

Creado por:

Isabel Maniega

# FastAPI Framework - API Rest

<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.

## IMPORTANTE

Seguir siempre el protocolo para los distintos métodos:

- **GET:** Muestra información
- **POST:** Inserta información
- **PUT:** Actualiza la información
- **DELETE:** Elimina información

Sigue el llamado **CRUD** (Create, Read, Update, Delete)

Instalaciones necesarias:

In [ ]:

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

In [ ]:

```
# pip install fastapi
```

Importar las librerías que vamos a usar:

In [ ]:

```
# from fastapi import FastAPI, status, Response
# 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/`

- Nos muestra información
- 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 --> BigData --> Backend --> main.py

Para navegar por las carpetas usaremos:

- cd nombre de la carpeta

Ejemplo: `cd BigData/Backend`

Para asegurarnos que tenemos el archivo main.py usamos:

- LINUX/MAC --> ls
- Windows --> dir

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>).

**FastAPI** 0.1.0 OAS3  
/openapi.json

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
```

Pasos:

- Desplegar el botón de GET
- Pulsar Try it out
- Vemos el boton azul de execute y lo pulsaremos.

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
```

Declaramos el método Post a la url "insertData", leemos el archivo iris.csv y añadimos a la última fila del dato correspondiente a los datos que hemos enviado a través de 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
```

**FastAPI** 0.1.0 OAS3

/openapi.json

default

GET /test/ Test 1

GET /iris/ Iris

POST /insertData/ Insertdata

Schemas

HTTPValidationError >

Iris {

- sepal\_length\* Sepal Length > [...]
- sepal\_width\* Sepal Width > [...]
- petal\_length\* Petal Length > [...]
- petal\_width\* Petal Width > [...]
- species\* Species > [...]

}

Modelo de Datos (clase Iris)

ValidationError >

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.

POST /insertData/ Insertdata

Parameters Cancel Reset

No parameters

Request body required application/json

```
{
  "sepal_length": 8.7,
  "sepal_width": 4.0,
  "petal_length": 6.8,
  "petal_width": 0,
  "species": "Test"
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/insertData/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "sepal_length": 8.7,
    "sepal_width": 4.0,
    "petal_length": 6.8,
    "petal_width": 0,
    "species": "Test"
  }'
```

Request URL

```
http://127.0.0.1:8000/insertData/
```

Server response

Code	Details
201	<p>Response body</p> <pre>{   "sepal_length": 8.7,   "sepal_width": 4,   "petal_length": 6.8,   "petal_width": 0,   "species": "Test" }</pre> <p>Response headers</p> <pre>content-length: 92 content-type: application/json date: Wed, 05 Oct 2022 12:25:35 GMT server: uvicorn</pre>

## Método PUT a la url "/updateData/"

In [ ]:

```
# Método PUT a la url "/updateData/"
# llamaremos a nuestra aplicación (app.put)
# @app.put("/updateData/{item_id}")
# async def updateData(item_id: int, item:Iris):
#     Leemos el csv con ayuda de pandas:
#     df = pd.read_csv(MEDIA_ROOT)
#     Modificamos el último dato con los valores que nos lleguen:
#     df.loc[df.index[-1], "sepal_length"] = item.sepal_length
#     df.loc[df.index[-1], "sepal_width"] = item.sepal_width
#     df.loc[df.index[-1], "petal_length"] = item.petal_length
#     df.loc[df.index[-1], "petal_width"] = item.petal_width
#     df.loc[df.index[-1], "species"] = item.species
#     convertir a csv
#     df.to_csv(MEDIA_ROOT, index=False)
#     Retornamos el id que hemos modificado y el dato en formato diccionario:
#     return {"item_id": item_id, **item.dict()}
```

Una vez que damos a intentar nos habilita dos campos necesarios:

- **item\_id**: donde introduciremos el dato a modificar
- **Request Body**: donde introduciremos los nuevos campos.

PUT

/updateData/{item\_id}

Updatedata

^

Parameters

Cancel

Reset

Name	Description
item_id <small>required</small>	
integer	152
(path)	

Request body required

application/json

```
{
  "sepal_length": 3.5,
  "sepal_width": 0.6,
  "petal_length": 1.0,
  "petal_width": 4.6,
  "species": "xxxxxxx"
}
```

Execute

Clear

Responses

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/updateData/152' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "sepal_length": 3.5,
    "sepal_width": 0.6,
    "petal_length": 1.0,
    "petal_width": 4.6,
    "species": "xxxxxxx"
  }'
```

Request URL

http://127.0.0.1:8000/updateData/152

Server response

Code	Details
200	<div><div>Response body</div><pre>{   "item_id": 152,   "sepal_length": 3.5,   "sepal_width": 0.6,   "petal_length": 1,   "petal_width": 4.6,   "species": "xxxxxxx" }</pre><div><div>Download</div></div></div>

Response headers

La respuesta por parte del servidor es un diccionario con el item\_id del dato a modificar y los datos modificados

### Método DELETE a la url "/deleteData/"

In [ ]:

```
# Método DELETE para eliminar ese último dato del archivo.
# Método DELETE a la url "/deleteData/"
# @app.delete("/deleteData/{item_id}")
# async def deleteData(item_id: int):
#     Leemos el csv con ayuda de pandas:
#     df = pd.read_csv(MEDIA_ROOT)
#     Eliminar la última fila:
#     df.drop(df.index[-1], inplace=True)
#     convertir a csv
#     df.to_csv(MEDIA_ROOT, index=False)
#     return {"item_id": item_id, "msg": "Eliminado"}
```

Una vez que damos a intentar nos habilita un campo necesario:

- **item\_id**: donde introduciremos el dato a eliminar

**DELETE** /deleteData/{item\_id} Deletedata

Parameters

Name	Description
item_id <sup>required</sup> integer (path)	152

Execute Clear

Responses

Curl

```
curl -X 'DELETE' \
'http://127.0.0.1:8000/deleteData/152' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/deleteData/152
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "item_id": 152,   "msg": "Eliminado" }</pre> <p>Response headers</p> <pre>content-length: 33 content-type: application/json date: Thu, 06 Oct 2022 09:55:03 GMT server: uvicorn</pre>

La respuesta por parte del servidor es un diccionario con el `item_id` del dato a eliminado y un mensaje de eliminado

Creado por:

Isabel Maniega



