2- FastAPI Practise

By: Saleh Alsaeed
eng. Esraa Madhi

To begin using FastAPI, the first step is to install the required libraries, namely FastAPI itself and Unicorn. You can achieve this by using the following pip command:

```
pip install fastapi uvicorn pydantic
```

It's important to note that the number of libraries you install depends on the specific requirements of your project, and there may be additional dependencies beyond these two.

Getting Started!

1. **Setting Up Your Project**

Start by creating a new folder for your project. Within this folder, create a file named `main.py`. This file will contain the initial FastAPI code.

---

2. **Writing Your First FastAPI Code**

In the `main.py` file, add the following code:

```
from fastapi import FastAPI


app = FastAPI()


@app.get("/")
def root():
  return "Welcome To Tuwaiq Academy"
```

This simple FastAPI application creates an instance of the FastAPI class and defines a single route ("/") that returns a welcome message.

---

3. Running Your FastAPI Server Locally

To run your FastAPI server locally, open your command line in the same path of the file`main.py`and enter the following command:

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

The `main` refers to the name of your Python file (`main.py`), and `app` is the name of the FastAPI instance in your code. The `--reload` flag enables automatic reloading of the server when changes are made to the code.

- Optional: `--host` **and** `--port`:
  - Specify the host and port on which your FastAPI server will run. For example:

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

  - This command runs the server on all available network interfaces (`0.0.0.0`) and sets the port to `8000`.

- Optional: **--workers**`:
  - Configure the number of worker processes to handle incoming requests. For example:

```
uvicorn main:app --reload --workers 4
```

  - This command starts the server with 4 worker processes. The number can be adjusted based on your server's requirements and available resources.

- Optional: `--log-level`:
  - Set the log level for the server. Options include `critical`, `error`, `warning`, `info`, and `debug`. For example:

```
uvicorn main:app --reload --log-level debug
```

  - This command sets the log level to `debug`, providing more detailed logs for debugging purposes.

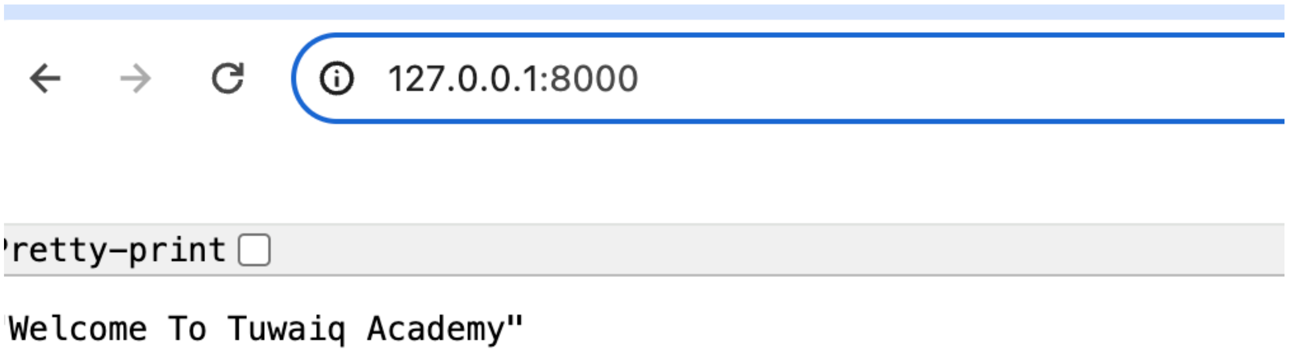
---

4. Accessing Your FastAPI Application

Open your web browser and navigate to the following URL:

- `http://127.0.0.1:8000` or `http://localhost:8000`

You should see your FastAPI application's welcome message displayed in the browser. This confirms that your FastAPI server is up and running locally.

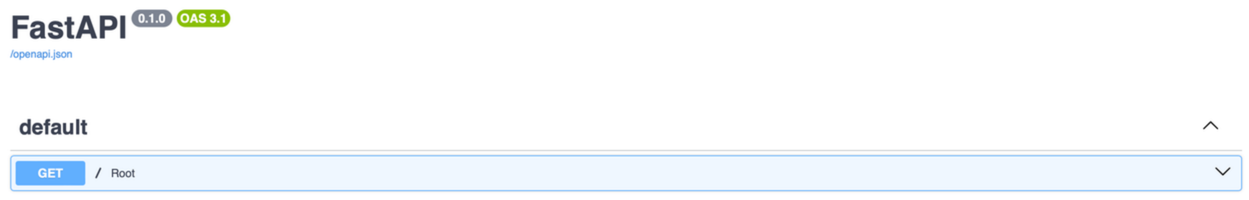

127.0.0.1:8000

'retty-print

```
Welcome To Tuwaiq Academy"
```

Interactive Documentation:
FastAPI generates interactive documentation automatically, which can be accessed at `/docs` and `/redoc`:

- **Swagger UI (`/docs`):**
    - Visit `http://127.0.0.1:8000/docs` or `http://localhost:8000/docs` in your browser.
    - You'll see an interactive UI where you can explore your API, send requests, and view responses.
    - The interactive documentation is generated based on the types and comments in your code.



Defining Different HTTP Methods:

```
from fastapi import FastAPI, HTTPException

app = FastAPI()

# GET request
@app.get("/")
def read_root():
    return {"message": "Welcome to Tuwaiq Academy"}

# get request
@app.get("/items/")
def create_item(item: dict):
    return {"item": item}
```

FastAPI makes it simple to create API endpoints with various HTTP methods. Let's extend the example by adding a POST endpoint for creating a new item and a PUT endpoint for updating an existing item.

In this example:

- The `create_item` endpoint accepts a POST request with a JSON payload representing an item.

- The `update_item` endpoint accepts a PUT request with a path parameter (`item_id`) and a JSON payload representing the updated item.

FastAPI for Machine Learning Deployment

1. **Save your model and any scaler:**

```
import joblib
joblib.dump(model, 'knn_model.joblib')
joblib.dump(scaler, 'Models/scaler.joblib')
```

Go to the notebook of knn model or any model you built it before and run those 2 lines at the end of the notebook

2. **Load Your Model in the API script:** Start by loading your trained machine learning model into your application in `main.py` file.

```
import joblib
model = joblib.load('knn_model.joblib')
scaler = joblib.load('Models/scaler.joblib')
```

3. **Preprocessing input data:** Implement functions to preprocess the incoming data into a format your model expects.

- Try to receive input first (add the next 3 blocks code in `main.py` file.

```
from pydantic import BaseModel


# Define a Pydantic model for input data validation
class InputFeatures(BaseModel):
    Year: int
    Engine_Size: float
    Mileage: float
    Type: str
    Make: str
    Options: str
```

```
def preprocessing(input_features: InputFeatures):
    dict_f = {
            'Year': input_features.Year,
            'Engine_Size': input_features.Engine_Size,
            'Mileage': input_features.Mileage,
            'Type_Accent': input_features.Type == 'Accent',
            'Type_Land Cruiser': input_features.Type == 'Land
Cruiser',
            'Make_Hyundai': input_features.Make == 'Hyundai',
            'Make_Mercedes': input_features.Make == 'Mercede
s',
            'Options_Full': input_features.Options == 'Full',
            'Options_Standard': input_features.Options == 'Sta
ndard'
        }
    return dict_f
```

```
@app.get("/predict")
def predict(input_features: InputFeatures):
    return preprocessing(input_features)
```

- In your terminal, run the following request:

```
curl -X GET "http://localhost:8000/predict" \
     -H "Content-Type: application/json" \
     -d '{
          "Year": 2020,
          "Engine_Size": 2.5,
          "Mileage": 15000,
          "Type": "Accent",
          "Make": "Hyundai",
          "Options": "Full"
         }'
```

- Modify **preprocessing** function to do scaling for input data:

```
def preprocessing(input_features: InputFeatures):
    dict_f = {
            'Year': input_features.Year,
            'Engine_Size': input_features.Engine_Size,
            'Mileage': input_features.Mileage,
            'Type_Accent': input_features.Type == 'Accent',
            'Type_Land Cruiser': input_features.Type == 'Land
Cruiser',
            'Make_Hyundai': input_features.Make == 'Hyundai',
            'Make_Mercedes': input_features.Make == 'Mercede
s',
            'Options_Full': input_features.Options == 'Full',
            'Options_Standard': input_features.Options == 'Sta
ndard'
        }

    # Convert dictionary values to a list in the correct order
    features_list = [dict_f[key] for key in sorted(dict_f)]

    # Scale the input features
    scaled_features = scaler.transform([list(dict_f.values
())])
```

```
    return scaled_features
```

4. **Create Prediction Endpoint:** Define an API endpoint that receives input data, processes it, and returns predictions made by your model.

```
@app.post("/predict")
async def predict(input_features: InputFeatures):
    data = preprocessing(input_features)
    y_pred = model.predict(data)
    return {"pred": y_pred.tolist()[0]}
```

- In your terminal, run the following request:

```
curl -X POST "http://localhost:8000/predict" \
     -H "Content-Type: application/json" \
     -d '{
          "Year": 2020,
          "Engine_Size": 2.5,
          "Mileage": 15000,
          "Type": "Accent",
          "Make": "Hyundai",
          "Options": "Full"
         }'
```

5. Follow steps in **"Host your ML application.pdf"**

Deployment into Docker Container:

- https://towardsdatascience.com/step-by-step-approach-to-build-your-machine-learning-api-using-fast-api-21bd32f2bbdb

- https://dev.to/code_jedi/machine-learning-model-deployment-with-fastapi-and-docker-llo
- https://engineering.rappi.com/using-fastapi-to-deploy-machine-learning-models-cd5ed7219ea

---

Resources:

- https://fastapi.tiangolo.com/tutorial/path-params/
- https://www.tutorialspoint.com/fastapi/fastapi_rest_architecture.htm
- https://medium.com/@reza.shokrzad/fastapi-the-modern-toolkit-for-machine-learning-deployment-af31d72b6589
- https://www.datacamp.com/tutorial/introduction-fastapi-tutorial
- https://dorian599.medium.com/ml-deploy-machine-learning-models-using-fastapi-6ab6aef7e777