AI in Web Programming

LECTURE 5 – FASTAPI

What is FastAPI?

- FastAPI is a well-known web framework for creating APIs with Python, leveraging standard Python type hints.
- It is user-friendly and straightforward, allowing developers to build production-ready applications quickly.
- Additionally, it offers full compatibility with OpenAPI and JSON Schema.

Why Excessive Growth of Usage for FastAPI with Machine Learning?

- Machine learning projects often comprise data scientists who primarily have expertise in statistics.
- These professionals may lack experience in software development or in deploying applications to deliver their ML projects.
- FastAPI provides data scientists with a straightforward way to create APIs for their projects.

Advantages of using FastAPI

- Compared to other Python web frameworks, FastAPI is simple yet fully functional.
- Mainly using decorators and type hints, it allows you to build a web application without the complexity of building a whole ORM (object-relational mapping) model and with the flexibility of using any database, including any SQL and NoSQL databases.
- FastAPI also provides automatic documentation generation, support for additional information and validation for query parameters, and good async support.

Advantages of using FastAPI (Cont.)

1 - Fast Development

- Creating API calls in FastAPI is as easy as adding decorators in the Python code.
- Minimal to no prior backend development experience is required for individuals seeking to transform a Python function into an application capable of handling API requests.

Advantages of using FastAPI (Cont.)

2 - Fast Documentation

- FastAPI provides automatic interactive API documentation using Swagger UI, which is an industry standard.

Advantages of using FastAPI (Cont.)

3 - Easy Testing

- Writing tests is one of the most important steps in software development, but it can also be one of the most tedious, especially when the time of the data scientist is valuable.

Advantages of using FastAPI (Cont.)

4 - Fast Deployment

- FastAPI comes with a CLI tool that can bridge development and deployment smoothly.
- It allows you to switch between development mode and production mode easily.
- Once development is completed, the code can be easily deployed using a Docker container with images that have Python prebuilt.

Getting Start with FastAPI

- **Uvicorn** is an ASGI (Asynchronous Server Gateway Interface) server that runs the FastAPI application.
- FastAPI apps need an ASGI server to handle asynchronous requests, and Uvicorn is a lightweight and fast server that works well with FastAPI's asynchronous capabilities.
- To install:
- pip install fastapi uvicorn

Run Uvicorn

- uvicorn main:app –reload
- This will start the server on http://127.0.0.1:8000. You can test the endpoints in your browser or use tools like **Postman** or **curl**.
- FastAPI also provides an interactive API documentation at http://127.0.0.1:8000/docs.

Create a Greeting Server

1-Check the current hour and returns a different greeting based on the time:

- Morning: 5 AM to 12 PM
- Afternoon: 12 PM to 6 PM
- Evening: 6 PM to 5 AM

Greet the Client

```python
from fastapi import FastAPI
from datetime import datetime

app = FastAPI()

def get_time_based_greeting():
    current_hour = datetime.now().hour
    if 5 <= current_hour < 12:
        return "Good morning"
    elif 12 <= current_hour < 18:
        return "Good afternoon"
    else:
        return "Good evening"

@app.get("/")
def time_based_greeting():
    greeting = get_time_based_greeting()
    return {"message": f"{greeting}!"}
```

Greet the Client

```python
@app.get("/")
def time_based_greeting():
    greeting = get_time_based_greeting()
    return {"message": f"{greeting}!"}

# Personalized greeting with time-based message
@app.get("/greet/{name}")
def greet_name(name: str):
    greeting = get_time_based_greeting()
    return {"message": f"{greeting}, {name}!"}
```

Run code2

```python
# main.py
from fastapi import FastAPI
from pydantic import BaseMode
from typing import List

app = FastAPI()

# In-memory storage for items
items = []

# Define a model for the item
class Item(BaseModel):
    name: str
    description: str = None
    price: float
```

Run the code

```
17  @app.get("/")
18  async def read_root():
19      return {"message": "Welcome to the FastAPI app in AI in Web Class!"}
20
21  @app.get("/items", response_model=List[Item])
22  async def get_items():
23      return items
24
25  @app.post("/items", response_model=Item)
26  async def create_item(item: Item):
27      items.append(item)
28      return item
```

To Test

- In terminal run:
- uvicorn main:app –reload
- From browser run:
- 1-http://127.0.0.1:8000/
- 2-http://127.0.0.1:8000/items