PROYECTO: DESARROLLO DE APIS CON PYTHON

FECHA DE MAXIMA DE ENTREGA : Mayo 16, 2024

Para la entrega del proyecto del módulo, los estudiantes pueden elegir entre dos opciones según sus preferencias:

OPCION 1 : Desarrollo de API conectado a una BD

Para este proyecto, deberán utilizar Docker (ya sea localmente o en la nube) para ejecutar dos contenedores:

1. API usando Flask o FastAPI desarrollado por el estudiante.
2. Una base de datos relacional o no relaciones según preferencia (MySQL, Postgres, MongoDB,etc). Usar imagen de Docker Hub. Se deberá crear un dataset llamado my_collections, donde se tendra una tabla my_movies con los campos “ID”, “Autor”, “Decripcion”, y “Fecha de Estreno”

El API desarrollado debe implementar los métodos GET, POST, PUT y DELETE, y cada uno de ellos deberá realizar lo siguiente:

- Metodo GET: Leer datos de la BD. Puede codificarlo para obtener solo un datos de alguna fila con un id especifico, o puede codificarlo para obtener todo el dataset.
- Metodo POST: Escribir datos en la BD. Codificarlo para que agregar datos a la tabla my_movies y que el id se incremente secuencialmente según el ultimo valor agregado.
- Metodo PUT: Actualizar datos de la BD. Puede codificarlo para modificar solo uno o mas campos de un elemento especifico de la tabla, o puede codificarlo para que modifique todos los campos de un elemento de la tabla.
- Metodo DELETE: Borrar datos de la BD. Codificiarlo para que borre un elemento especifico (según id) de la tabla my_movies.

Los entregable finales son :

- Link del codigo Github
- Capturas de ejecucion del docker run
- Capturas de pruebas GET y POST usando Postman.

El repositorio debe tener los siguientes archivos:

- main.py
- models.py (optional)
- routes.py(optional)
- controllers/movies_controllers.py (optional)
- Dockerfile
- requirements.txt
- .gitignore
- README.md (con instrucción del dockerizado de la BD)

En el repositorio no debe estar archivos .env, llaves, tokens, contraseñas, datos en csv, archivos pesados, o ambientes virtuales.

Los criterios que se tomaran para considerar la tarea como completa son los siguiente:

1. Usar controlador. Tener una carpeta de controllers donde este el controlador usado. El controlador debe tener la logica de los metodos GET, POST, PUT y DELETE (optional) .
   Si elije no crear el file controlador, entonces deberá agregar la funcionalidad de este punto en el archivo main.py
2. Archivo o carpeta de routes: Tener el archivo routes.py (optional)
3. Archivo main.py : Dentro del archivo main.py solo debe estar el inicializador de la app FastAPI o Flask. (optional) Si elije no usar el archivo main.py unicamente como inicializador del app, entonces deberá agregar todo el codigo en este file.
4. Archivo o carpeta models : Dentro del archivo models.py debe de estar el esquema de los datos de entrada y salida (optional).
   Si elije no crear el file models, entonces deberá agregar la funcionalidad de este punto en el archivo main.py
5. Uso de Postman : Enviar capturas del uso de los metodos GET y POST dentro de Postman para los endpoints creados.
6. Archivo requirements.txt : Se crea el archivo requirements.txt donde se incluyen todas la librerias/frameworks utilizados. Este servira para instalar estas mismas librerias en otros entornos.
7. Dockerfile : Se toma formato de los ejercicios en clase y se crea un archivo Dockerfile que servira de instrucciones para crear la imagen Docker para el API.
8. Repositorio Github : Se utiliza Github para crear un repositorio remoto. Se vincula este repositorio remoto al repositorio local
9. Guardar codigo en Github: Se utiliza git para guardar el codigo en un repositorio.

10. **Pruebas de funcionalidad API con BD** : Enviar captura del uso de docker run para el API desplegado localmente y como los request realizan cambios o extraen datos de la BD.
11. **Archivo README**. Se documenta el proyecto realizado.

OPCION 2 : Desarrollo de API para predicciones ML

Para este proyecto, deberán utilizar Docker (ya sea localmente o en la nube) para ejecutar un contenedores:

- API usando Flask o FastAPI desarrollado por el estudiante que permita realizar predicciones usando un modelo ML

En base al entregable del modulo de Machine Learning, se utilizará el modelo obtenido del entrenamiento ML usando el dataset de la siguiente competicion de Kaggle: https://www.kaggle.com/competitions/spaceship-titanic/data .Ademas se deberá crear una lista o diccionario Python que simulará una BD. La lista deberá permitir de Input un id (autoescalable o no), y los atributos de entrada que requiera el modelo ML; y de Output, un id id (autoescalable o no), los atributos de entrada usados para la prediccion, y la salida de la prediccion.

El API desarrollado debe implementar los métodos GET, POST, PUT y DELETE, y cada uno de ellos deberá realizar lo siguiente:

- **Metodo GET:** Leer datos de predicciones pasadas. Puede codificarlo para obtener solo un datos de alguna fila con un id especifico, o puede codificarlo para obtener todas las predicciones (toda la lista o diccionario).
- **Metodo POST:** Realizar una prediccion. Codificarlo para que agregar datos a la lista o diccionario Python, dandole como entrada los datos al modelo ML y guardando los datos de entrada y la prediccion resultante.
- **Metodo PUT:** Actualizar prediccion. Codificarlo para modificar uno o mas campos de un elemento especifico de la lista o diccionario Python y realizar la ejecucion ML para guardar la nueva salida.
- **Metodo DELETE:** Borrar predicciones pasadas. Codificiarlo para que borre un elemento especifico (según id) de la lista o diccionario Python.

Los entregable finales son :

- Link del codigo Github
- Capturas de ejecucion del docker run
- Capturas de pruebas GET y POST usando Postman.

El repositorio debe tener los siguientes archivos:

- main.py
- models.py (optional)
- routes.py(optional)
- controllers/prediccion_controllers.py (optional)
- Dockerfile
- requirements.txt
- .gitignore
- README.md

En el repositorio no debe estar archivos .env, llaves, tokens, contraseñas, datos en csv, archivos pesados, modelos .pkl o .h5, o ambientes virtuales.

Los criterios que se tomaran para considerar la tarea como completa son los siguiente:

1. Usar controlador. Tener una carpeta de controllers donde este el controlador usado. El controlador debe tener la logica de los metodos GET, POST, PUT y DELETE (optional)
   Si elije no crear el file controlador, entonces deberá agregar la funcionalidad de este punto en el archivo main.py
2. Archivo o carpeta de routes: Tener el archivo routes.py(optional)
3. Archivo main.py : Dentro del archivo main.py solo debe estar el inicializador de la app FastAPI o Flask (optional). Si elije no usar el archivo main.py unicamente como inicializador del app, entonces deberá agregar todo el codigo en este file.
4. Archivo o carpeta models : Dentro del archivo models.py debe de estar el esquema de los datos de entrada y salida (optional).
   Si elije no crear el file models, entonces deberá agregar la funcionalidad de este punto en el archivo main.py
5. Uso de Postman : Enviar capturas del uso de los metodos GET y POST dentro de Postman para los endpoints creados.
6. Archivo requirements.txt : Se crea el archivo requirements.txt donde se incluyen todas la librerias/frameworks utilizados. Este servira para instalar estas mismas librerias en otros entornos.

7. **Dockerfile** : Se toma formato de los ejercicios en clase y se crea un archivo Dockerfile que servira de instrucciones para crear la imagen Docker para el API.
8. **Repositorio Github** : Se utiliza Github para crear un repositorio remoto. Se vincula este repositorio remoto al repositorio local
9. **Guardar codigo en Github**: Se utiliza git para guardar el codigo en un repositorio.
10. **Pruebas de funcionalidad API**: Enviar captura del uso de docker run para el API desplegado localmente y como los request realizan un prediccion ML
11. **Archivo README**. Se documenta el proyecto realizado.