" and the "Flow" is "password". The "username" field contains "usuario1" and the "password" field contains masked characters "...." with a visibility toggle icon. The "Client credentials location" is set to "Authorization header" via a dropdown menu. The "client_id" and "client_secret" fields are empty. At the bottom, there are "Authorize" and "Close" buttons.

Una vez pulsamos en authorize, cerramos la pestaña y lo tenemos.

6. Ahora estamos autenticados y podemos probar los endpoints protegidos como /predict, enviando preguntas para que el modelo las clasifique.

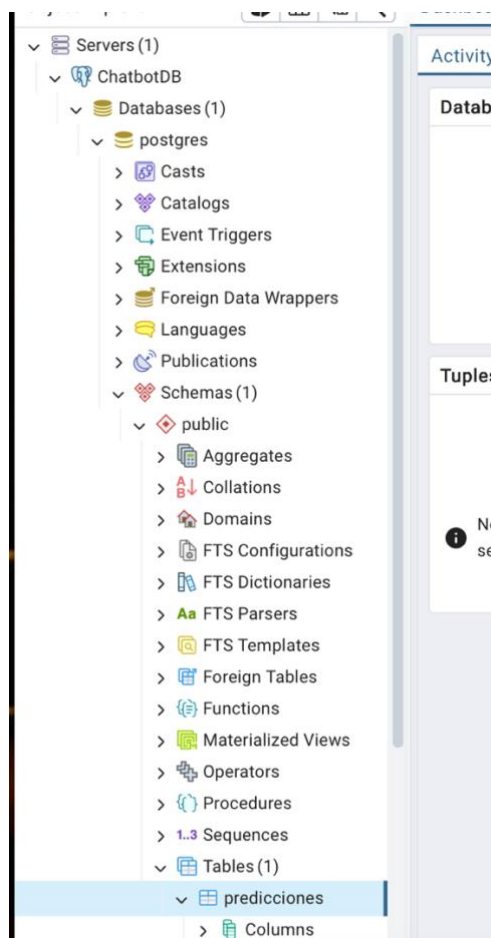


The image shows the Swagger UI interface. At the top, a JSON body is entered: `{ "texto": "Ayuda" }`. Below the input area are "Execute" and "Clear" buttons. Under the "Responses" section, the "Curl" tab is active, displaying a curl command for a POST request to `http://127.0.0.1:8000/predict` with an Authorization header. The "Request URL" is `http://127.0.0.1:8000/predict`. The "Server response" section shows a status code of 200 and a response body: `{ "texto": "Ayuda", "categoria": "AYUDA" }`. A "Download" button is present next to the response body.

Pgadmin

Para confirmar que los datos enviados a través de la API se almacenan correctamente, utilizamos PgAdmin, la interfaz gráfica de PostgreSQL:

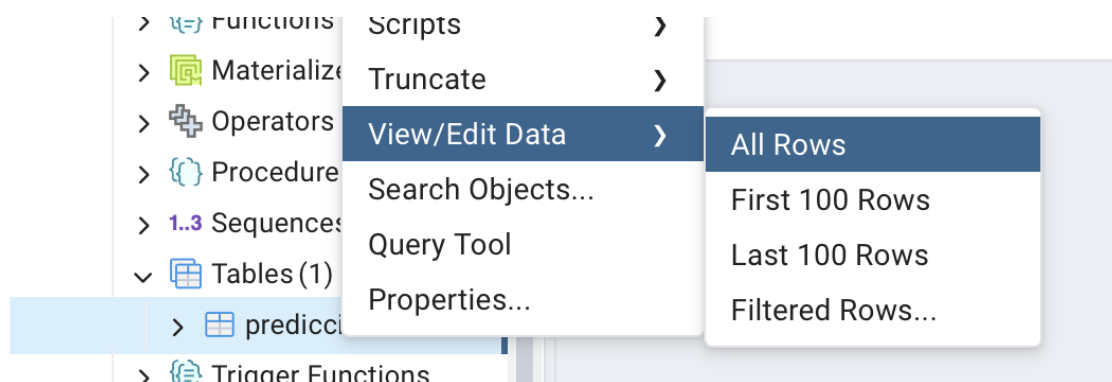
1. Abrir PgAdmin e iniciar sesión con el usuario y contraseña configurados en tu contenedor Docker.
2. En el panel izquierdo, desplegamos:
 - Servers → el nombre de nuestro servidor → Databases → seleccionamos la base de datos de nuestro proyecto (predicciones).



3. Dentro de la base de datos, accedemos a:

Schemas → public → Tables

Aquí vemos como muestra nuestra tabla llamada predicciones. Seguidamente hacemos clic derecho sobre la tabla de predicciones → View/Edit Data → All Rows.



4. Se abre una tabla con todas las filas guardadas. Aquí podremos ver:

- El texto de la pregunta enviada.
- La categoría devuelta por el modelo.

id	texto	categoria
1	Hola	AYUDA
2	Me he roto un brazo	AYUDA
3	No funciona el pc	AYUDA
4	Se rompió el lavaplatos	AYUDA
5	Okey	AYUDA
6	Ayuda	AYUDA
7	Adios	AYUDA
8	Que tal todo	SERVICIO_TECNICO
9	Que tal todo, necesito ayuda	SERVICIO_TECNICO
10	Hoy es lunes	AYUDA
11	Hoy	AYUDA
12	Ayer	AYUDA
13	hola que tal	SERVICIO_TECNICO
14	ayudame, no funciona el pc	AYUDA
15	ayudame, se callo el coche en un barranco	AYUDA

Esto verifica que:

- El endpoint está funcionando.
- La base de datos está conectada correctamente.
- Los datos se almacenan de forma persistente en PostgreSQL.

1.1. Despliegue de la aplicación

En la fase actual , la aplicación será desplegada y ejecutada de forma local, lo que permite un entorno controlado para realizar pruebas, validaciones y mejoras continuas sin necesidad de conexión externa. Esta decisión se debe a que todavía no contamos con los conocimientos necesarios para realizar un despliegue en un entorno de producción remoto.

No obstante, contemplamos la posibilidad de alojar la aplicación en un servidor proporcionado por el instituto si el tiempo y el aprendizaje lo permiten en las próximas semanas. Este paso permitiría realizar pruebas de acceso remoto, evaluar el rendimiento bajo distintas condiciones y preparar la aplicación para un entorno real.

El despliegue local se realiza a través de FastAPI, ejecutando el servidor Uvicorn, lo que nos proporciona una ejecución ligera, rápida y compatible con los entornos más comunes.

Prueba test

Demos crear una carpeta llamada Test, e incluir el siguiente código:

```
test_main.py X
test > test_main.py > ...
1  from fastapi.testclient import TestClient
2  from main import app
3
4  client = TestClient(app)
5
6  # Credenciales correctas (ajusta según tu fake_users_db)
7  USERNAME = "usuario1"
8  PASSWORD = "1234"
9
10 def test_read_root():
11     response = client.get("/")
12     assert response.status_code == 200
13     assert response.json() == {"Mensaje": "Bienvenido a la API de ChatBot. Ve a /docs para la documentacion."}
14
15 def test_login_correcto():
16     response = client.post("/login", data={"username": USERNAME, "password": PASSWORD})
17     assert response.status_code == 200
18     assert "access_token" in response.json()
19
20 def test_login_fallido():
21     response = client.post("/login", data={"username": "incorrecto", "password": "incorrecto"})
22     assert response.status_code == 401
23
24 def test_predict():
25     # Primero, logueamos para obtener el token
26     response = client.post("/login", data={"username": USERNAME, "password": PASSWORD})
27     assert response.status_code == 200
28     token = response.json()["access_token"]
29
30     # Ahora, hacemos la predicción usando el token
31     headers = {"Authorization": f"Bearer {token}"}
32     pregunta = {"texto": "¿Qué es una variable en programación?"}
33     response = client.post("/predict", json=pregunta, headers=headers)
34     assert response.status_code == 200
35     assert "texto" in response.json()
36     assert "categoria" in response.json()
37
38 def test_predict_sin_token():
39     pregunta = {"texto": "¿Qué es una variable en programación?"}
40     response = client.post("/predict", json=pregunta)
41     assert response.status_code == 401
42
```

Después de tener todo listo, debemos ejecutar el siguiente comando para correr las pruebas unitarias correctamente:

- PYTHONPATH=. Pytest
-

Esto le indica a Python que el directorio actual (.) es el directorio raíz del proyecto, permitiendo que pytest encuentre e importe correctamente los módulos de nuestro proyecto (por ejemplo, main.py, app/, etc.). Si no especificamos PYTHONPATH, podríamos encontrarnos con errores de "ModuleNotFoundError" al ejecutar las pruebas, porque pytest no sabría desde dónde resolver las rutas de importación.

```
(venv) pablo@MacBook-Pro-de-Pablo ProyectoChatBot % PYTHONPATH=, pytest
===== test session starts =====
platform darwin -- Python 3.11.6, pytest-8.3.5, pluggy-1.6.0
rootdir: /Users/pablo/Desktop/Personal/2º-DAM/Sistemas-Gestion/Final/ProyectoChatBot
plugins: anyio-4.9.0
collected 5 items

test/test_main.py ..... [100%]

===== warnings summary =====
venv/lib/python3.11/site-packages/passlib/utils/_init_.py:854
/Users/pablo/Desktop/Personal/2º-DAM/Sistemas-Gestion/Final/ProyectoChatBot/venv/lib/python3.11/site-packages/passlib/utils/_init_.py:854: DeprecationWarning: 'crypt'
is deprecated and slated for removal in Python 3.13
  from crypt import crypt as _crypt

app/bd/database.py:14
/Users/pablo/Desktop/Personal/2º-DAM/Sistemas-Gestion/Final/ProyectoChatBot/app/bd/database.py:14: MovedIn20Warning: The ``declarative_base()`` function is now availabl
e as sqlalchemy.orm.declarative_base(). (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 5 passed, 2 warnings in 4.07s =====
```

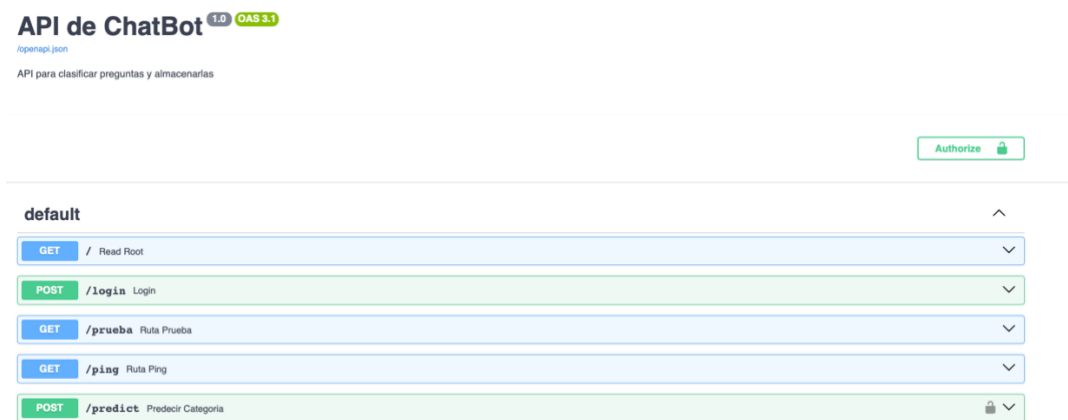
5. MANUALES

a. Manual de usuario:

Breve explicación de cómo se usa la API con capturas de pantalla

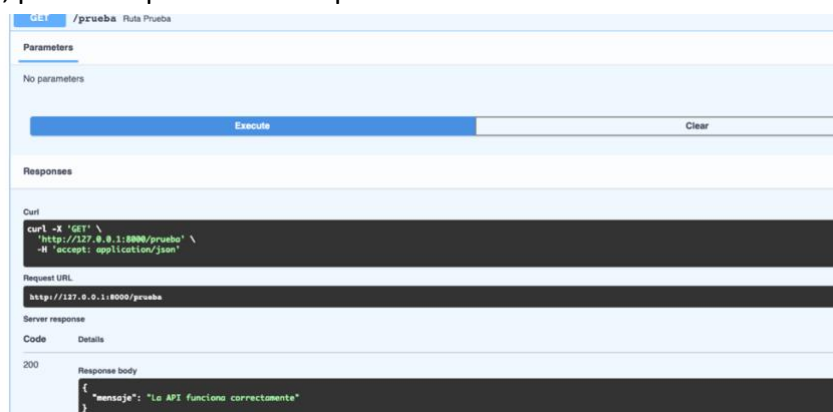
1. Acceso a la aplicación

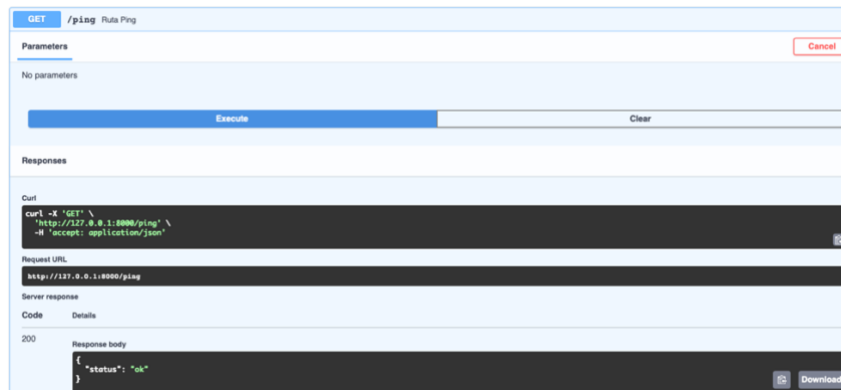
Una vez que hemos arrancado nuestro contenedor de Docker y activado el entorno virtual, accedemos a la URL que se muestra en consola al iniciar la app. Esta URL nos abre una interfaz similar a la siguiente:



2. Pruebas básicas (GET)

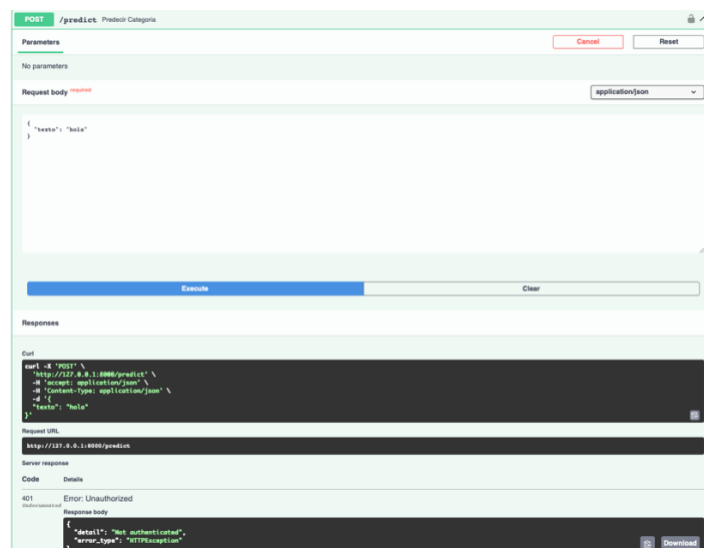
Al abrir la interfaz, podemos probar los endpoints básicos:





3. Intento de POST sin autenticación

Si intentamos hacer una solicitud POST al endpoint /predecir-categoria, obtendremos un error de autenticación, ya que no hemos iniciado sesión.



4. Registro y autenticación

Para autenticarnos:

- Haz clic en el icono del candado en la parte superior izquierda.
- Ingresa las siguientes credenciales:
 - Username: usuario1
 - Password: 1234
- Una vez autenticado correctamente, cierra la ventana emergente.

Available authorizations

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes.

API requires the following scopes. Select which ones you want to grant to Swagger UI.

OAuth2PasswordBearer (OAuth2, password)

Authorized

Token URL: login

Flow: password

username: usuario1

password: *****

Client credentials location: basic

client_secret: *****

Logout

Close

5. Prueba del POST autenticado

Ahora que estamos autenticados, podemos volver al endpoint POST /predecir-categoria, probarlo, y verificar que funciona correctamente.

POST

/predict

Practical Category

Parameters

No parameters

Request body application/json

```
{
  "texto": "hola"
}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  http://127.0.0.1:8080/predict \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bm90cmVzcCI6ImF1dG8iLCJpYXN0ZWQiOiJ1OTk0OTY3LTM0LWYyZjY3OTAwIiwiaWF0IjoxNjU0MjY0MDAwfQ' \
  -d '{
    "texto": "hola"
  }'
```

Request URL:

```
http://127.0.0.1:8080/predict
```

Server response

Code

Details

200

Response body

```
{
  "texto": "hola",
  "categoria": "ATUSA"
}
```

Como podemos comprobar, funciona correctamente.

b. Manual de instalación:

Pasos que seguir para desplegar la aplicación: servidor local o remoto, otras formas de despliegue...

Pasos que seguir para desplegar la aplicación en servidor local

Antes de desplegar la aplicación, asegúrate de tener instalado:

- Docker y Docker Compose
- Python 3.10 o superior
- Virtualenv
- Navegador web (para acceder a Swagger y pgAdmin)

1. Crear entorno virtual y activarlo

En macOS/Linux:

- `python3 -m venv venv`
- `source venv/bin/activate`

En Windows:

- `python -m venv venv`
- `.\venv\Scripts\activate.ps1`

2. Instalar dependencias

`pip install -r requirements.txt`

3. Levantar los contenedores con Docker

`docker-compose up -d`

Esto levantará dos servicios:

- PostgreSQL en el puerto 5432
- pgAdmin en el puerto 80

4. Crear las tablas en la base de datos

Las tablas se crean automáticamente al ejecutar el proyecto gracias a:

```
Base.metadata.create_all(bind=engine)
```

5. Ejecutar el servidor FastAPI

```
uvicorn app.main:app --reload
```

Esto arrancará la API en <http://127.0.0.1:8000>

6. Acceder a la aplicación

Swagger (documentación y pruebas)

- Visita: <http://127.0.0.1:8000/docs>

Desde allí puedes autenticarte, enviar preguntas, y visualizar la documentación de la API.

Pasos que seguir para desplegar la aplicación en pgAdmin

- Visita: <http://localhost>
- Accede con:
 - o Email: pgadmin4@pgadmin.org
 - o Password: admin
- Conéctate al servidor PostgreSQL usando:
 - o Host: db
 - o Usuario: odoo
 - o Contraseña: odoo
 - o Base de datos: postgres

6. CONCLUSIONES Y POSIBLES AMPLIACIONES

Dificultades encontradas en el desarrollo de la aplicación, grado de satisfacción en el trabajo realizado, aprendizaje...

Posibles ampliaciones: indicar al menos una

- Dificultades

Durante el desarrollo de este proyecto me he encontrado con varios retos, sobre todo al principio, cuando prácticamente todo era nuevo para mí. Por ejemplo, ni siquiera sabía lo que era Google Colab o cómo se entrenaba un modelo de red neuronal desde cero.

Al principio parecía imposible, pero poco a poco fui entendiendo mejor cada parte y encajando las piezas.

A pesar de las dificultades, estoy bastante satisfecho con el trabajo realizado. He aprendido muchísimo, que va desde el entrenamiento de modelos de machine learning hasta el despliegue de una API funcional con documentación y validación. Me ha servido para entender cómo se integran distintas tecnologías en un solo proyecto y cómo organizarlo para que sea fácil de mantener y reutilizar.

- Posibles ampliaciones

Una posible ampliación del proyecto sería añadir una interfaz gráfica (web o móvil) que permita a los usuarios interactuar con el sistema de forma más amigable, sin necesidad de utilizar herramientas externas como Postman. También se podría mejorar el modelo de clasificación incorporando técnicas más avanzadas, como transformers (por ejemplo, BERT), para aumentar la precisión y capacidad de comprensión del sistema. Por último, se podría internacionalizar la aplicación para admitir preguntas en varios idiomas y aplicar técnicas de traducción automática si fuese necesario.

7. BIBLIOGRAFÍA

<https://fastapi.tiangolo.com/>

<https://fastapi.tiangolo.com/features/#interactive-api-docs>

<https://chatgpt.com>

Normas de entrega: FECHA DE ENTREGA: 29 de mayo de 2025

- Es necesario incluir todos los apartados de la memoria. Extensión mínima: 20 páginas
- El proyecto tiene que estar subido a Github como público: El repositorio tiene que contener el código fuente de la aplicación y un archivo README con el siguiente contenido: Título del proyecto, descripción, enlace a la memoria del proyecto en formato PDF.
- Nombre del archivo de la memoria:
 - Apellido1_Apellido2_Nombre_Memoria_2Proyecto3trimestre_DAM2425.pdf

Presentación oral del proyecto en clase: realizar una presentación de Powerpoint o guión para explicar la memoria. También es necesario demostrar que funciona la aplicación. Duración: 10 minutos

RÚBRICA UTILIZADA PARA CORREGIR:

DESARROLLO DEL CÓDIGO		0,1	0,2	0,3	0,4	0,5	TOTAL
1	Archivos del entrenamiento del modelo generados correctamente: modelo.keras, vocabulary.txt y etiquetas.json						0
2	Entrenamiento del modelo documentado con comentarios en Google Colab						0
3	Estructura de FastAPI creada: /ping, /prueba, /predict						0
4	Documentación clara en código y rutas de la API						0
5	Docker-compose funcional con contenedores de PostgreSQL y pgAdmin						0
6	Tabla predicciones correctamente definida con SQLAlchemy						0
7	Estructura modular del proyecto: app, db, routers, schemas, main						0
8	Creación de requirements.txt y entorno virtual						0
9	Fichero model.py correcto						0
10	Ruta /login implementada y funcional						0
11	Generación y validación de Token JWT						0
12	Captura de errores comunes y uso de excepciones						0
13	Mensajes claros y códigos HTTP correctos (400, 401...)						0
14	Presencia de tests con pytest u otra herramienta						0
15	Prueba documentada de /predict y funcionamiento correcto						0
16	Modelos definidos correctamente con BaseModel						0

APARTADOS DE LA MEMORIA		0,1	0,2	0,3	0,4	0,5	TOTAL
17	Portada, índice, introducción, descripción general del proyecto						0
18	Estado del arte bien redactado y profundización en la investigación						0
19	Explicación detallada del desarrollo técnico (análisis, diseño, implementación, pruebas, despliegue)						0
20	Manual de usuario y de instalación claros y completos						0
21	Conclusiones personales y propuesta de ampliaciones, bibliografía						0
REPOSITORIO GITHUB		0,1	0,2	0,3	0,4	0,5	TOTAL
22	Repositorio público accesible y bien organizado						0
23	README con título, descripción y enlace a la memoria en PDF						0
PRESENTACIÓN ORAL		0,1	0,2	0,3	0,4	0,5	TOTAL
24	Claridad y orden en la exposición (uso de guión o presentación)						0
25	Demostración del funcionamiento de la aplicación						0
26	Grado de conocimiento y dominio de los contenidos expuestos						0
					TOTAL		0
Por cada día de retraso: 1 punto menos respecto a la puntuación total							
Copia de proyectos: la nota será 0 para todos los implicados							