

FastAPI Framework

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

FastAPI es un web framework moderno y rápido (de alto rendimiento) para construir APIs con Python 3.6+ basado en Python.

Sus características principales son:

- **Rapidez:** Alto rendimiento, al a par con NodeJS y Go (gracias a Starlette y Pydantic). Uno de los frameworks más rápidos.
- **Rápido de programar:** Incrementa la velocidad de desarrollo entre 200% y 300%.
- **Menores Errores:** Reduce los errores humanos (de programador) aproximadamente un 40%.
- **Intuitivo:** Gran soporte en los editores con auto completado en todas partes.
- **Fácil:** Está diseñado para ser fácil de usar y aprender. Gastando menos tiempo leyendo documentación.
- **Corto:** Minimiza la duplicación de Código. Múltiples funcionalidades con cada declaración de parámetros. menos errores.
- **Robusto:** Crea código listo para producción con documentación automática interactiva.
- **Basado en estándares:** Basado y totalmente compatible con los estándares abiertos para APIs: OpenAPI (conocido previamente como Swagger) y JSON schema.

Instalaciones necesarias:

In []:

```
# pip install uvicorn[standard]
```

In []:

```
# pip install fastapi
```

Importar las librerías que vamos a usar:

In []:

```
# from fastapi import FastAPI, status, HTTPException
# import pandas as pd
# import json
# import csv
# import os
# from pydantic import BaseModel
```

Creamos nuestra primera aplicación con FastAPI:

In []:

```
# app = FastAPI()
```

Doy la ruta donde se encuentra el dataset:

In [2]:

```
# MEDIA_ROOT = "iris.csv"
```

Método GET a la url "/test/"

- Llamaremos a nuestra aplicación (<nombre aplicación> + <método permitido>)

In []:

```
# @app.get("/test/")  
# async def test_1():  
#     return "Bienvenido a FastAPI"
```

Para ejecutar la aplicación debemos ir a la ruta donde se encuentra el script:

- DataScience --> Backend --> main.py

In [3]:

```
# Ejecutamos con:  
# uvicorn main:app --reload
```

Copiaremos la url que nos indica al ejecutarlo:

INFO: Will watch for changes in these directories: ['/home/isabel/FEI_projects/DataScience/BigData/Backend']

INFO: Uvicorn running on <http://127.0.0.1:8000> (<http://127.0.0.1:8000>) (Press CTRL+C to quit)

INFO: Started reloader process [17990] using WatchFiles

INFO: Started server process [17992]

INFO: Waiting for application startup.

INFO: Application startup complete.

Probaremos nuestra aplicación poniendo la url "/test/":

- <http://127.0.0.1:8000/test/> (<http://127.0.0.1:8000/test/>)

Mostrando el mensaje de **Bienvenido a FastAPI**

También tenemos la opción de testear nuestro Backend usando Swagger:

- <http://127.0.0.1:8000/docs> (<http://127.0.0.1:8000/docs>)

default

GET /iris/ Iris

Parameters

No parameters

Execute

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

"string"

Método Get a la url "/iris"

In []:

```
# @app.get("/iris/")
# async def iris():
#     # Crear el dataframe con la información de iris:
#     # df = pd.read_csv(MEDIA_ROOT)
#     # print(df)
#     # lo transformamos a json para poder gestionarlo desde el front:
#     # data = df.to_json(orient="index")
#     # cargar la información con formato Json:
#     # data = json.loads(data)
#     # return data
```

Nos muestra la información del Iris dataset

Método Post a la url "/insertData"

Usamos el BaseModel importado de Pydantic para la creación del modelo de los datos:

In []:

```
# Modelo de datos:
# class Iris(BaseModel):
#     # sepal_length: float
#     # sepal_width: float
#     # petal_length: float
#     # petal_width: float
#     # species: str
```

Decalramos el método Post a la url "insertData", leemos el archivo iris.csv y añadimos a la última fila del dato correspondiente a los datos del docs.

In []:

```
# Método POST a la url "/insertData/"
# @app.post("/insertData/", status_code=201)
# async def insertData(item: Iris):
#     # leemos el archivo iris.csv e
#     # insertar en la última línea los campos a insertar
#     # with open(MEDIA_ROOT, "a", newline="") as csvfile:
#         # Nombres de los campos:
#         # fieldnames = ['sepal_length', 'sepal_width',
#             # 'petal_length', 'petal_width', 'species']
#     # escribimos el csv:
#     # writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
#     # insertamos los valores en la última fila:
#     # writer.writerow({"sepal_length": item.sepal_length,
#         # "sepal_width": item.sepal_width,
#         # "petal_length": item.petal_length,
#         # "petal_width": item.petal_width,
#         # "species": item.species})
#     # retornamos los valores que comprende la ultima fila añadida
#     return item
```

Para testear el backend realizamos la petición con los datos en formato JSON:

```
{ "sepal_length": 4.6, "sepal_width": 4.0, "petal_length": 6.8, "petal_width": 0, "species": "Test" }
```

Si es correcto nos responde con un 201 creado y el json enviado.

In []: